Lambda Calculus

University of Connecticut

Daniel Byrne

0.1 Introduction

Lambda calculus is a model of computation. It is an abstract system that can do all computation that a computer can do .Such models of computation are important to have because programming languages must be tailored to actual usability and not mathematical rigor. The best example of this is the print statement, which is a key part of programming languages. It is useless in the analysis of the program because it can only access the output that was previously computed. It does no real extra computing. Moreover, it allows us to explore the limitations of computation. The best example of the limitation of computation is the halting problem. Is a computer able to determine if a program will halt before the program is run? The answer to that is no, but to even approach an answer to this question one must have a theoretical way of thinking about computation.

The most famous such model of computation is the Turing machine, which is a finite set of rules and an infinite tape. This model of computation is hard to deal with in practice. Nobody would ever program something through the paradigm of a Turing machine. However, lambda calculus is much easier to deal with. This is because it has baked into it the idea of inputs and recursion, self-reference. This means that writing a λ-term to raise a number to some power is actually quite simple. Furthermore, the structure of λ-terms is much more inductive, so one can apply inductive arguments to say things about all λ-terms. This does not work with Turing machines because they do not have the same structure. Lastly, the greater emphasis on recursion is a different viewpoint on computation, which allows for different insights in general. Normally, we emphasize the sequential nature of computation. It is an action followed by another action. However, recursion is a totally different way of thinking. Such ideas have even been implemented into programming languages, which have some strong benefits not seen in procedural or even object oriented languages.

In this piece, we will define the lambda calculus, and prove the most important property of this model, the Church Rosser property. We will then prove its equivalence to regular computation and hence Turing machines. Lastly, we will prove some statements about the format and structure of λ-terms, which will allow us to introduce the idea of the Böhm tree, which is a key tool used in the study of lambda calculus..

0.2 Basics of Lambda Calculus

0.2.1 Axioms

The language of lambda terms is defined by the following rules.

1. The alphabet of Lambda Calculus, over which words are defined, consists of the following symbols.
   (a) Variables: $v_0, v_1, ...$
   (b) Abstractor: λ
   (c) Paranthesis: (,)
2. The set of λ terms, Λ can be combined inductively under the following rules.

(a) $x \in \Lambda$

(b) $M \in \Lambda \Rightarrow (\lambda x M) \in \Lambda$

(c) $M, N \in \Lambda \Rightarrow (MN) \in \Lambda$

3. In the following, M and N will be used for arbitrary terms while x,y, and z... denote arbitrary variables

4. According to this definition every lambda term should be surrounded by parentheses. However, it is not customary to put parentheses around the whole lambda term just subparts of it. These parentheses around the whole term do not change the meaning of the term and are also redundant as we already know that it is a lambda term.

The first thing that we will define on this language will be the idea of syntactic equality. This is a binary relation that only compares the strings of the terms and does not relate at all to the meaning of the term.

This idea of syntactic equality is defined by the following axioms, and it will be denoted by $\equiv$.

1. M $\equiv$ M (reflexivity)

2. M $\equiv$ N $\implies$ $N \equiv M$ (symmetry)

3. M $\equiv$ N, N $\equiv$ L $\implies$ M $\equiv$ L (Transitivity)

4. M $\equiv$ N $\implies$ MZ $\equiv$ NZ

5. M $\equiv$ N $\implies$ ZM $\equiv$ ZM

The first three ensure that this is an equivalence relation. The last two ensure that appending some arbitrary term to two already syntactically equivalent terms in the same place does not destroy syntactic equivalence.

We can also define equality in other ways by adding new axioms.

The two most important examples of these extra axioms are β-conversion and ξ-conversion.

Definition 1. $\beta-$**conversion** is defined by $(\lambda x.M)N = M[x := N]$ where $M[x := N]$ denotes replacing every instance of the variable x in M with the term N.

Example 1. Let $M \equiv (\lambda x.x)y$ and $N \equiv y$. According to the axioms of syntactic equality, these two terms are not equivalent. However, if we use the rule of β-reduction, we can convert M into a form that is syntactically equivalent to N. $(\lambda x.x)y =_\beta x[x := y] = y$. Since both are now just the variable y, they are syntactically equivalent.

This can cause some problems with the naming of different variables, which will be dealt with in the next section.

Definition 2. ξ**-conversion** is defined by $M = N \implies \lambda x.M = \lambda x.N$.

We need this extra axiom because $\equiv$ is defined only upon combinations of λ-terms and appending some term with $\lambda x.$ is not the same as adding a term because $\lambda x.$ is not a valid term alone.

From this situation, we do have a slight problem in that from equality we cannot figure out what rules were applied to get there. For example, if we allow for both the ξ-converion and β-conversion rules to be a part of our system, then we could not tell if two terms are equivalent because of some sequence of both rules or just because of repeated application of one rule. Moreover, we might just want to get terms that are the result of repeated or singular application of β-conversion to prove things about the results of such actions. However, with equality we cannot tell which term reduces to which or if either term reduces to each other at all. Therefore, we cannot group together terms that are the result of β-converion on some term.

To combat this problem we introduce the binary relations $\rightarrow$ and $\twoheadrightarrow$.

Definition 3. $\rightarrow$ This is defined as one application of any of the deduction rules. It is formally defined as

1. $(\lambda x.M)N \rightarrow M[x := N]$
2. $M \rightarrow N \implies ZM \rightarrow ZN$
3. $M \rightarrow N \implies MZ \rightarrow NZ$
4. $M \rightarrow N \implies \lambda x.M \rightarrow \lambda x.N$

$\twoheadrightarrow$ is the transitive and reflexive extension of $\rightarrow$. Therefore, it is defined as

1. $M \rightarrow N \implies M \twoheadrightarrow N$
2. $M \twoheadrightarrow M$
3. $M \twoheadrightarrow N, N \twoheadrightarrow L \implies M \twoheadrightarrow L$

Such ideas can be extended to an arbitrary relation/deduction rule but that shall not be done here.

Definition 4.

1. A term in a reduction is a **redux** if it is the term we apply the reduction rule to. Hence, if $M \rightarrow N$, then M is the redux of N.
2. A term is a **contractum** if it is the result of a single application of a reduction rule. If $M \rightarrow N$, then N is the contractum.
3. A term N is in **normal form** if $N \twoheadrightarrow M \implies N \equiv M$. There is also the **head normal form** which is defined as M being in the form $\lambda \overrightarrow{x}.y\overrightarrow{N}$

0.2.2 Syntactic Notions

Definition 5. A variable,x, is **free** in a $\lambda-$term M if x is not in the scope of a λx. Otherwise x is **bound**.

Definition 6. FV(M) is the set of free variables in M and is defined as follows.

1. FV(x) = {x}
2. FV($\lambda x.M$)= FV(M) - {x}
3. FV(MN) = FV(M) $\cup$ FV(N)

Λ_0 is the set of all lambda terms with no free variables.

It should be noted that all terms are treated as the same modulo a change in bound variables but not with a change of free variables. For example, $\lambda x.x = \lambda y.y$, but $x \neq y$. We want to consider bound variables as simply placeholders for future inputs according to β-conversion. This is much the same as with normal functions. $f(z) = z^2$ in an equivalent function to $f(x) = x^2$, but x and z are treated as distinct objects by themselves that are not equal to each other.

Example 2. For $M \equiv x\lambda y.xy$, we say that x appears free twice while y appears bounded once.

Definition 7. A **context**, $C[]$ is a term with holes in it. It can be filled by placing any term in the hole. It is different from substitution in that variables in the term we place in the context can be become bound after we place them in the context.

0.2.3 Substitution

We want to treat variables in the most naive way possible, but that can cause problems if one does not institute rules on the naming of variables.

1. All bound variables are chosen to be distinct from all free variables. This means a free variable can never be converted into a bound variable by a reduction. If we did not say this, then $(\lambda x.\lambda y.x)y = \lambda y.y$. However, those two terms are actually different because $(\lambda x.\lambda y.x)MN = M$ but $(\lambda y.y)MN = MN$.

2. Terms that are α-congruent are treated the same where **α-congruence** , $\equiv_\alpha$, is defined as being congruent up to a change in the variables used to represent the bound variables in a lambda term. One can also treat this as an extra axiom upon = given by $\lambda x.M = \lambda y.M[x := y]$ where y does not occur free or bound in M. All that we care about is how the inputs are permuted for closed lambda terms. We only care what the function does and not how it looks according to some arbitrary variable labeling system.

   (a) $\lambda x.xy \equiv_\alpha \lambda z.zy \not\equiv_\alpha \lambda y.yy$.

   (b) $\lambda x.x(\lambda x.x) \equiv_\alpha \lambda x', x'(\lambda x.x) \equiv_\alpha \lambda x'.x'(\lambda x''.x'')$.

β-reduction is defined through this idea of substitution, which is axiomatically defined as follows.

Definition 8. Substitution, $M[x := N]$ where $M, N \in \Lambda$ is defined recursively as follows

$$x[x := N] \equiv N$$
$$y[x := N] \equiv y$$

if $x \not\equiv y$. (We must state that $x \not\equiv y$ because we never say that one variable object cannot be labeled two distinct ways. It would be ridiculous to label something like that, but we must cover all of the bases.)

$$(\lambda y.M_1)[x := N] \equiv \lambda y.M_1[x := N]$$
$$M_1M_2[x := N] \equiv M_1[x := N]M_2[x := N]$$

0.3 Important Lambda Calculus Terms

One great example of the power of lambda calculus is the encoding of the numerals and the normal operations on them such as addition and multiplication in terms of lambda terms.

1. **Number Encoding** Below is a standard approach to the encoding of the natural numerals in the lambda calculus called the Church numerals. However, it is not the only possible encoding of the numerals.

   (a) We encode some number n by $\lambda fx.f^nx$, so the encoding of 0 is $\lambda fx.x$, the encoding of 1 is $\lambda fx.fx$, and the encoding of 5 is $\lambda fx.fffffx$. Let $\ulcorner n \urcorner$ denote the church numeral encoding of n. The encoding of the numbers are in normal form to ensure no weird behavior which could be introduced by reductions.

   (b) The successor function S^+ is coded by the term $\lambda xyz.xz(yz)$. This means that when we apply S^+ to some number $\ulcorner n \urcorner$, we can reduce down to $\ulcorner n + 1 \urcorner$

   Example 3. Let us say we are given the encoding for 3 that is $\lambda fx.fffx$. To get 4 we apply the successor function to 3.

   $$\lambda xyz.y(xyz)(\lambda fx.fffx)$$
   $$\lambda yz.y(\lambda fx.fffx)yz$$
   $$\lambda yz.y(\lambda x.yyyx)z$$
   $$\lambda yz.yyyyz$$

   Because of our conventions, $\lambda yz.yyyyz$ is equivalent to $\lambda fx.fffffx$

   (c) The predecessor function is encoded by the term $P^- = \lambda x.x(\lambda xy.y)$.

   (d) Addition, A_+, is defined by $\lambda xypq.xp(ypq)$

Example 4. Let us say that we want to add 4 and 5 together. We then apply 4 and 5 to the A_+

$$\lambda xypq.xp(ypq)(\lambda fx.ffffx)(\lambda fx.fffffx)$$

$$\lambda ypq.(\lambda fx.ffffx)p(ypq)(\lambda fx.fffffx)$$

$$\lambda pq.(\lambda fx.ffffx)p((\lambda fx.fffffx)pq)$$

$$\lambda pq.(\lambda x.ppppx)((\lambda fx.fffffx)pq)$$

$$\lambda pq.pppp(\lambda fx.fffffx)pq$$

$$\lambda pq.pppp(\lambda x.ppppppx)q$$

$$\lambda pq.ppppppppppq$$

The result is then the encoded value of 9 as desired.

