The λ-Calculus

1 Introduction

The normal system of mathematics is based in set theory and recursively defines everything in terms of sets. In particular, the standard Zermelo–Fraenkel axioms of set theory define functions in terms of sets [Cie97, §§2.1–2.2]. It is natural to ask if we could define mathematics in another way, using something else as a basis. Could we, instead of considering collections of arbitrary objects, instead consider transformations of arbitrary objects[1]? The answer is yes, and there are two main frameworks for this approach. In this essay we will consider λ-calculus, instead of the closely-related combinatory logic.

We want a purely functional universe—that is, a system in which everything is a 'pure' function. A pure function, in this case, takes one pure function as input and returns one pure function. So in this world, it's functions all the way down.

Modern computers are heavily based on the idea of a *Turing machine*, which is a system for computing functions. Working with and programming a Turing machine[2] is an arduous task, where the programmer has to keep the entire machine in their head and understand its inner workings. Lisp and ALGOL, programming languages invented in the late 1950s, revolutionised the act of programming a computer and turned it into something that a child could do. Most modern programming languages can trace their roots back to Lisp or ALGOL, and both take heavy inspiration from λ-calculus [AFT12, p. 4].

2 Syntax

Definition 2.1 (λ-term)**.** The λ-calculus consists of λ-*terms*, defined in the following recursive way [HS08, p. 3]:

(1) All variables (labelled $a, b, \dots, z, a_1, b_1, \dots$) are λ-terms, called *atoms*.

[1] This essay is about *untyped* λ-calculus. There are variations of the theory with types added on, where the objects are less 'arbitrary'.

[2] A 'real' Turing machine has an infinitely long tape to store data, which is clearly impossible. By 'working with a Turing machine', we mean working with the finite real-world analog, sometimes called a 'von Neumann machine' [AFT12, p. 4].

(2) If M and N are any two λ-terms then (MN) is a λ-term. This is called *application*.

(3) If M is any λ-term and x is any variable then $(\lambda x.M)$ is a λ-term. This is called *abstraction*.

The reason that Church choose the symbol λ has been lost to time. It is either derived from the notation $\hat{x}$ used by Whitehead and Russell, which became $\wedge x$ and then λx for ease of printing, or the symbol was chosen basically at random [CH06, p. 6].

In this essay we will use various sizes of parentheses to ease parsing for the reader, but these different sizes have no affect on the semantics.

Definition 2.2 (Scope)**.** In a λ-term $\lambda x.M$, the occurrence of M is called the *scope* of the occurrence of λx.

Definition 2.3 (Free and bound variables)**.** Let M be an arbitrary λ-term. A variable x is *free* in M if x is not in the scope of any λx; otherwise, x is *bound* [Bar81, p. 24]. If we're considering the x immediately proceeding the λ (that is, the first x in $\lambda x.x$), it is called *bound and binding* [HS08, p. 6–7].

We define $\mathrm{FV}(M)$ as the set of all free variables in the expression M. A λ-term with no free variables is called *closed*.

Example 2.1 (Free and bound variables)**.**

(1) In $x(\lambda y.xy)$, x occurs free twice and y occurs bound twice, one of which is also binding.

(2) In $y(\lambda y.y)$, y occurs free once and bound twice. This is confusing and discouraged in practice, because the y in the parentheses doesn't refer to the same thing as the y outside, but we allow this syntax to keep the definitions simple.

Notice that 'λx' binds in the same way as '$\forall x$' in familiar predicate logic.

3 Conversion and reduction

Given λ-terms M and N and a variable x, we write $M[x := N]$ to mean a new λ-term which is M with all occurrences of x replaced with N [Bar81, p. 27][3]. For example,

$$(\lambda xy.zxz)[z := (\lambda a.a)] \to (\lambda xy.(\lambda a.a)x(\lambda a.a)).$$

[3]Other notations include '$S_N^x M$|' [Chu41, p. 9; CR36] and '$[N/x]M$' [HS08, p. 7; CF58, §**2**C2].

Definition 3.1 (α-conversion)**.** Let a λ-term P contain an occurrence of $\lambda x.M$ and let $y \notin \mathrm{FV}(M)$. The act of replacing the $\lambda x.M$ with $\lambda y.M[x := y]$ is called *α-conversion*.

If a λ-term Q can be reached by a finite (possibly empty) sequence of α-conversions, we say that P *α-converts to* Q and write $P \equiv_\alpha Q$ [HS08, p. 9].

Example 3.1 (α-conversion)**.**

(1) $\lambda x.x \equiv_\alpha \lambda y.y$

(2) $\lambda x.t \equiv_\alpha \lambda y.t$

(3) $\lambda x.y \not\equiv_\alpha \lambda y.y$ because $y \in \mathrm{FV}(y)$. The LHS in this case is the constant function that always returns y, and the RHS is the identity function. These are not the same, so it doesn't make sense to call them equivalent in the sense of α-conversion.

(4) $\lambda abc.bc(ab)c\big(a(cb)\big) \equiv_\alpha \lambda xyz.yz(xy)z\big(x(zy)\big)$

(5) $\lambda g_2 j a_{16}.a_{16}(jg_2)a_{16} \equiv_\alpha \lambda xyz.z(yx)z$

Definition 3.2 (β-contraction)**.** Any term of the form $(\lambda x.M)N$ is called a *β-redex* and the corresponding term $M[x := N]$ is called its *contractum*.

If a λ-term P contains an occurrence of $(\lambda x.M)N$ then we can replace that occurrence by $M[x := N]$. We can call the result P' and then we say that P *β-contracts to* P', and we write $P \to_\beta P'$ [HS08, pp. 11–12; Bar81, p. 23].

Definition 3.3 (β-reduction)**.** If a λ-term Q can be reached by a finite (possibly empty) sequence of β-contractions, we say that P *β-reduces to* Q and write $P \twoheadrightarrow_\beta Q$ [HS08, pp. 11–12]. Formally, $\twoheadrightarrow_\beta$ is the *reflexive, transitive closure* of $\to_\beta$ meaning:

(1) $M \to_\beta N$ implies $M \twoheadrightarrow_\beta N$;

(2) $M \twoheadrightarrow_\beta M$;

(3) if $M \twoheadrightarrow_\beta N$ and $N \twoheadrightarrow_\beta L$ then $M \twoheadrightarrow_\beta L$ [Bar81, p. 51].

Definition 3.4 (β-normal form)**.** A λ-term Q which contains no β-redexes is said to be in *β-normal form*, sometimes written *β-nf*. If $P \twoheadrightarrow_\beta Q$ then Q is called a *β-normal form of* P [HS08, p. 12; Bar81, p. 34].

Example 3.2 (β-contraction)**.** Let N be an arbitrary λ-term.

(1) $(\lambda x.x)N \to_\beta N$

(2) $(\lambda x.y)N \to_\beta y$

(3) $(\lambda xy.y)N \to_\beta \lambda y.y$

(4) $\big(\lambda xy.x(\lambda z.xzy)\big)(\lambda ab.baa) \to_\beta \lambda y.(\lambda ab.baa)(\lambda z.(\lambda ab.baa)zy)$
$\to_\beta \lambda y.(\lambda ab.baa)(\lambda z.(\lambda b.bzz)y)$
$\to_\beta \lambda y.(\lambda ab.baa)(\lambda z.yzz)$
$\to_\beta \lambda y.\big(\lambda b.b(\lambda z.yzz)(\lambda z.yzz)\big)$
$\equiv \lambda yb.b(\lambda z.yzz)(\lambda z.yzz)$

(5) $\big(\lambda xyz.x(zyx)\big)(\lambda ab.bba) \to_\beta \lambda yz.(\lambda ab.bba)\big(zy(\lambda ab.bba)\big)$
$\equiv_\alpha \lambda yz.(\lambda ab.bba)\big(zy(\lambda cd.ddc)\big)$
$\to_\beta \lambda yz.\Big(\lambda b.bb\big(zy(\lambda cd.ddc)\big)\Big)$
$\equiv \lambda yzb.bb\big(zy(\lambda cd.ddc)\big)$

This example uses α-conversion to rename a to c and b to d after the first β-contraction. Without this renaming, we would get confusing (but technically valid) variable bindings:

$$\lambda yzb.bb\big(zy(\lambda ab.bba)\big)$$

In this case, the b inside the innermost brackets is completely unrelated to the b outside, so giving them different names makes the expression easier to understand.