(e) Multiplication, A_*, is defined by $= \lambda abx.a(b(x))$.

It would be very good practice to convince yourself that both predecessor and multiplication also work.

2. Fixed point combinators. A **fixed point combinator** is a lambda term, M, such that for all F, $MF = F(MF)$. If we think of the MF term as a point and the F term as a function, we get that F with MF as input returns MF, which is the original pint, so MF is a fixed point of the function F. In order to prove a fixed point is a fixed point we try to reduce the MF term to the F(MF) term instead of the other way around.

   The reason for requiring making MF the fixed point and not just F is that we need to use the combinator as an input to prevent weird behavior.

   (a) **Y-combinator**,Y, also known as the paradoxical combinator. We can construct this by first defining $W \equiv \lambda x.F(xx)$ and $X \equiv WW$. This gives us the following reduction

   $$X \equiv (\lambda x.F(xx))W \to F(WW) \equiv FX.$$

   We can add an extra input to the X to fit it in the fixed-combinator formula. So, $Y \equiv \lambda f.(\lambda x.f(xx))\lambda x.f(xx))$. This leads to the path of

   $$YF = WW = F(WW) = F(YF)$$

   However, it is not the case the $WW \twoheadrightarrow YF$. Instead $YF \twoheadrightarrow WW$. That is what makes it the paradoxical combinator.

   (b) **Turing Combinator**, θ. This term is built up from $A \equiv \lambda xy.y(xxy)$ as $\theta \equiv AA$. Hence,

   $$\theta F = AAF = \lambda xy.y(xxy)AF \twoheadrightarrow F(AAF)$$

   exactly making it a fixed piont combinator. Moreover, the equality comes just from reduction, making it theoretically simpler.

3. **Truth and False**

   (a) **Truth**, $T \equiv \lambda xy.x$. This lambda term chooses the first of any two inputs. Therefore, it operates in the same way as an if statement would operate. When some lambda term reduces down to this, it simply chooses the first of two inputs. This is the same as with an if/else statement in a programming langauge.When the condition is true the first branch gets executed. Otherwise, the second branch gets executed. As one can see below the False combinator acts in the opposite way reinforcing this idea.

   This term is also known as K both for historical reasons and because it is so useful that it becomes convenient to use it outside of the concept of truth/false.

   (b) **False**, $F \equiv \lambda xy.y$. F chooses the second of any two inputs. It operates just as an if statement when the term in the if statement is false. Moreover, it is equivalent to 0, in the above numeral system, which is also another programming practice.

   (c) **Random Useful Constructions/Terms.** The following are very useful constructions

      i. **Identity Term** $I \equiv \lambda x.x$. This term simplifies down to its argument.

      ii. **Inductive Conjunction**.[$*$] We will define this pairing in three steps

$$[M] \equiv M$$

$$[M, N] \equiv \lambda z.zMN$$

$$[M_0, \ldots, M_{n+1}] \equiv [M_0, [M_1, \ldots M_{n+1}]]$$

For example, $[M_0, M_1, M_2] = [M_0, [M_1, M_2] = [M_0, \lambda z.M_1M_2] = \lambda z_1.M_0\lambda z_0.M_1M_2$

0.4 Church Rosser Property

We will now show that each lambda term has at most one normal form. This means that we do not have to care about where and when we applied β-reduction when reducing down to normal forms.

Lemma 1 (Substitution Lemma)**.** If $x \not\equiv y$ and $x \notin FV(L)$ $M[x := N][y := L] \equiv M[y := L][x := N[y := L]]$

Proof. We do this by induction on the structure of M.

1. M is a variable

   (a) If $M \equiv x$. then both sides equal $N[y := L]$ since $x \not\equiv y$

   (b) If $M \equiv y$, then both sides equal L because $x \notin FV(L) \implies L[x := \cdots] \equiv L$

   (c) If $M \equiv z$ and $z \not\equiv x, y$, then both sides equal z because x and y are not free so cannot be replaced.

2. If $M \equiv \lambda z.M_1$, then we can assume that $z \not\equiv x, y$ and that z is not free in N,L by the variable conventions. Applying the inductive hypothesis and the definition of substitution gives us

$$\begin{aligned}(\lambda z.M_1)[x := N][y := L] &\equiv \lambda z.M_1[x := N][y := L] \\ &\equiv \lambda z.M_1[y := L][x := N[y := L]] \\ &\equiv (\lambda z.M_1)[y := L][x := N[y := L]]\end{aligned}$$

.

3. If $M \equiv M_1 M_2$, then one can apply the inductive hypothesis to both M_1 and M_2.

□

Attached to this definition of substitution is the idea of substitutivity

Definition 9. A reduction rule R is **substiutive** if $\forall M, N, L \in \Lambda$ and all variables x one has $M \to_R N \implies M[x := L] = N[x := L]$

Lemma 2. β-reduction is substitutive.

Proof. Let $M \to_\beta N$. Then $M \equiv (\lambda y.P)Q$ and $N \equiv P[y := Q]$. Therefore,

$$M[x := L] \equiv (\lambda y.(P[x := L]))(Q[x := L])$$

$$N[x := L] \equiv P[y := Q][x := L]$$
$$N[x := L] \equiv P[y := Q[x := L]]$$

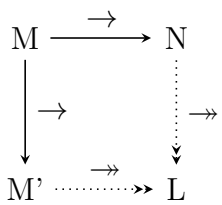
by the substitution lemma and our variable conventions. □

To help us prove that each lambda term has one normal form under β-reduction, we will first define some extra terms regarding reduction and then use induction by going through each possible type of reduction that can be applied to a term.

Definition 10. 1. The **Church Rosser property** says that if two elements are equal, then they reduce by β-reduction to the same term. Stated in quantitifies

$$M =_\beta N \implies \exists Z[M \twoheadrightarrow_\beta Z \wedge N \twoheadrightarrow_\beta Z]$$

. The **Weak Church Rosser property** is when $\to$ satiisfies the same property so $\forall M$



To make things a little bit easier we will define a modification of $\twoheadrightarrow_\beta$.

Definition 11. The binary relation $\twoheadrightarrow_1$ is defined inductively as follows.

1. $M \twoheadrightarrow_1 M$
2. $M \twoheadrightarrow_1 M' \implies \lambda x.M \twoheadrightarrow_1 \lambda x.M'$
3. $M \twoheadrightarrow_1 M'$ and $N \twoheadrightarrow_1 N' \implies MN \twoheadrightarrow_1 M'N'$
4. $M \twoheadrightarrow_1 M'$ and $N \twoheadrightarrow_1 N' \implies (\lambda x.M)N \twoheadrightarrow_1 M'[x := N']$

Proposition 1. If $M \twoheadrightarrow_1 M'$ and $N \twoheadrightarrow_1 N' \implies M[x := N] \twoheadrightarrow_1 M'[x := N']$

Proof. We will use induction on the definition of $M \twoheadrightarrow_1 M'$.

1. $M \twoheadrightarrow_1$ is $M \twoheadrightarrow_1 M$. We need to show that $M[x := N] \twoheadrightarrow_1 M[x := N']$, which we will do by induction on the structure of M.
   (a) If $M = x$, then need to show that $N \twoheadrightarrow_1 N'$, which is true by our hypothesis.
   (b) If $M = y$, then we need to show that $y \twoheadrightarrow_1 y$, which is trivially true.
   (c) If $M = PQ$, then by the inductive hypothesis we have $P[x := N] \twoheadrightarrow_1 P'[x := N']$ and $Q[x := N] \twoheadrightarrow_1 Q'[x := N']$. We can concatenate the terms by the definition of $\twoheadrightarrow_1$, which gives us $P[x := N]Q[x := N] \twoheadrightarrow_1 P'[x := N']Q'[x := N']$ which is the same as $M[x := N] \twoheadrightarrow M'[x := N']$.
   (d) If $M = \lambda y.P$, then by inductive hypothesis we have $P[x := N] \twoheadrightarrow_1 P'[x := N']$ and then by the definition of $\twoheadrightarrow_1$ we have $(\lambda x.P)[x := N] \twoheadrightarrow_1 (\lambda x.P')[x := N']$.
2. $M \twoheadrightarrow_1 M'$ is $\lambda y.P \twoheadrightarrow_1 \lambda y.P'$. This case directly follows from the second property of $\twoheadrightarrow_1$.
3. $M \twoheadrightarrow_1 M'$ is $PQ \twoheadrightarrow_1 P'Q'$. We apply the inductive hypothesis to both P and Q to get $P[x := N] \twoheadrightarrow_1 P'[x := N']$ and $Q[x := N] \twoheadrightarrow_1 Q'[x := N']$. It follows that $P[x := N]Q[x := N] \twoheadrightarrow_1 P'[x := N']Q'[x := N']$. Since $M[x := N] \equiv P[x := N]Q[x := N]$ and $M'[x := N'] \equiv P'[x := N']Q'[x := N']$, we have $M[x := N] \twoheadrightarrow_1 M'[x := N']$.
4. $M \twoheadrightarrow_1 M'$ is $(\lambda y.P)Q \twoheadrightarrow_1 P[y := Q]$.

$$\begin{aligned} M[x := N] \equiv (\lambda y.P[x := N])(Q[x := N]) &\twoheadrightarrow_1 P'[x := N'][y := Q'[x := N']] \\ &\equiv P'[y := Q'][x := N'] \\ &\equiv M'[x := N'] \end{aligned}$$

by an application of the induction hypothesis and the substitution lemma.

□

Proposition 2. 1. $\lambda x.M \twoheadrightarrow_1 N \implies N \equiv \lambda x.M'$ with $M \twoheadrightarrow_1 M'$.

2. $MN \twoheadrightarrow_1 L \implies L \equiv M'N'$ with $M \twoheadrightarrow_1 M'$ and $N \twoheadrightarrow_1 N'$ or $M \equiv \lambda x.P, L \equiv P'[x := N']$ and $P \twoheadrightarrow_1 P', N \twoheadrightarrow_1 N'$.

Proposition 3. $\twoheadrightarrow_1$ satisfies the diamond property(another name for the Church-Rosser property)

Proof. We will show that for all $M \twoheadrightarrow_1 M_1$ and $M \twoheadrightarrow_1 M_2$, there is a term M_3 such that $M_1 \twoheadrightarrow_1 M_3$ and $M_2 \twoheadrightarrow_1 M_3$ by induction on the definition of $\twoheadrightarrow_1$.

1. If $M \equiv M_1$, then we can take M_3 to be M_2 where P''' is the common reduct that P' and P'' have by the induction hypothesis.

2. If $M \twoheadrightarrow_1 M$ is $\lambda x.P \twoheadrightarrow_1 \lambda x.P' \equiv M_1$, then one can take $M_3 \equiv \lambda x.P'''$.

3. If $M \twoheadrightarrow_1 M_1$ is $PQ \twoheadrightarrow_1 P'Q'$, then there are two subscases

   (a) If $M_2 \equiv P''Q''$ with $P \twoheadrightarrow_1 P''$, $Q \twoheadrightarrow_1 Q''$, we can apply the inductive hypothesis on P and Q to get $M_3 \equiv P'''Q'''$.

   (b) If $P \equiv \lambda x.P_1$, $M_2 \equiv P''[x := Q'']$ with $P \twoheadrightarrow_1 P''$ and $Q \twoheadrightarrow_1 Q''$, one can take $M_3 \equiv P'''[x := Q''']$.

4. If $M \twoheadrightarrow_1 M_1$ is $(\lambda x.P)Q \twoheadrightarrow_1 P'[x := Q']$ with $P \twoheadrightarrow_1 P'$ and $Q \twoheadrightarrow_1 Q'$, then there are two subcases.

   (a) $M_2 \equiv (\lambda x.P'')Q''$ with $P \twoheadrightarrow_1 P''$ and $Q \twoheadrightarrow_1 Q''$. Here we apply the inductive hypothesis to get terms P''' and Q''' such that $P' \twoheadrightarrow_1 P'''$, $P'' \twoheadrightarrow_1 P'''$ and $Q' \twoheadrightarrow_1 Q'''$, $Q'' \twoheadrightarrow_1 Q'''$. This allows to take $M_3 \equiv P'''[x := Q''']$.

   (b) $M_2 \equiv P''[x := Q'']$. Here we use he same term, $M_3 \equiv P'''[x := Q''']$,which exists again by the inductive hypothesis.

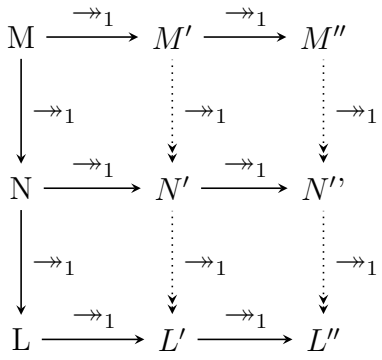
□

Lemma 3. $\twoheadrightarrow$ is the transitive closure of $\twoheadrightarrow_1$

Proof. The only difference between $\rightarrow$ and $\twoheadrightarrow_1$ is that $\twoheadrightarrow_1$ has two extra rules namely that $M \twoheadrightarrow_1 M$ and $M \twoheadrightarrow_1 M'$ and $N \twoheadrightarrow_1 N' \implies MN \twoheadrightarrow_1 M'N'$. However, that is equivalent to a combination of the first rule of $\twoheadrightarrow$ and the second and third rules of $\rightarrow$. Moreover, $M \twoheadrightarrow_1 M$ is already a rule in $\twoheadrightarrow$. Therefore, the only rule of $\twoheadrightarrow$ not in $\twoheadrightarrow_1$ is just the transitive rule. Since $\twoheadrightarrow_1$ contains no extra rules besides those in $\twoheadrightarrow$, $\twoheadrightarrow$ is the transitive closure of $\twoheadrightarrow_1$ □

Theorem 1. β-reduciton has the Church Rosser property.

Proof. This can be shown best through the following diagram.



□

Theorem 2. M can have at most one β-nf or normal form.

Proof. This just comes from the fact that the only term that a normal form can reduce to is itself, so M must reduce to this normal form by the Church Rosser property. □

It should be noted though that the single reduction binary relation$\rightarrow$ does not have the Church Rosser property. Instead it has a weaker property named the weak Church Rosser Property.

Definition 12. A reduction scheme is **weakly Church Rosser** if $[M \rightarrow N \wedge M \rightarrow L \implies \exists P[N \twoheadrightarrow P \wedge L \twoheadrightarrow P]]$. The term can also be used for any binary relation, in which case it is called the **weak diamond property** and is defined by $[M \rightarrow N \wedge M \rightarrow L \implies \exists P[N \rightarrow^*_= P \wedge L \rightarrow^*_= P$ where we take $\rightarrow^*_=$ as the transitive reflexive closure of the binary relation.

One might at first think that this is essentially the same as the Church Rosser property. However, the two are not equivalent. Instead this is a much more local property. It is only saying that after one reduction, there is a reduction path from the result of one reduction to another. However, if there is some way that after inifnitely many reduction, you reach the normal form, then Church Rosser is violated. This is because that infinite reduction path breaks through the local property.

Lemma 4.

1. The relations R and its reflexive closure $\rightarrow_=$ do not satisfy the Church Rosser Property.

2. The relation $\rightarrow$ satisfies the weak diamond property.

   Proof. For part 1, let $R \rightarrow R'$ and let R and R' not be in normal form. Let $M \equiv (\lambda x.xx)R$. We have two possible β reductions, namely $M \rightarrow RR$ and $M \rightarrow (\lambda x.xx)R'$. However, one cannot get either of those forms to the other in one reduction, so they do satisfy the relation $\rightarrow$.

   For part 2, let $M \rightarrow M_1$ and $M \rightarrow M_2$. If we can construct and M_3 such that $M_1 \twoheadrightarrow M_3$ and $M_2 \twoheadrightarrow M_3$, then we have shown that the weak diamond property is satisfied as $\twoheadrightarrow$ is the transitive reflexive closure of $\rightarrow$ by construction. Let $\Delta_i : M \rightarrow M_i$, i = 1,2 with $\Delta_i = (\lambda x_i.P_i)Q$ be the β-reductions from $M \rightarrow M_i$ and let $\Delta'_i \equiv P_i[x_i = Q_i$. The possible relative positions of Δ_1 and Δ_2 in M are the following

(a) $\Delta_1 \cup \Delta_2 = \emptyset$

(b) $\Delta_1 = \Delta_2$

(c) $\Delta_1 \subset \Delta_2$

  i. $\Delta_1 \subset P_2$

  ii. $\Delta_1 \subset Q_2$

(d) $\Delta_2 \subset \Delta_2$

  i. $\Delta_2 \subset P_1$

  ii. $\Delta_2 \subset P_1$

Case (a). We have

$$M \equiv \cdots \Delta_1 \cdots \Delta_2 \cdots$$

$$M_1 \equiv \cdots \Delta_1' \cdots \Delta_2 \cdots$$

$$M_2 \equiv \cdots \Delta_1 \cdots \Delta_2' \cdots$$

Therefore, we can just take $M_3 \equiv \cdots \Delta_1' \cdots \Delta_2' \cdots$
Case (b). $M_1 \equiv M_2$, so we can just take $M_3 \equiv M_1$
Case (c i). $M \equiv \cdots ((\lambda x_2 \cdots \Delta_1 \cdots) Q_2$ where $\cdots \Delta_1 \cdots \equiv P_2$

$$M_1 \equiv \cdots ((\lambda x_2 \cdots \Delta_1' \cdots) Q_2)$$

$$M_2 \equiv \cdots (\cdots \Delta_1 \cdots)[x_2 := Q_2]$$

Therefore, we can take

$$M_3 \equiv \cdots (\cdots \Delta_1' \cdots)[x_2 := Q_2] \cdots$$

Therefore, $M_1 \to M_3$ by construction, and $M_2 \to M_3$ by the substitutivity of β.
(Case c ii). $M \equiv \cdots ((\lambda x_2.P_2)(\cdots \Delta_1 \cdots)) \cdots$ where $\cdots \Delta_1 \cdots \equiv Q_2$

$$M_1 \equiv \cdots ((\lambda x_2.P_2)(\cdots \Delta_1' \cdots)) \cdots$$

$$M_2 \equiv \cdots (P_2[x_2 := (\cdots \Delta_1 \cdots)])$$

Hence, we can just take

$$M_3 \equiv \cdots (P_2[x_2 := (\cdots \Delta_1' \cdots)])$$

So, $M_1 \to M_2$ and $M_2 \twoheadrightarrow M_3$ because one can reduce Δ_1 in M_2 independent of the other terms.

□

0.5 Computability Theory

The most important aspect of lambda calculus is that it can do any possible computation. The requirements for a complete model of computation is that it is closed under total functions.

Definition 13. 1. A **numeric function** is a mapping $\phi : \mathbb{N}^p \to \mathbb{N}$.

2. Let ϕ be a numeric function with p arguments. ϕ is called **λ-definable** if for some $F \in \Lambda$, $\forall n_1, \ldots, n_p \in \mathbb{N}$ $\mathrm{F}\ulcorner n_1 \urcorner, \ldots \ulcorner n_p \urcorner = \ulcorner \phi(n_1, \ldots, n_p) \urcorner$. Therefore, for all possible inputs F applied to their encoding is equal to the encoding of the result of the function with those inputs.

3. The class $\mathcal{R}$ of **recursive functions** is the least class of functions which contains all initial functions and is closed under primitive recursion and minimilization, which we will define below.

Definition 14. The **initial functions** are the numeric functions $U_i^p, S^+,$ and Z.

The **projection function**, U_n^I, is defined by $U_i^p(n_0, \ldots, n_p) = n_i$ for $0 < i < p$. Therefore, it selects the i^{th} input of p inputs.

The **successor function**, S^+, is defined by $S^+(n) = n + 1$. It simply adds 1 to the input.

The **Zero function**, Z is defined as $Z(n) = 0$, so for every value it returns 0.

The **minimal value function**, μm is defined as $\mu m[P(n)]$ is the least value for which P(m) holds. If $P(m)$ is never true, then we leave it to be undefined. Ultimately, we construct it so that it is not 'solvable', which we will define later. This lack of solvability means that we make it a meaningless term.

These form as the basic ingredients of computation that we will then combine together. The following are the closure properties, so the ways that we can combine functions to create new functions.

Definition 15. Let $\mathscr{P}$ be a class of numeric functions

1. $\mathscr{P}$ is closed **under composition** if for all ϕ defined by

$$\phi(\overrightarrow{n}) = \chi(\psi_1(\overrightarrow{n}), \ldots, \psi_m(\overrightarrow{n}))$$

with $\chi, \psi_1, \ldots, \psi_m, \in \mathscr{P}$, one has $\phi \in \mathscr{P}$.

2. $\mathscr{P}$ is closed **under primitive recursion** if for all ϕ defined by

$$\phi(0, \overrightarrow{n}) = \chi(\overrightarrow{n})$$

$$\phi(k+1, \overrightarrow{n}) = \psi(\phi(k, \overrightarrow{n}), k, \overrightarrow{n})$$

with $\chi, \psi \in \mathscr{P}$, one has $\phi \in \mathscr{P}$.

3. $\mathscr{P}$ is closed **under minimilization** if for all ϕ

$$\phi(\overrightarrow{n}) = \mu m[\chi(\overrightarrow{n}, m) = 0]$$

with $\chi \in \mathscr{P}$ such that

$$\forall \overrightarrow{n} \exists m \quad \chi(\overline{n}, m) = 0$$

one has $\phi \in \mathscr{P}$.

Theorem 3. The total numeric terms that are definable by λ terms are exactly the recursive functions.

Proof. We will show that the recursive functions are all λ-definable

Initial Functions

Let

$$U_i^p \equiv \lambda x_0 \cdots x_n . x_i$$

$$S^+ \equiv \lambda x.[F, x]$$

$$Z \equiv \lambda x.\ulcorner 0 \urcorner$$

Here F is the false constructor.

One should verify for themselves that these functions are lambda definitions of each of the initial functions.