(6) $\big(\lambda xyz.xz(yz)\big)\big((\lambda xy.yx)u\big)\big((\lambda xy.yx)v\big)w$
$\to_\beta \big(\lambda xyz.xz(yz)\big)(\lambda y.yu)\big((\lambda xy.yx)v\big)w$
$\to_\beta \big(\lambda xyz.xz(yz)\big)(\lambda y.yu)(\lambda y.yv)w$
$\equiv_\alpha \big(\lambda xyz.xz(yz)\big)(\lambda a.au)(\lambda b.bv)w$
$\to_\beta \big(\lambda yz.(\lambda a.au)z(yz)\big)(\lambda b.bv)w$
$\to_\beta \Big(\lambda z.(\lambda a.au)z\big((\lambda b.bv)z\big)\Big)w$
$\to_\beta (\lambda a.au)w\big((\lambda b.bv)w\big)$
$\to_\beta wu\big((\lambda b.bv)w\big)$
$\to_\beta wu(wv)$

Note that this example, as well as the previous three, are all in β-normal form.

(7) $(\lambda x.xx)(\lambda x.xx) \to_\beta (\lambda x.xx)(\lambda x.xx)$
$\to_\beta (\lambda x.xx)(\lambda x.xx)$
$\to_\beta \cdots$

This example shows that the β-contraction process doesn't always simplify. This term is called Ω. Since Ω contains β-redexes and $\Omega \to_\beta \Omega$, we see that Ω is an example of a λ-term with no β-normal form. But

Ω is 'minimal' in the sense that it cannot be reduced to any different term [HS08, p. 13].

(8) $$\begin{aligned}(\lambda x.xxy)(\lambda x.xxy) &\to_\beta (\lambda x.xxy)(\lambda x.xxy)y \\ &\to_\beta (\lambda x.xxy)(\lambda x.xxy)yy \\ &\to_\beta \cdots\end{aligned}$$

In this example, the β-contraction process actually makes the expression more 'complicated' (in the sense of containing more terms). The initial term $(\lambda x.xxy)(\lambda x.xxy)$ is called $\mathbf{L}$, and it has no β-normal form [HS08, p. 12].

(9) Let $P \equiv (\lambda a.b)\mathbf{L}$. Then P can be reduced in multiple ways, by β-contracting either $(\lambda a.b)$ or $\mathbf{L}$ [HS08, pp. 11–12]:

(i) $$\begin{aligned}P &\equiv (\lambda a.b)\mathbf{L} \\ &\to_\beta b[a := \mathbf{L}] \\ &\equiv b\end{aligned}$$

(ii) $$\begin{aligned}P &\equiv (\lambda a.b)\mathbf{L} \\ &\to_\beta (\lambda a.b)\mathbf{L}y \\ &\to_\beta (\lambda a.b)\mathbf{L}yy \\ &\to_\beta \cdots\end{aligned}$$

So P has a β-normal form b, but it also has an infinite reduction.

It is important to note that all of the $\to_\beta$ in these examples could've been written with $\twoheadrightarrow_\beta$. Note that $P \twoheadrightarrow_\beta Q$ does not imply that Q is a β-nf, nor even that $Q \not\equiv P$, as seen in the case of $\Omega \twoheadrightarrow_\beta \Omega$.

Definition 3.5 (Rule ξ)**.** We now introduce a simple premise which is historically given the name *rule* ξ [Bar81, p. 23; HS08, p. 70]:

$$M = N \implies \lambda x.M = \lambda x.N. \tag{ξ}$$

In normal mathematics, function equality is 'extensional', meaning it cares about what the function does, not how it is defined internally. In normal maths, if $\forall x, f(x) = g(x)$, we say that $f = g$.

Currently, our system of λ-calculus is 'intensional', meaning it doesn't have this property. For example, Let $F = y$ and $G = \lambda x.yx$. Then $FX = GX$ for any λ-term X, but currently $F \neq G$ [HS08, p. 76].

We can extend our theory of λ-calculus to make it extensional in two useful ways, which turn out to be equivalent.

Definition 3.6 (Weak extensionality)**.** Let M and N be λ-terms and $x \notin \mathrm{FV}(MN)$. Then we say [Bar81, p. 31]

$$Mx = Nx \implies M = N. \tag{ζ}$$

Definition 3.7 (η-conversion)**.** Let M and N be λ-terms and $x \notin \mathrm{FV}(M)$. Then we say [Bar81, p. 32].

$$\lambda x.Mx = M. \tag{η}$$

Theorem 3.1. *Weak extensionality and η-conversion are equivalent.*

Proof. We want to show that if we add weak extensionality, we can derive the η-conversion rule, and vice versa[4].

Consider λ-calculus with weak extensionality. The statement

$$(\lambda x.Mx)x = Mx$$

is of course true. If $x \notin \mathrm{FV}(M)$ then we can apply (ζ) to get

$$\lambda x.Mx = M,$$

which is (η).

Conversely, consider λ-calculus with η-conversion. Let $Mx = Nx$ with $x \notin \mathrm{FV}(MN)$. Then by rule ξ,

$$\lambda x.Mx = \lambda x.Nx.$$

Then we apply (η) and get $M = N$. □

It doesn't matter which of these extensionality axioms we choose to add, but most literature uses η-conversion, so we will do the same.

Definition 3.8 (η-reduction)**.** Any term of the form $\lambda x.Mx$ with $x \notin \mathrm{FV}(M)$ is called a *η-redex* and its *contractum* is M.

The phrases *P η-contracts to Q* and *P η-reduces to Q* are defined in the same way as we defined the β equivalents in Definitions 3.2 and 3.3, and we instead write $P \to_\eta Q$ and $P \twoheadrightarrow_\eta Q$ respectively.

4 Arithmetic

Definition 4.1 (Church numerals)**.** [Chu41, p. 28; HS08, p. 48] We define a system of *Church numerals* as:

$$\begin{aligned}
\overline{0} &:= \lambda fx.x,\\
\overline{1} &:= \lambda fx.fx,\\
\overline{2} &:= \lambda fx.f(fx),\\
\overline{3} &:= \lambda fx.f(f(fx)),
\end{aligned}$$

etc. We also define a successor function

$$\overline{\sigma} := \lambda abc.b(abc).$$

[4]Proof taken from [Bar81, p. 32].

Example 4.1 (Successor)**.**

$$\begin{aligned}\overline{\sigma}\,\overline{2} &\equiv \big(\lambda abc.b(abc)\big)\big(\lambda fx.f(fx)\big)\\ &\to_\beta \lambda bc.b\Big(\big(\lambda fx.f(fx)\big)bc\Big)\\ &\to_\beta \lambda bc.b\Big(\big(\lambda x.b(bx)\big)c\Big)\\ &\to_\beta \lambda bc.b\big(b(bc)\big)\\ &\equiv_\alpha \lambda fx.f(f(fx))\\ &\equiv \overline{3}\end{aligned}$$

Definition 4.2 (Arithmetic operations)**.** [Chu41, p. 10] We define addition as

$$\overline{m} + \overline{n} := \lambda ab.(\overline{m}a)(\overline{n}ab),$$

multiplication as

$$\overline{m} \times \overline{n} := \lambda a.\overline{m}(\overline{n}a),$$

and exponentiation as

$$\overline{m}^{\overline{n}} := \overline{n}\,\overline{m}.$$

Example 4.2 (Arithmetic)**.**

$$\begin{aligned}\overline{2} + \overline{3} &\equiv \lambda ab.\big(\overline{2}a\big)\big(\overline{3}ab\big)\\ &\equiv \lambda ab.\Big(\big(\lambda fx.f(fx)\big)a\Big)\Big(\big(\lambda fx.f(f(fx))\big)ab\Big)\\ &\to_\beta \lambda ab.\big(\lambda x.a(ax)\big)\Big(\big(\lambda fx.f(f(fx))\big)ab\Big)\\ &\to_\beta \lambda ab.\big(\lambda x.a(ax)\big)\big(a(a(ab))\big)\\ &\to_\beta \lambda ab.a(a(a(a(ab))))\\ &\equiv_\alpha \lambda fx.f(f(f(f(fx))))\\ &\equiv \overline{5}\end{aligned}$$