2. **Closed under Composition**
If we let $\chi, \psi_1, \ldots, \psi_m$ be λ-defined by G, $H_1, \ldots H_m$, then $\phi(\overrightarrow{n}) = \chi(\psi_1(\overrightarrow{n}), \ldots \psi_m(\overrightarrow{n}))$ is λ-defined by

$$F \equiv \lambda \overrightarrow{x}.G(H_1 \overrightarrow{x}) \cdots (H_m \overrightarrow{x})$$

3. **Closed Under Primitive Recursion** Let a function ϕ be defined by

$$\phi(0, \overrightarrow{n}) = \chi(\overrightarrow{n})$$

$$\phi(k+1, \overrightarrow{n}) = \psi(\phi(k-1, \overrightarrow{n}), k-1, \overrightarrow{n})$$

where χ and ψ are λ-defined by G and H respectively. We will show that we can construct a lambda function that can compute the value for ϕ. We will simply construct a function that each step tests if k=0. If it does, then we compute $\chi(\overrightarrow{n})$, and otherwise we compute the ψ function with all of its inputs.
We first define a function that outputs true if the input is 0 and false if the input is any other number.

$$Zero = \lambda x.xT$$

This does not work the Church numerals, but does work with the numerals defined by

$$\ulcorner 0 \urcorner \equiv I$$

$$\ulcorner n+1 \urcorner \equiv [F, \ulcorner n \urcorner]$$

These are also the numerals that work with the definition of successor supplied above. We then define a term F such that

$$Fx\overrightarrow{y} = \text{if } Zero(x) \text{ then } G\overrightarrow{y}$$

$$else \quad H(F(x^{-}\overrightarrow{y})x^{-}\overrightarrow{y})$$

. Here x^{-} is just another way of saying the predecessor of x, so it would formally be equal to Px We can expressly write this in lambda terms by

$$\theta\lambda zx\overrightarrow{y}.Zero(x)(G(x))H(z(Px)y)Pxy$$

One can also quite easily show that this term is equivalent to the primitive recursion formula for all values of k by induction. Here θ is the previously defined Turing combinator.

4. **Closed Under Minimilization** Let P represent a formula that is either equivalent to T or F. We then define the function that runs until the number is reached in two steps.

$$H_p \equiv \theta(\lambda hz.P(z)zhS^{+}z)$$

$$\mu P \equiv H_p{}^{\ulcorner}0^{\urcorner}$$

In order to get the formula in the exact form of minimilization, we set P to test if the desired formula is 0 and add an extra input for the other inputs of the χ function, which we assume is defined by G. Therefore, we get the following.

$$\lambda\overrightarrow{x}.\theta(\lambda hz\overrightarrow{x}.Zero(G\overrightarrow{x}z))zhS^{+}z\overrightarrow{x}$$

One should now work through these formula and convince themselves that these are equivalent to the original formulas found in the definition.

Since we have defined the primitive functions in terms of lambda formulas and showed it is closed under composition, primitive recursion and minimilization, lambda terms have been shown to be able to encode recursive function. Since the lambda calculus was defined in a recursive way, every lambda term can be made to be equivalent to a recursive function. □

This proof was to show that lambda calculus offers a model of calculation. This is one of the main reasons lambda calculus has been studied.

0.6 Head Normal Forma and Solvability

Definition 16. 1. A closed lambda term, $M \in \Lambda^0$, is **solvable** if there is some sequence of lambda terms that one can append to the original lambda term, which then simplifies done to the identity lambda term, I.

$$\exists n \exists N_1 \dots N_n \in \Lambda \quad MN_1 \dots N_n = I$$

2. An arbitrary term $M \in \Lambda$ is **solvable** if a closure $\lambda\overrightarrow{x}.M$ of M is solvable. This is regardless of the choice of $\overrightarrow{x}$.

3. $M \in \lambda$ is unsolvable if it is not solvable.

Example 5. 1. $K \equiv \lambda xy.x$ is solvable because $KII \to I$

2. $S \equiv \lambda xyz.xz(yz)$ is also solvable by $SIII$

3. $\Omega \equiv ((\lambda x.xx)(\lambda x.xx))$ is not solvable. The parenthesis tell us that we must substitute the second term into the first when reducing it giving us the same term back again. Since this is a closed term, we must first apply reduce them before other terms. They do not allow us to substitute anything into the second term.

The choice for I comes from the fact that we can make I into any other λ-term. $IM \twoheadrightarrow M$. It also makes sense that if we see this term as the identity term. This idea of solvability is the core of what we will use to analyze terms. One of the core ideas of Lambda Calculus is to use topology and continuity to study computation. Though we will not show it here, abstraction and application is are continuous maps with respect to lambda terms. Abstraction is adding a $\lambda x.$ to a term so M becomes $\lambda x.M$. Application is just the concatenation of two lambda terms so the application of M to N is MN.

Proposition 4. Let $M \in \Lambda$

1. M is solvable $\iff$ there exists a closed substitution instance $M*$ and terms $\overrightarrow{N} \in \lambda_0$ such that $M * \overrightarrow{N} = I$.

2. M is solvable $\iff$ $\lambda x.M$ is solvable.

Proof. 1. Let $\lambda x \cdots x_n.M$ be the closure of M and that is solvable to show the forward implication. Then for some terms $N_1, \ldots, N_m$

$$(*) \quad (\lambda x_1 \cdots x_n.M)N_1, \ldots N_m = I$$

Since the left hand side (LHS) reduces down to I, all of the unbound variables must disappear. Therefore, one can rewrite the N_i so they have no free variables or simply replace all free variables with I terms, making all the variables in the N_i terms bound. We can also add I terms to the sequence so that we can say that $m > n$. After substitution each N_i into M we get

$$(**) \quad M[x_1 := N_1] \cdots [x_n := N_n]N_{n+1} \ldots N_m = I$$

We now show the reverse implication. If (**) holds, then for each variable we can add an abstractor for it and replace the N_i with the x_i giving us the form in $(*)$, which is the form that we wanted originally.

2. Let $x_1 \equiv x$.
   M is solvable $\iff \exists \overrightarrow{N}, \overrightarrow{P} \in \Lambda^0 M[x_1 := N_1] \cdots [x_n := N_n]\overrightarrow{P} = I$

   $\iff \exists \overrightarrow{N}, \overrightarrow{P} \in \Lambda^0 (\lambda x.M) N_1 [x_2 := N_2] \cdots [x_n := N_n]\overrightarrow{P} = I$

   $\iff \exists \overrightarrow{N}, \overrightarrow{P} \in \Lambda^0 (\lambda x.M) [x_2 := N_2] \cdots [x_n := N_n] N_1 \overrightarrow{P} = I$

   $\iff \lambda x.M$ is solvable.

□

Corollary 1. If M is unsolvable, then so are MN and $M[x := N]$ and $\lambda x.M$ for all $N \in \Lambda$.

Proof. The first scenario derives directly from the definition of solvable. The second scenario derives from the fact that we have shown that $M[x := N]$ is solvable if and only if $(\lambda x.M)N$ is solvable. Moreover, $(\lambda x.M)N$ being solvable would violate the fact that M is solvable if a closure of M is solvable as $(\lambda x.M)$ is a closure of M. The last situation comes from a direct application of the second part of the last proposition. □

We now define some syntactic categories of terms and prove things about their solvability.

Definition 17. 1. M is an **application term** if M is of the form NL.

2. M is an **abstraction term** if M is of the form $\lambda x.N$.

Lemma 5. 1. Each M is either a variable, an application term or an abstraction term.

2. Each application term M is of the form $M \equiv N_1 N_2, \ldots N_n$ with $n \geq 2$.

3. Each abstraction term M is of the form $M \equiv \lambda x_1 \cdots x_n.N$ with $n \geq 1$ and N not an abstraction term.

This just comes form the definition of lambda terms. One option is a lambda term is just a variable. If it is two variables, then it is officially of the form of an application term because we made no restrictions upon what N and L had to be. If it cannot be divided into two at all, then it must have begin with an abstractor and N cannot be an abstraction term because we would fold the variable that N is an abstractor for into the first abstractor.

Proposition 5. Each M is of one of the following two forms.

1. $\lambda x_1 \cdots x_n.xM_1 \cdots M_m, \quad n, m \geq 0$

2. $\lambda x_1 \cdots x_n.(\lambda x.M_0)M_1 \cdots M_m \quad n \geq 0, m \geq 1$

Proof. If M is a variable then it is of form 1 with n and m being 0. If M is an applicator term, it could be either of the terms with n being 0. If M is an abstactor, then after the abstractor must be another term that begins with an abstractor or a variable as that is all that the lambda calculus allows for terms to be made of. □

Definition 18. 1. M is a **head normal form** if $M \equiv \lambda x_1 \cdots x_n.xM_1 \cdots M_m \quad n, m \geq 0$.

2. M has a head normal form (abberviated hnf) if $\exists N$ such that $M = N$ and N is a head normal form.

3. If M is of the form

$$M \equiv \lambda x_1 \cdots x_n.(\lambda x.M_0)M_1 \cdots M_m \quad {\scriptstyle n \geq 0 m \geq 1}$$

then $(\lambda x.M_0)M_1$ is the **head redex** of M.

The key example of a term without a head normal form is $\Omega \equiv ((\lambda x.xx)(\lambda x.xx))$. The only term that Ω can reduce to is itself. If we append any other term to it, those terms are ignored because of the parenthesis. In the official parlance, we would say that any term that we add on would not be in the scope of either abstractor. As one can see, being an hnf can be verified by just looking at the symbols that the term consists of, whereas for solvability, one has to consider the application of terms to get the new term to reduce down to I. We will see that one can identify all terms without head normal forms as having the same meaning in a consistent way, really highlighting the special nature of head normal forms.

We use this definition of head redex to remove the choice of possible reduction/substitution moves to make at any given step, allowing us to prove things more easily.

Definition 19. 1. Suppose Δ is the head redex of M. We write $M \to_h N$ if N results from M by contracting Δ. The reduction $\to_h$ is called a **one step reduction**. We will denote the contraction of any redex as $\to_\Delta$, and we will denote the contraction of a head redex as $\to_h$.

2. $\twoheadrightarrow_h$ is the transitive reflexive closure of $\to_h$.

3. The **head reduction** of M is the uniquely determined sequence $M_0, M_1, \ldots$ such that $M \equiv M_0 \to_h M_1 \to_h \cdots$. If M_n is a hnf then the head reduction of M **terminates** at M_n. If there is no such M_n, then M has an **infinite head reduction**.

0.7 β Reduction

In this section, we will show that M has a normal form if and only if the head reduction path of M terminates.

The first major idea here is the labeling of elements of lambda terms. This allows us to restrict the number of reductions that we can do and to ensure the reductions are of the right form. Moreover, there is a mapping from numbered terms to unnumbered terms by just reducing the numbers.