$$
\begin{aligned}
\overline{2} \times \overline{3} &\equiv \lambda a.\overline{2}\big(\overline{3}a\big) \\
&\equiv \lambda a.\big(\lambda fx.f(fx)\big)\Big(\big(\lambda fx.f(f(fx))\big)a\Big) \\
&\to_\beta \lambda a.\big(\lambda fx.f(fx)\big)\big(\lambda x.a(a(ax))\big) \\
&\equiv_\alpha \lambda a.\big(\lambda fx.f(fx)\big)\big(\lambda y.a(a(ay))\big) \\
&\to_\beta \lambda a.\lambda x.\big(\lambda y.a(a(ay))\big)\Big(\big(\lambda y.a(a(ay))\big)x\Big) \\
&\equiv \lambda ax.\big(\lambda y.a(a(ay))\big)\Big(\big(\lambda y.a(a(ay))\big)x\Big) \\
&\to_\beta \lambda ax.\big(\lambda y.a(a(ay))\big)\big(a(a(ax))\big) \\
&\to_\beta \lambda ax.a(a(a(a(ax)))) \\
&\equiv_\alpha \lambda fx.f(f(f(f(fx)))) \\
&\equiv \overline{6}
\end{aligned}
$$

$$
\begin{aligned}
\overline{2}^{\overline{3}} &\equiv \overline{3}\,\overline{2} \\
&\equiv \big(\lambda fx.f(f(fx))\big)\big(\lambda fx.f(fx)\big) \\
&\equiv_\alpha \big(\lambda fx.f(f(fx))\big)\big(\lambda gy.g(gy)\big) \\
&\to_\beta \lambda x.\big(\lambda gy.g(gy)\big)\Big(\big(\lambda gy.g(gy)\big)\big(\big(\lambda gy.g(gy)\big)x\big)\Big) \\
&\to_\beta \lambda x.\big(\lambda gy.g(gy)\big)\Big(\big(\lambda gy.g(gy)\big)\big(\lambda y.x(xy)\big)\Big) \\
&\equiv_\alpha \lambda x.\big(\lambda gy.g(gy)\big)\Big(\big(\lambda gy.g(gy)\big)\big(\lambda z.x(xz)\big)\Big) \\
&\to_\beta \lambda x.\big(\lambda gy.g(gy)\big)\Big(\lambda y.\big(\lambda z.x(xz)\big)\Big(\big(\lambda z.x(xz)\big)y\Big)\Big) \\
&\to_\beta \lambda x.\big(\lambda gy.g(gy)\big)\Big(\lambda y.\big(\lambda z.x(xz)\big)\big(x(xy)\big)\Big) \\
&\to_\beta \lambda x.\big(\lambda gy.g(gy)\big)\big(\lambda y.x(x(x(xy)))\big) \\
&\equiv_\alpha \lambda x.\big(\lambda gy.g(gy)\big)\big(\lambda z.x(x(x(xz)))\big) \\
&\to_\beta \lambda x.\lambda y.\big(\lambda z.x(x(x(xz)))\big)\Big(\big(\lambda z.x(x(x(xz)))\big)y\Big) \\
&\equiv \lambda xy.\big(\lambda z.x(x(x(xz)))\big)\Big(\big(\lambda z.x(x(x(xz)))\big)y\Big) \\
&\to_\beta \lambda xy.\big(\lambda z.x(x(x(xz)))\big)\big(x(x(x(xy)))\big) \\
&\to_\beta \lambda xy.x(x(x(x(x(x(xy)))))) \\
&\equiv_\alpha \lambda fx.f(f(f(f(f(f(fx)))))) \\
&\equiv \overline{8}
\end{aligned}
$$

5 Confluence

Confluence, generally, is a property of rewriting systems in which the order of reductions is unimportant. For example, in normal arithmetic, we could evaluate the expression $(2+3)\times(5+1)$ by simplifying the left parenthetical and then the right, or right and then left. Either way, we get the same answer.

We want λ-calculus to be confluent because we want one term to always produce the same result, no matter what order its β-redexes were contracted. $\overline{1}+\overline{1}$ should always β-reduce to $\overline{2}$, regardless of the inner workings of the reduction. If λ-calculus is confluent, it would allow us to ignore these inner workings for much of the theoretical work that we want to do with the system.

Theorem 5.1 (The Church–Rosser Theorem for β-reduction)**.** *If $P \twoheadrightarrow_\beta M$ and $P \twoheadrightarrow_\beta N$, then $\exists T$ such that $M \twoheadrightarrow_\beta T$ and $N \twoheadrightarrow_\beta R$. See Figure 1.*

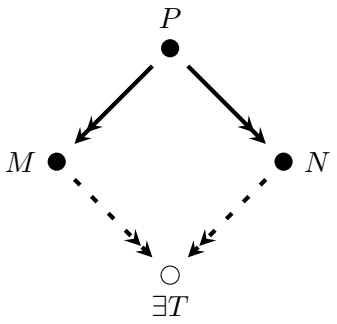

Figure 1: The 'diamond property' of confluence[5].

This was first proved by Church and Rosser in 1936 [CR36] but the proof we will use is attributed to William Tait and Per Martin-Löf [Bar81, pp. 59–62; HS08, §A2A].

Ideally, we want break this down into a series of β-contractions and do some diagram chasing, as seen in Figure 2. If we could prove that β-contraction is confluent, that is

$$P \to_\beta M \text{ and } P \to_\beta N \implies \exists T \text{ such that } M \to_\beta T \text{ and } N \to_\beta T,$$

then we could chain steps of single β-contractions together. We would first prove Theorem 5.1 in the special case that $P \to_\beta M$ by induction on the length of the reduction to N, as seen on the left of Figure 2. Then we could deduce that general case by induction on the length of the reduction to M, as seen on the right side of the diagram.

[5]Reproduced from Fig. A2:1 in [HS08, p. 282].
[6]Reproduced from Fig. A2:2 in [HS08, p. 283].

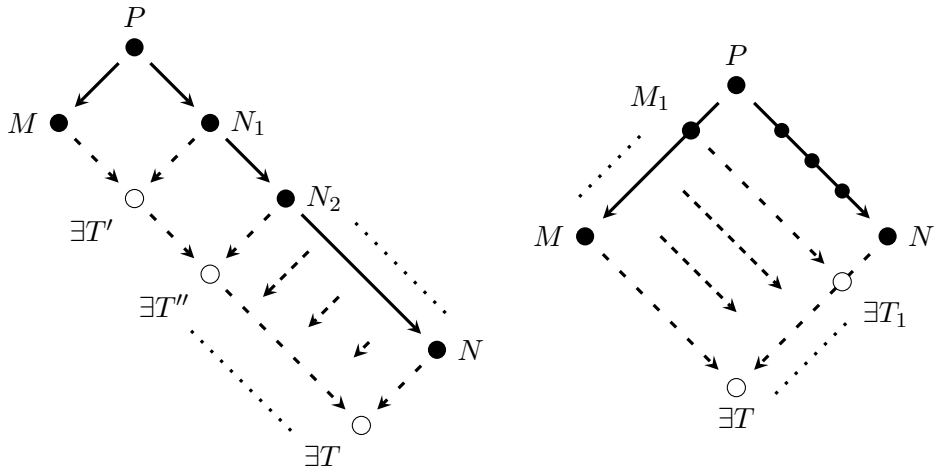

Figure 2: The desired diagram chase[6].

The problem is that this doesn't work. Consider for example $P := (\lambda y.uyy)(\mathbf{I}z)$ where $\mathbf{I} = \lambda x.x$. Then $P \to_\beta u(\mathbf{I}z)(\mathbf{I}z)$ and $P \to_\beta (\lambda y.uyy)z \to_\beta uzz$. In this case we have $M := u(\mathbf{I}z)(\mathbf{I}z)$ but M cannot be reduced to uzz in just one contraction. It can, however, be reduced using two non-overlapping, 'parallel' contractions. It turns out that we can define a new relation $\twoheadrightarrow_\parallel$ (parallel reduction) whose transitive closure is the same as that of $\twoheadrightarrow_\beta$. Then if $\twoheadrightarrow_\parallel$ satisfies the diamond property, then so does $\twoheadrightarrow_\beta$ and we have proven the Church–Rosser Theorem.

Unfortunately, the obvious definition of $\twoheadrightarrow_\parallel$ as simultaneous non-overlapping contractions doesn't work, but we can define parallel reduction in a more subtle way.