Definition 20. 1. Λ' is the set of words over the following alphabet

(a) $v_0, v_1, \ldots$ variables

(b) $\lambda, \lambda_1, \lambda_2 \ldots$ lambdas

(c) (,) paranthesis

2. We define terms on Λ' inductively according to these rules

(a) $x \in \Lambda'$

(b) $M \in \Lambda' \implies (\lambda x.M) \in \Lambda'$

(c) $M, N \in \Lambda' \implies (MN) \in \Lambda'$

(d) $M, N \in \Lambda' \implies ((\lambda_i x.M)N) \in \Lambda' \quad \forall i \in \mathbb{N}$

3. We also define the operation $|*|$ for $M' \in \Lambda'$ as $|M'| \in \Lambda$ and $|M'|$ is the Λ term with out any of the indices of M'. So for example $|(\lambda_1 x.xx)((\lambda_2 x.x)(\lambda x.x))| \equiv (\lambda x.x)((\lambda x.x)(\lambda x.x))$ or $I(II)$

We will now define $\beta-$reduction on this new set of terms. This change to Λ' adds some new notation. We will denote reduction paths as $\sigma : M \equiv M'_0 \to^{\Delta'_0} M'_1$ where each Δ'_i denotes the specific term being contracted in the reduction. The $'$ will denote that we are doing this based off of the rules in Λ' and not just regular Λ. This also means that we can meaningfully say $|\Delta'_0|$ as it just means the same reduction but without any regard to the indexing at all. Therefore, if we have $M \equiv (\lambda_1 x.xy)(\lambda x_2.xx)(\lambda_3 x.xy)$, we can have $\Delta'_0 \equiv (\lambda x_1.xy)(\lambda x_2.xx)$ which when contracted would give us $M_1 \equiv (\lambda_2 x.xx)y)(\lambda_3 x.xy)$. If we were instead to decide to apply the $|*|$ to Δ'_0 we would instead be reducing $(\lambda x.xy)(\lambda x.xx)$ which would in turn give us $M_1 \equiv ((\lambda x.xx)y)(\lambda_3 x.xy)$

Definition 21. 1. **Substitution** can be defined in essentially the same way on Λ' because we are only labeling the abstractors and not actual variables. Therefore we get

$$((\lambda_i x.M)N)[z := L] \equiv (\lambda_i.M[z := L])(N[z := L])$$

2. $\beta'-$**reduction** here the $'$ denotes that we are operating on Λ' and not regular Λ terms. We split reduction into two cases when we apply β' reduction on an indexed term and when we apply it on a non-indexed term.
The rule for index terms is called β_0 and is defined by

$$(\lambda_i x.M)N \to M[x := N]$$

The rule for terms without an index is called β_1

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

3. We then define the binary relations $\to_{\beta'}$ and $\twoheadrightarrow_{\beta'}$ as we did for $\to_\beta$ and $\twoheadrightarrow_\beta$ previously.

Lemma 6. (Projecting) Let σ' be a β' reduction starting with $M' \in \Lambda'$ of the form $\sigma' : M' \equiv M'_0 \to^{\Delta'_0}_{\beta'} M'_1 \to^{\Delta'_1}_{\beta'} \ldots$. Then $|\sigma'| : |M'| \equiv |M'_0| \to^{|\Delta'_0} |M_1| \to^{\Delta'_1}$ is a β-reduction starting with $|M'|$.

Proof. This comes from the fact that β'-reduction acts in the same way as the β-reduction. □

Lemma 7. (Lifting)

Let σ be a β-reduction starting with $M \in \Lambda$. Then for each $M' \in \Lambda'$ with $|M'| \equiv M$ there is a β'-reduction σ' starting with M' such that $|\sigma'| = \sigma$

Proof. Here we can just add indices to certain abstractors/terms to produce a reduction path with the necessary qualities. □

Definition 22. 1. Let $M \in \Lambda$. $\Delta \in M$ denotes that Δ is a redex occurrence in M.

2. Let $\mathscr{F}$ be a set of redex occurrences. $\mathscr{F} \subseteq M$ denotes that $\forall \Delta \in \mathscr{F} \quad \Delta \in M$

3. Let $\mathscr{F} \subseteq M \in \Lambda$. Then $(M, \mathscr{F}) \in \Lambda'$ is the indexed term obtained from M by indexing the redex occurrences of M that are in $\mathscr{F}$ by 0 .

Now we can define the idea of a residual which is at the core of the equivalence between these types of terms.

Definition 23. Let $M, N \in \Lambda$ and $\sigma : M \twoheadrightarrow N$

1. Let $\mathscr{F} \subseteq M$. The set of **residuals** of $\mathscr{F}$ in N relative to σ is defined as follows. Let $M \equiv (M, \mathscr{F})$ and lift σ to $\sigma' : M' \twoheadrightarrow N'$. Since the β'-reductions do not create new indices N' only has 0 indices and so $N' \equiv (N, \mathscr{F}')$ for some $\mathscr{F}'$.

2. If $\Delta \in M$, then the residual of Δ in N relative to σ denoted by $\Delta//\sigma$ is the same. The only difference in this situation is that there is only one term in the $\mathscr{F}$

This allows us to mark terms and define them as unique objects throughout the process. Previously, when looking at the initial λ term and the term in its normal form we could not say that certain subterms were unaffected. The same term could have been removed and then reconstructed in other ways. However, now we can say that this term with a given index is the same term term as some other term.

We now officially state some minor properties of residuals.

Lemma 8. Let $M, N \in \Lambda$, $\sigma : M \twoheadrightarrow N$ and $\mathscr{F} = \{\Delta_1, \ldots, \Delta_n\}$. Then $\mathscr{F}/\sigma = \Delta_1/\sigma \cup \cdots \cup \Delta_n/\sigma$

This means that the set residuals of $\mathscr{F}$ is defined by whether each individual residual is unaffected by β-reduction through the path.

Lemma 9. Let $\sigma : M \twoheadrightarrow N, \tau M \twoheadrightarrow L$ and $\mathscr{F} \subseteq M$. Then $\mathscr{F}/(\sigma + \tau) = (\mathscr{F}/\sigma)/\tau$

This in turn means that it does not matter how we splice up the reduction paths. As long as the sets of reductions are the same, we have the same properties. We want residuals to in some way represent individual terms throughout the process of reduction, so this is an important property. The same reductions will affect the residuals in the same way because the same reductions will affect the same terms in the same way.

Example 6. Let $\Delta \equiv (\lambda a.a(Ix))(xb)$

1. $(\lambda x.xx)a\Delta \to aa\Delta$. The residual of Δ relative to the reduction is unchanged because the reduction does not affect the Δ term in any way.

2. $(\lambda x.xx)\Delta \to \Delta\Delta$. Here, Δ has two residuals the two instances of it in the reduced term.

3. $(\lambda x.x)\Delta \to (\lambda x.x)(xb(Ix)$. Here, Δ has no residuals because there indices of Δ are completely gone. The indices are only stoed in the λx parts of the terms. However, this reduction of Δ removes that abstractor, so the residual no longer exists.

4. $\Delta \to (\lambda x.ax)(xb)$. Here, the residual of Δ is $(\lambda a.ax)(xb)$ because the abstractor that identified it as a Δ residual remains in the term.

To give some more context, the residual is no longer there because the thing we used to mark the term is no longer there. The only thing that we have to mark term is indices on abstractors in those terms. Howver, once those disapear there is no longer a syntactic part of the term that marks it as part of the original term. We rely upon syntactic terms because we need to automate each step to be able to generalize about it. If there was any context one needed to define a residual, then one could not abstract beyond it.

If the reduction path of M is implied, one can just say the residuals of $\mathcal{F}$.

Definition 24. Let $M \in \Lambda$ and $\mathcal{F} \in M$.

1. A **development** of $(M, \mathcal{F})$ is a reduction path $\sigma : M \equiv M_0 \to^{\Delta_0} M_1^{\Delta_1} \cdots$ such that each redex $\Delta_i \in M_i$ is a residual of a redex in $\mathcal{F}$ relative to the reduction path form by the Δ_i.

2. $\sigma : M \twoheadrightarrow N$ is a **complete development** of $(M, \mathcal{F})$ if σ is a development of $(M, \mathcal{F})$ and $\mathcal{F}/\sigma = \emptyset$. Therefore, the path is a complete development if it ends without any o fthe original residuals.

3. A **development of M** is a development of $(M, \mathcal{F})$ where M is the set of all redex occurrences in M.

4. $M \twoheadrightarrow_{dev} N$ if and only if N occurs in some development of M. Therefore, N is the result of some series of reductions that form a reduction.

The key ideas

Lemma 10. σ is a development of $(M, \mathcal{F})$ if and only if σ is lifted to σ' starting with $(M, \mathcal{F})$ is a β_0-reduction

0.7.1 Finiteness of β-reduction

The core idea of this proof is to keep track in some way of the number of variables in the proof and show that they are always being reduced at each level. This ensures that the reduction is itself finite.

Definition 25. $\Lambda' *$ is the set of weighted Λ'-terms defined indcutvely as follows

1. $x^n \in \Lambda' *$ for every variable and every $n \in \mathbb{N}, n > 0$

2. $M \in \Lambda' * \implies (\lambda x.M) \in \Lambda' *$

3. $M, N \in \Lambda' * \implies (MN) \in \Lambda' *$

4. $M, N \in \Lambda'* \implies ((\lambda_i x.M)N) \in \Lambda'*$ where $i \in \mathbb{N}$

We can also define such terms as a Λ' term paired with another term that is a weighting of each of the variables, I. We will denote such a pari by (M_0, I) where M_0 is the Λ' term and I is the weighting of the variables within M_0.

The integers that we attach to the variables will be called weights. In this new alphabet, each variable not directly attached to a abstractor will have a variable. Therefore, $\lambda x.xx$ is a regular lambda term but it is not a $\Lambda*'$ term. However, if we rewrite the term as $\lambda x.x^1x^1$ or $\lambda x.x^1x^2$ it would become a valid $\Lambda*'$ term.

Definition 26. The notion of reduction β_0 is extended to $\Lambda'*$ by defining substiution in the same way so

$$x^n[x := N] \equiv N$$

. So we are essentially ignoring the weights when actually doing the substiution. We will also define a version of β_0 for this new language as β_0' and define it as

$$\beta_0^* : (\lambda_i x.M)N \to M[x := N] \quad \forall M, N \in \Lambda'*$$

We will also extend the definitions of $\to_{\beta_0^*}$ and $\twoheadrightarrow_{\beta_0^*}$ in the natural way.

The most important change here is that we are only dealing with indexed abstractors when doing reductions, so if a term appears as as $\lambda x.x^1x^1$ in the language, we will never apply the β_0^* rule to it and substitute x for any term in the term.

Definition 27. Let $M \in \Lambda'*$, for $N \subset M$ define

$$\|N\|'$$

=sum of the weights occurring n N

This number will always be above 0 because any subterm needs to contain at least 1 variable and each variable is bounded below by 0.

Definition 28. Let $M \equiv (M_0, I)$.
The weighting I is **decreasing** if for every β_0^*-redex $(\lambda_i x.P)Q$ in M one has

$$\|x\|' > \|Q\|' \quad \text{for all occurrences of x in P.}$$

If a term has a decreasing weighting then the β_0^* reduction will decrease the sum of weighted variables so $\|P[x := Q]\|' < \|(\lambda_i x.P)Q\|$

Example 7. 1. $(\lambda_2 x.x^6x^7)(\lambda x.x^2x^3)$ has a decreasing weighting

2. $(\lambda_1 x.x^4x^7)(\lambda x.x^2x^3)$ does not have a decreasing weighting

Lemma 11. Let $M \in \Lambda'$. There is a waiting I for M that is a decreasing weighting

Proof. There are many ways to do this but one of the simplest is to use powers of 2. One can count the variables from right to left and then give them the index of 2 to the power of the number on the index, starting with 0. For $xxyyzzaa$, one would have $x^{32}x^{16}z^8z^4a^2a^1$. Since $2^n = 1 + \sum_{i=0}^{n-1} 2^i$ the decreasing property holds. □

We will now prove the core of our argument regarding finite development, namely that β_0 reduction allows for a way that continually reduces the sum of weights of a term. This will be a very technical proof, but most lambda calculus proofs are very technical.

Lemma 12. Let $M^* \equiv (M, I) \in \Lambda'^*$ so M* is equivalent to M with I as a decreasing weighting and let $M^* \to_{\beta_0^*} N^* \equiv (N, I')$.

1. $\|M^*\|' > \|N^*\|'$.
2. I' is a decreasing weighting.

Proof. Since we are only dealing with the β_0^* reduction, we can assume that the changed part of the term is of the form $\Delta_1 \equiv (\lambda_i x_1.P_1)Q_1$. So, $\Delta_1 \subseteq M$. Hence, Δ_1 is the redex contracted in $M^* \to_{\beta_0^*} N^*$. We will now show that the new weighting has a reduced weighting sum and the new weighting is still decreasing.

1. Since each x_1 in P_1 is replaced by Q_1 in the reduction and since $\|x_1\|' > \|Q_1\|'$ by the fact that the weighting of M is a decreasing weighting, the sum of the weights in the contractum is decreased. Moreover, if $x_1 \notin P_1$ the sum also decreases. This comes from the fact that $\|Q_1\|' > 0$ and we are essentially removing Q_1 from M entirely. Therefore, the sum of weights has decreased in M.

2. We will now look at a β_0^* redex of N $\Delta_0 \equiv (\lambda_j x_0.P_0)Q_0$. Since Δ_0 is indexed, it is the residual of a redex in M^*. This is because the reduction cannot create a new abstractor and residuals are defined by abstractors. Let us define the redex in $M*$ as $\Delta_2 \equiv (\lambda_j x_2.P_2)Q_2$.

   Of those cases in the table there are only two scenarios in which it is not trivial to show that I' is still decreasing.

   (a) The first of these cases is when $\Delta_1 \subset Q_2$.

$$M^* \equiv \cdots (\lambda_j x_2.P_2)\boxed{\cdots((\lambda_i x_1.P_1)Q_1)\cdots}_2 \cdots$$

$$\beta_{0^*} \downarrow$$

$$N^* \equiv \cdots (\lambda_j x_2.P_2)\boxed{\cdots P_1[x := Q_1]\cdots}_0 \cdots$$

   where $\boxed{\quad}_2 \equiv Q_2$ and $(\lambda_j x_2.P_2)\boxed{\quad}_0 \equiv (\lambda_j x_0.P_0)Q_0$. Since M^* has a decreasing weighting, one has

$$\|(\lambda_i x_1.P_1)Q_1\|' > \|P_1[x_1 := Q]\|'. \tag{1}$$

   and for all $x_2 \equiv x_0$ in $P_2 \equiv P_0$

$$\|x_2\|' > \|Q_2\|' \tag{2}$$

   From (1) it follows that $\|Q_2\|' > \|Q_0\|'$, and therefore by (2) we have $\|x_0\| > \|Q_0\|'$, which is the definition of a decreasing weighting, as all other weightings remain unchanged and therefore decreasing.

(b) The second of these cases is when $\Delta_2 \subset P_1$. This means that

$$M^* \equiv \cdots (\lambda_i x_1.\boxed{\cdots x_1 \cdots ((\lambda_j x_2.P_2)Q_2 \cdots}_1)Q_1 \cdots$$

$$\beta_{0^*} \downarrow$$

$$N^* \equiv \cdots \boxed{\cdots Q_1 \cdots ((\lambda_j x_2.P_2[x_1 := Q])(Q_1[x_1 := Q_1]))}$$

where $\boxed{\quad}_1 \equiv P_1$ and $(\lambda_i x_0.P_0)Q_0 \equiv (\lambda_j x_2.P_2[x_1 := Q_1])(Q_2[x_1 := Q])$. Since M^* has decreasing weights,

$$\|x_1\|' > \|Q_1\|' \quad \forall x_1 \in P_1 \tag{3}$$

and

$$\|x_2\|' > \|Q_2\|' \qquad \forall x_2 \in P_2 \tag{4}$$

By (3), we have $\|Q_2\|' \geq \|Q_2[x_1 := Q_1]\|'$. The two are equal when $x_1 \notin FV(Q_2)$. Then we can apply (4) to get $\|x_2\|' > \|Q_2[x_1 := Q_1]\|'$ for all x_2 in $P_2[x_1 := Q_1]$.

□

Theorem 4 (FD:Finitude of Developments)**.** Let $M \in \Lambda$. Then all developments of M are finite.

Proof. We have already shown that β_0 reduction paths are developments. Since weighted β_0 reduction paths form a strictly decreasing chain, they must reach a minimum value at some point. Therefore, we can map the development into a β_0 reduction path, making it complete in finite time.

□

This means that the developments give us a computable path to a normal form of lambda terms. From this we shall show that having a hnf is equivalent to the head reduction path of M terminating. We will now prove a slight extension of this concept.

Corollary 2. Let $M \in \Lambda$.

1. For $\mathscr{F} \subseteq M$, each development of $(M, \mathscr{F})$ can be extended to a complete one.
2. The set $\{M | M \twoheadrightarrow_{dev} N\}$ is finite.

Proof. 1. By the FD, there is a development of $(M, \mathscr{F})$ of maximal length, so there must be a complete one.

2. Each term only has finitely many one step reducts. Since at each stage there are only finitely many moves to make and each development is finite, the whole graph of the stages must be finite by Koenigs lemma.

□

Lemma 13. 1. β_0 is weakly Church Rosser.

2. If $M, M_1, M_2 \in \Lambda'$ and $\sigma : M \to^{\Delta_1}_{\beta_0} M_1$, $\tau : M \to^{\Delta_2}_{\beta_0} M_2$, then there are reductions $\sigma' : M_2 \twoheadrightarrow_{\beta_0} M_3$ and $\tau' : M_1 \twoheadrightarrow_{\beta_0} M_3$ where σ' and τ' are β_0-reductions formed by contracting the residuals of Δ_1/τ and $\Delta_{2/\sigma}$ one after the other from left to right.

3. The same holds for β'

Proof. All of this follows from our proof that $\to$ is weakly Church Rosser. □

Corollary 3. 1. The notion of β_0 reduction on Λ' is Church Rosser.

2. Each $M' \in \Lambda'$ has a unique β_0-nf.

Proof. 1. This is because the chain of reductions must be finite and it is weakly Church Rosser. The local property extends to the entirety of all reduction paths.

2. Since it is Church Rosser, it must have unique normal forms.

□

Now, we will actually show that there is a natural extension to make on FD, which we will call FD!

Theorem 5 (FD!)**.** Let $M \in \Lambda$ and $\mathscr{F} \subseteq M$.

1. All developments of M are finite.

2. All developments of $(M, \mathscr{F})$ can be extended to a complete development of $(M, \mathscr{F})$.

3. All complete developments of $(M, \mathscr{F})$ end with the same term.

Proof. 1. By FD and part 2 of Corollary 2.

2. By FD and part 1 of Corollary 2 .

3. By Lemma 10 in Section 6 and the previous corollary.

□

Definition 29. 1. Let $\sigma : M_0 \to^{\Delta_0} M_1 \to^{\Delta_1} M_2 \to^{\Delta_2} \cdots$ be a reduction. σ is a **standard reduction** if $\forall i \forall j < i[\Delta_i$ is not a residual of a redex to the left of $\Delta_j]$. Here P being to the left of Q in some term M means that every part of P is written to the left of Q, so leftness corresponds to the position of the terms relative to each other syntactically. This is relative to the reduction from M_j to M_i.

2. We write $M \to_s N$ if there is a standard reduction $\sigma : M \to N$.

Let us say we were to reduce a subterm Δ_0 in M. If we want the reduction path to be a standard reduction path, we cannot apply the reduction rule to any subterm to the left of Δ_0. Since head reduction always choose the leftmost substitution possible, it is standard.

Example 8. 1. $\lambda a.(\lambda b.\underline{(\lambda c.c)}bb)d \to \lambda a.\underline{(\lambda b.bb)}d \to \lambda a.dd$. This reduction path is not a standard reduction because after we reduce the λc subterm we reduce the λb subterm and the latter was to the left of the former term.

2. $\lambda z.\underline{\lambda b.(\lambda c.c)bb)d} \to \lambda a.\underline{(\lambda c.c)dd} \to \lambda a.dd$. This reduction path is a standard reduction because we never reduced any lambda term that was to the left of a reduce lambda term.

To ensure that the contraction being done is a standard reduction, one could label each of the abstractors to the left of the abstractor being reduced. After that point, it would be forbidden to reduce any of the lambda terms with a label. This is exactly what is meant by the use of residuals in the definition of the term. It also becomes necessary because it would otherwise be hard to identify which abstractors were to the left of the reduced term. As the term was reduced, there would be no abstractor to define it as being to the left of anymore. I would like to recall here that if a head redex exists it is simply the leftmost redex of the form $(\lambda x.M)N$.

Definition 30. 1. Let $M \in \Delta$ and $\Delta \in M$. Δ is **internal** in M if Δ is not the head redex of M.

2. We say that $M \twoheadrightarrow_i N$ if there is a reduction $\sigma : M \equiv M_0 \to^{\Delta_0} M_1 \to^{\Delta^1} \cdots \to M_n \equiv N$ such that each Δ_i is internal in M_i for $0 \leq i < n$.

3. We say that $M \twoheadrightarrow_{1,i} N$ if there is a reduction $\sigma : M \twoheadrightarrow_i$ which is at the same time a complete reduction of some $(M, \mathscr{F})$.

Example 9. If we let $M \equiv \lambda x.(\lambda z.zz)(I(Ix))$, then Ix and $I(Ix)$ are the internal redexes and $(\lambda z.zz)(I(Ix))$ is the head redex.

1. $M \to_i \lambda x.(\lambda z.zz)(Ix)$ is an internal reduction.

2. $M \twoheadrightarrow_{1,i} \lambda x.(\lambda z.zz)x$. The $\mathscr{F} \equiv \{Ix, I(Ix)\}$.

3. $M \to_h \lambda x.(I(Ix))(I(Ix))$ is a head reduction.

4. $M \twoheadrightarrow_h \lambda x.x(I(Ix))$. Here, the only reduction we would do would be internal reductions, so it is in its head normal form.

5. $M \twoheadrightarrow_1 \lambda x.xx$ if $\mathscr{F} = \{\lambda x.(\lambda z.zz)(I(Ix)), Ix, I(Ix)\}$.

Here are some basic results about these redexes.

Lemma 14. Let $\sigma : M \to^{\Delta} N$ where Δ is an internal redex of M.

1. If N has a head redex, then so does M.

2. If Δ_h is the head redex of M, then Δ_h/σ consists of exactly one element which is the head redex of N.

3. If Δ_i is an internal redex of M, then all elements of Δ_i/σ are internal redexes of N.

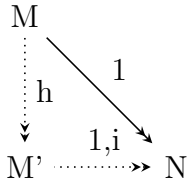
Proof. 1. If M has no head redex, then it must be in head normal form, but then N is also in head normal form. However, since N has a head redex it cannot be in head normal form.