Definition 5.1 (Residuals)**.** Let a λ-term P contain β-redexes R and S. Let P' be the result of β-contracting R in P, so $P \to_\beta P'$. The *residuals* of S with respect to R are β-redexes in P', defined as follows:

Case 1. R and S are non-overlapping parts of P. Then β-contracting R leaves S unchanged. We call the unchanged S in P' the *residual* of S.

Case 2. $R \equiv S$. Then β-contracting R is the same as β-contracting S. We say that S has *no residual* in P'.

Case 3. R is a proper part of S, meaning R is a part of S and $R \not\equiv S$. Then S has form $(\lambda x.M)N$ and R is in M or in N. Then β-contracting R changes M to M' or N to N'. This changes S to $(\lambda x.M')N$ or $(\lambda x.M)N'$ in P'. We call this the *residual* of S.

Case 4. S is a proper part of R. Then R has the form $(\lambda x.M)N$ and S is in M or in N. There are subcases in this instance for whether S is in M or in N [HS08, pp. 284–285] but case 4 is not needed for our confluence proof, so we shall ignore it from now on.

Definition 5.2 (Minimal residual)**.** Let $n \geq 0$ and let $R_1, \ldots, R_n$ be β-redexes in a λ-term P. An R_i is called *minimal* in $\{R_1, \ldots, R_n\}$ if and only if none of the other $R_1, \ldots, R_n$ is a proper part of R_i.

Definition 5.3 (Parallel reduction)**.** Let $n \geq 0$ and let $R_1, \ldots, R_n$ be β-redexes in a λ-term P. A *parallel reduction*[7] of $\{R_1, \ldots, R_n\}$ in P is a β-reduction obtained by first β-contracting a minimal R_i, then a minimal residual of $R_1, \ldots, R_n$, and continuing to β-contract minimal residuals until no residuals are left, and finally perhaps doing some α-conversions.

If a parallel reduction changes P to a λ-term Q, we say $P \twoheadrightarrow_{\|} Q$.

Note: The rest of this proof is long and complicated. If I include it, I'm worried I won't have space for interesting things like combinators. If I don't include it, I'm worried that I won't have a nice focal point of the essay, or I won't have enough mathematical depth.

[7] Parallel reductions were sometimes called *minimal complete reductions* [HS86, p. 315].

References

[AFT12] Andrew W. Appel, Solomon Feferman and Alan M. Turing. *Alan Turing's Systems of Logic: The Princeton Thesis*. English. Princeton, New Jersey: Princeton University Press, 2012. ISBN: 9780691155746.

[Bar81] Hendrik P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. English. Ed. by Kenneth J. Barwise et al. 1st ed. Vol. 103. Studies in Logic and the Foundations of Mathematics. Amsterdam: North-Holland, 1981. ISBN: 0444854908.

[CH06] Felice Cardone and Roger Hindley. 'History of Lambda-calculus and Combinatory Logic'. English. In: 2006. URL: https://www.di.unito.it/~felice/pdf/lambdacomb.pdf.

[Chu41] Alonzo Church. *The Calculi of Lambda-Conversion*. English. Reprinted in 1965 by Kraus Reprint Corporation. Princeton: Princeton University Press, 1941.

[CR36] Alonzo Church and John B. Rosser Sr. 'Some Properties of Conversion'. English. In: *Transactions of the American Mathematical Society* 39.3 (1936), pp. 472–482. ISSN: 0002-9947, 1088-6850. DOI: 10.2307/1989762.

[Cie97] Krzysztof Ciesielski. *Set Theory for the Working Mathematician*. English. Vol. 39. Cambridge: Cambridge University Press, 1997. ISBN: 9781139173131. DOI: 10.1017/CBO9781139173131.

[CF58] Haskell B. Curry and Robert Feys. *Combinatory Logic, Volume I*. English. Amsterdam: North-Holland, 1958. ISBN: 9780720422078. URL: https://archive.org/details/combinatorylogic0001curr.

[HS86] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-calculus*. English. Vol. 1. London Mathematical Society Student Texts. Cambridge: Cambridge University Press, 1986. ISBN: 0521318394.

[HS08] J. Roger Hindley and Jonathan P. Seldin. *Lambda-Calculus and Combinators: An Introduction*. English. 2nd ed. Cambridge: Cambridge University Press, 2008. ISBN: 9780511809835. DOI: 10.1017/CBO9780511809835.