2. Let us give the head redex Δ_h of M an index, 0. If we then contract internal redexes, this redex will not be canceled or duplicated. Therefore, the 0-redex remains the head redex the entire time.

3. Throughout the entire process the head redex is not affected, so none of the internal redexes can become head redexes. Therefore, Δ_i/σ consists only of internal redexes.

□

Lemma 15. For $M, M', N \in \Lambda$,



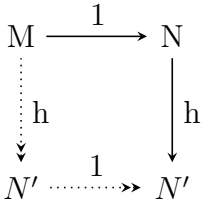
Proof. Let N be the complete reduct of $(M, \mathscr{F})$. We can then develop $(M, \mathscr{F})$ in the following steps.

1. Contract the consecutively indexed redexes that are head redexes. By FD, this process stops at some point. We shall call it M'.

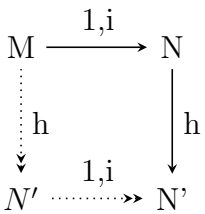
2. Complete the development by contracting the remaining indexed redexes. By FD!, this leads again to N.

By definition $M \twoheadrightarrow_h M'$ and $M' \twoheadrightarrow_1 N$. By 3 of the the previous lemma, $M' \twoheadrightarrow_{1,i} M$. □

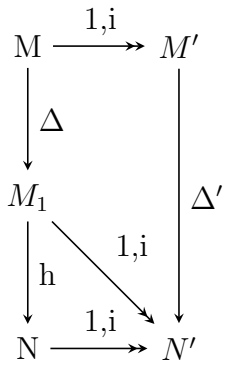
Lemma 16. For $M, N, N' \in \Lambda$



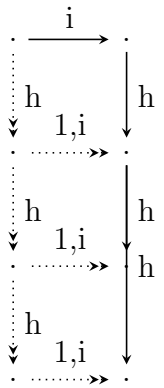
Proof. We will first show that the following diagram(which we will call 1) holds.



Let M' be the complete reduct of $(M, \mathscr{F})$ for some $\mathscr{F}$ such that all elements of it are internal redexes. By 1 in Lemma 11, M has a head redex. Let us call that head redex Δ. By the second part of that same lemma, Δ has exactly one residual Δ' in M' and $M' \to_h^{\Delta'} N'$. Therefore, $M \twoheadrightarrow_{1,i} M' \to_h N'$ is a complete development of $(M, \mathscr{F}) \cup \{\Delta\}$. Let M_1 be obtained form M by contracting Δ. By FD!, $M_1 \twoheadrightarrow_1 N'$ by a complete development of all residuals of $\mathscr{F}$ in M_1. Hence by Lemma 11,; there exists N such that $M_1 \twoheadrightarrow_h N \twoheadrightarrow_{1,i} N'$. This shows that 1 is true. The diagram of this reality is as follows.

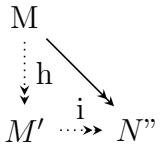


Since $M \to_i M' \twoheadrightarrow_{1,i} M'$, the statement of the lemma follows by diagram 1 and a diagram chase suggested in the following figure.



□

Lemma 17. For $M, N, M' \in \Lambda$,



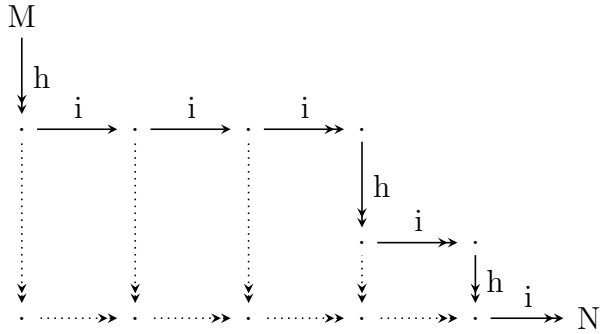
This means that for all $M, N \in \Lambda$ and reductions $M \twoheadrightarrow N$, there is $M' \in \Lambda$ and reductions $M \twoheadrightarrow_h M'$ and $M' \twoheadrightarrow_i N$ that make the above diagram commute.

Proof. Since any reductions $M \twoheadrightarrow N$ are of the form

$$M \twoheadrightarrow_h M_1 \twoheadrightarrow_i M_2 \twoheadrightarrow_h M_3 \cdots \twoheadrightarrow_i N,$$

we can divide this into both head and internal reductions because all terms are either a head normal redex or they are not. Since an internal redex is any redex other than the head redex, we can divide any sequence of reductions into two sets: those subsets that are just a series of internal reductions and those subsets that are a series of head reductions.

We can do a chase as mentioned at the end of the last theorem to get



□

Theorem 6. Standardization Theorem If $M \twoheadrightarrow N$, then $M \twoheadrightarrow_s$ N

Proof. We will prove this by induction on the lenght of N. By previous lemma we have $\exists M \twoheadrightarrow_h Z \twoheadrightarrow_i N$.

1. If $N \equiv x$, then we are done. This is because the last step must be a head reduction as there would only be one redex left in the entire term. Since each head reduction is also a standard reduction we are done.

2. If $N \equiv \lambda x_1 \cdots x_n . N_0 N_1 \cdots N_m$with $n + m > 0$, then Z must be of the form

   $$Z \equiv \lambda x_1 \cdots x_n . Z_0 Z_1 \cdots Z_m \text{ with } Z_i \twoheadrightarrow N_i \quad 0 \leq i \leq m$$

   By the induction hypothesis

   $$\exists \sigma_i \quad \sigma_i : Z_i \twoheadrightarrow_s N_i \quad 0 \leq i \leq m$$

   . If we let $\sigma : M \twoheadrightarrow_h Z$, then $\sigma + \sigma_0 + \cdots + \sigma_m : M \twoheadrightarrow_s N$.

□

Corollary 4. M has a hnf if and only if the head reduction path of M terminates.

Proof. $\Longrightarrow$ Let $M = \lambda \overrightarrow{x} . y \overrightarrow{M}$, so it is not that M is syntactically equivalent to $\lambda \overrightarrow{x} . y \overrightarrow{M}$, but that that is its head normal form. By Church Rosser though, if$\lambda \overrightarrow{x} . y \overrightarrow{M}$ reduces down to Z, then it must also have a reduction path to Z. Since Z is the result of a reduction path

form a head normal form, it must be of the form $Z \equiv \lambda \overrightarrow{x}.y\overrightarrow{N}$ with $M_i \twoheadrightarrow N_i$. Therefore, by the standardization theorem,

$$M \twoheadrightarrow_s \lambda \overrightarrow{x}.y\overrightarrow{N}. \tag{5}$$

Let this reduction be $M \equiv M_0 \to^{\Delta_0} M_1 \to^{\Delta_1} \cdots \to \lambda \overrightarrow{x}.y\overrightarrow{N}$. If all the Δ_i are head redexes, then (5) is a terminating head reduction. Otherwise, let Δ_i be the first internal redex. Then M_i must be in hnf because otherwise its head redex would remain. Therefore, $M \twoheadrightarrow M_i$ is a terminating head reduction.
$\Longleftarrow$ If the head reduction path terminates, then the form equation must be in head normal form or else one could continue the reduction sequence. □

Now we have a connection between the existence of the head normal form and the head reduction, so we can say a lot more about head reduction. Without this proof, the path could be some sequence of clever reductions that are of all different forms, which would require a much more complicated proof to say anything about it. There would be so many more cases. Moreover, these intricate modifications and constructions around λ-calculus is at the core of how one says things about it. This whole sequence of thought is a good example of how technical this subject can be.

0.8 HNF Facts

Here we will develop some facts about head reduction that will be used later.

Lemma 18. If $M \to_h M'$, then $M[z := N] \to_h M'[z := N]$.

Proof. By assumption, we have $M \equiv \lambda \overrightarrow{x}.(\lambda y.N_0)N_1N_2 \cdots N_m$ and $M' \equiv \lambda \overrightarrow{x}.N_0[y := N_1]N_2 \cdots N_m$. Therefore, $M[z := N] \equiv \lambda \overrightarrow{x}.(\lambda y.N_0[z := N])N_1[z := N]N_2^* \cdots N_m^*$ where $N_i^* = N_i[z := N]$ and $M'[z := N] \equiv \lambda \overrightarrow{x}.N_0[y := N_1][z := N]N_2^* \cdots N_m^*$. By the substitution lemma and our variable conventions, $M[z := N] \to_h M'[z := N]$.

□

Proposition 6. 1. $\lambda x.M$ has a hnf if and only if M has a hnf.

2. If $M[z := N]$ has a hnf, then M has a hnf.

3. If MN has a hnf, then M has a hnf.

Proof. 1. If $\lambda x.M \twoheadrightarrow N$ then $N \equiv \lambda x.N'$ and $M \twoheadrightarrow N'$, as there is no way to reduce away the outer variables without any terms to contract them with. The reverse argument applies in the reverse implication.

2. Suppose that M has no hnf. Then by Corollary 2, the head reduction is infinite. By the previous lemma, the reduction of $M[z := N]$ would also be infinite. Using N in the place of z could only add possible reductions and not remove them. Therefore, by Corollary 2 again, $M[z := N]$ has no hnf.

3. Let
$$M \equiv M_0 \to_h M_1 \twoheadrightarrow_h \cdots \tag{6}$$
be the finite head reduction of M, which exists by Corollary 2.

   (a) If M is not an abstraction term for any k, then it must be an application term. $MN \equiv M_0N \to_h M_1N \to_h$ is the head reduction of MN. Since this reduction must be finite, (6) must also be a finite reduction. This in turn gives us an abstraction term.

   (b) If for some M_k is an abstraction term for some k, then choose k minimial such that $M_k \equiv \lambda x.M'$. The head reduction of MN begins by
$$MN \equiv M_0N \to_h \cdots \to_h M_kN \equiv (\lambda x.M')N \to_h M'[x := N] \to_h \cdots .$$
   Since MN has a hnf, $M'[x := N]$ has a hnf by Church Rosser property. By the second part of this proposition, we know that M' must have a hnf as $M[x := N]$ does. By the first part of the proposition, this then means that $\lambda x.M'$ must have a hnf. This then gives us a head normal form for M, namely the head normal form of $\lambda x.M'$ as M has a reduction path to $\lambda x.M'$.

□

Now we have arrived at the correlation between solvability and head normal forms.

Theorem 7. M is solvable if and only if M has a hnf.

Proof. By part 2 of Proposition 4 in 0.5 and part 1 of Propsoition 6, we may assume that M is closed.

$\Longrightarrow$ If $M\overrightarrow{N} = I$ then $M\overrightarrow{N}$ has a hnf. Hence by part 2 of proposition 6 M has a hnf.

$\Longleftarrow$ If $M = \lambda x_1 \cdots x_n.x_iM_1 \cdots M_m$, then $M(K^mI)^{\sim n} = K^mIM_1^* \cdots M_m^* = I$. This is because each of the K terms essentially serves to remove one subterm from the overall term and the $\sim n$ allows us to use up all of the variables as we do the head normal reduction. When we do the substitution, we get
$$M(K^mI)^{\sim n} \twoheadrightarrow K^mIM_1^* \cdots M_m^* \to K^{m-1}IM_2^* \cdots M_n^*$$
where M_i^* represents the M_i with the different variables replaced by K_mI. □

We will now prove things about certain special cases.

Corollary 5. M is unsolvable if and only if for all substiution instances M^* and all sequences $\overrightarrow{N}$, $M^*\overrightarrow{N}$ has no nf.

Proof. $\Longrightarrow$ If $M^*\overrightarrow{N}$ had a nf, then by the previous thoerem and the first part of Proposition 4 in Section 0.5, for some $\overrightarrow{P}$ we have $(M^*\overrightarrow{N})^*\overrightarrow{P} = I$ where $(M^*\overrightarrow{N})^*$ is a closed instances of $M^*\overrightarrow{N}$. Therefore, $M^{**}\overrightarrow{N}^*\overrightarrow{P} = I$ and M is solvable by proposition 4 part 1 in 0.5.

$\Longleftarrow$ If M is solvable, then by Propsoition 4 in Section 0.5, $M^*\overrightarrow{N} = I$ for some $\overrightarrow{N}$ □

Lemma 19. Let $M \equiv \lambda x_1 \cdots x_n.xM_1 \cdots M_m$. If $M \twoheadrightarrow N$, then $N \equiv \lambda x_1 \cdots x_n.xN_1 \cdots N_m$ for some $\overrightarrow{N}$ with $M_1 \twoheadrightarrow N_1$.

Proof. The only possible redexes in M are the $M_1 \ldots M_m$. Therefore, if $M \twoheadrightarrow N$, then $N \equiv \lambda x_1 \cdots x_n.xN_1 \cdots N_m$ with $M_i \twoheadrightarrow N_i$. □

Corollary 6. 1. Let $M \in HNF$ where HNF is the set of terms in head normal form, and $M \twoheadrightarrow N$. Then $N \in HNF$.

2. If M has the hnfs N and N' such that

$$N \equiv \lambda x_1 \cdots x_n.xN_1 \cdots N_m$$

$$N' \equiv \lambda x_1 \cdots x_{n'}.x'N'_1 \cdots N'_m$$

then $n = n'$, $x \equiv x$, $m = m'$ and $N_i = N'_i$ for $1 \leq i \leq m$.

Proof. 1. Follows directly from the previous lemma.

2. By assumption, $N = M = M'$. Therefore, by the Church Rosser property, there exists Z such that $N \twoheadrightarrow Z$ and $N' \twoheadrightarrow Z$. By the previous lemma, Z is of the form $\lambda x_1 \cdots x_n.xZ_1 \cdots Z_m$ where $n = n'$, $x \equiv x'$, $m = m'$ and $N_i \twoheadrightarrow Z_i$ and $N'_i \twoheadrightarrow Z_i$ for $1 \leq i \leq m$.

□

Lemma 20. The set of set of normal forms NF can be defined in either of two ways.

1. (a) $x \in NF$.
   (b) $M_1, \ldots, M_m \in NF \implies \lambda x_1 \cdots x_m.xM_1 \cdots M_m \in NF$ where $n, m \geq 0$.
2. (a) $x \in NF$.
   (b) $M_1, \ldots, M_m \in NF \implies xM_1 \cdots M_m \in NF$ with $m \geq 0$.
   (c) $M \in NF \implies \lambda x.M \in NF$.

Proof. 1. Let $\mathscr{X}$ be the set determined by the inductive definition. $\mathscr{X}$ must be s subset of the set of normal forms. So let us say that M is a hnf. By Proposition 5 in Section 0.5, M is of the form $\lambda x_1 \cdots x_n.xM_1 \cdots M_m$. Moreover, the $M_1, \ldots, M_m$ are again in nf. By induction on the length of M it follows that $M \in \mathscr{X}$.

2. Both inductive definitions define the same set, so the second definition must also define all normal form terms.

□

Corollary 7. M has a nf $\iff \lambda\overrightarrow{x}.M$ has a nf.

Proof. $\implies$ Suppose $M = \lambda\overrightarrow{z}.y\overrightarrow{N}$. Then $\lambda\overrightarrow{x}.M = \lambda\overrightarrow{x}y.y\overrightarrow{N}$ which is a normal form.
$\impliedby$ Suppose $\lambda\overrightarrow{x}.M \twoheadrightarrow \lambda\overrightarrow{z}.y\overrightarrow{N}$. Then $\overrightarrow{z} = \overrightarrow{x}, \overrightarrow{x}$ and $M \twoheadrightarrow \lambda\overrightarrow{w}, yN\overrightarrow{N} \in NF$. □

A term may have several different hnf's, so we will now define a canonical choice for hnf.

Definition 31. If M has a hnf, then the least term of the terminating head reduction of M is called the **principal head normal form** of M.

Example 10. The principal head normal form of $M \equiv (\lambda x.xx)Iy(Ia)$ is $IIy(Ia)$ as shown by the following head reduction. $\lambda x.xx)Iy(Ia) \to IIy(Ia)$. The read reduction $(\lambda x.xx)Iy(Ia) \to IIy(Ia) \to y(Ia)$ shows that $y(Ia)$ is a different head normal form of M.

0.9 Böhm Trees

We will now define a structure called the Böhm tree, which is a way of assigning elements of lambda terms to a tree structure. This emphasizes the relationships between elements of the lambda term and also allows for many important theorems. However, it most importantly allows one to place a topology upon lambda terms. The most important aspect of this situation is that β-reduction is a continuous map. We will not prove that statement here because it is quite involved. Regardless, that means that β-reduction behaves nicely with open sets, which in turn allows us to do many things.

Definition 32. A **tree** is a set of sequences, Seq, such that

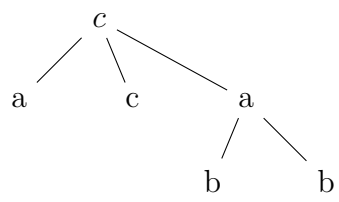
1. $Seq = \{\langle n_1, \ldots, n_k\rangle \in \mathbb{Z} | k \in \mathbb{N}, n_1, \ldots, n_k \in \mathbb{N}\}$.
2. $\alpha \in A$, $\beta \leq \alpha \implies \beta \in A$
3. $\alpha * \langle < n+1\rangle \in A \implies \alpha * \langle n\rangle \in A$

Upon trees we define an ordering $\leq$ where if $\alpha, \beta \in Seq$ and $\alpha \leq \beta$, then the length of α is less than the length of β, and we can add elements to α to make it β. Therefore, $\alpha \leq \beta$ if and only if we can extend α to make it β

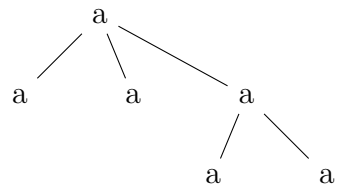
If $\alpha = \langle n_1 \ldots, n_k\rangle \in Seq$, then $lh(\alpha) = k$. Therefore, lh is just the length of the sequence. To concatenate sequences we will use the symbol $*$. Therefore, if $\alpha = \langle n_1, \ldots, n_k\rangle$ and $\beta = \langle m_1, \ldots m_l\rangle$, then $\alpha * \beta = \langle n_1, \ldots, n_k, m_1, \ldots m_l\rangle$.

Definition 33. Let Σ be a set of symbols. On the tree we will attach a map $\phi : X \hookrightarrow Y$ with domain $Dom(\phi) \subseteq X$. For $x \in X$, $\phi(x) \downarrow$ will mean that $\phi(x)$ is defined and $\phi(x) \uparrow$ will mean that $\phi(x)$ is not defined. A given tree with a partial map $\psi : X \hookrightarrow Y$ where $X = Seq$ and $Dom(\psi) = T_\psi = \{\alpha \in Seq \psi(\alpha) \downarrow\}$ and Y is the set of symbols in Σ is called a **Σ-labeled tree**. The set of sequences will be called the **naked tree** and $\psi(\alpha)$ will be the **label** at node α of the tree.

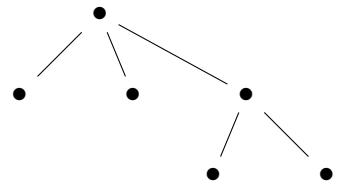
Example 11. If we let $\Sigma = \{a, b, c\}$, the following is a $\Sigma-$labeled tree.



However, there is no unique way of labeling the tree as we could have also labeled it as



The underlying tree in this case would be



We will use capital letters to denote specific partial maps., For a partial map A giving a labeled tree, $|A|$ is the underlying tree that A has as a domain.

Now we will define a way to write out the Böhm tree of a particular lambda term.

Definition 34. Let

$$\sum = \{\bot\} \cup \{\lambda x_1 \cdots x_n.y | n \in \mathbb{N}, x_1, \ldots, x_n, y \text{ variables}\}$$

The Böhm tree of a λ-term M (notation BT(M)) is the $\sum$- labeled tree defined as follows.

1. If M is unsolvable, then

$$BT(M)(\langle\rangle) = \bot$$
$$BT(M)(\alpha) \uparrow \quad \forall \alpha \neq \langle\rangle$$

   Therefore, the first element of the tree is the null element and the tree is undefined for all other sequences other than the initial sequence $\langle\rangle$.

2. If M is solvable, then we define the Böhm tree using its principal hnf. It must have one because all solvable terms have a principal hnf. Let the principal hnf of M be $\lambda x_1 \cdots x_n.yM_0 \cdots M_{m-1}$. We will then define the tree as

$$BT(M)(\langle\rangle) = \lambda x_1 \cdots x_n.y$$
$$BT(M)(\langle k\rangle * \alpha) = BT(M_k) \text{ for all } \alpha \text{ and for all } k < m$$
$$BT(M)(\langle k\rangle * \alpha) = \uparrow \quad \forall \alpha, k \geq m$$

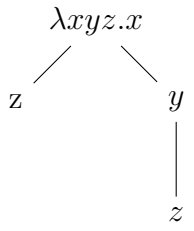
This definition of the Böhm tree is good in that it gives all non-solvable terms the same tree, which is a feature of many models of lambda calculus. ($\bot$ and ω can be considered symbols representing all unsolvable terms.) Moreover, it defines the individual nodes on the tree in terms of the smallest possible elements. Giving each input variable x_i its own node would only give long straight chains in the tree as each time we assigned a node to one variable it would give us a new term with a set of input variables of size $m-1$. Lastly, the construction of the tree is purely mechanical as we can just do head reduction on any term not in its principal head normal form.

Example 12. 1. For $I \equiv \lambda x.x$, BT(I) is

$$\lambda x.x$$

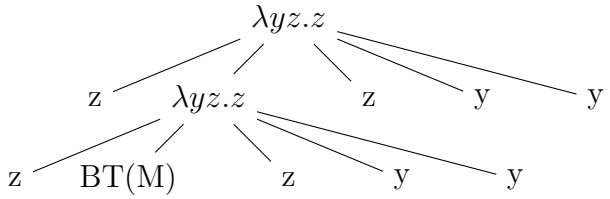
This is because there are no subterms in I.

2. For $S \equiv \lambda xyz.xz(yz)$, the BT(S) is



Here it is important to point out that here we treat terms made just out of variables in a similar way as to regular terms. The root of the tree just becomes the first term, which we already mandated to be a variable.

3. If we let $M \equiv \theta(\lambda xyz.zzxzyy)$ where θ is the Turing fixed point combinator we get the following tree



This recursive possibility is why the finitude of developments is so important. Otherwise if we had the θM as the first element of the head reduction, it would not be obvious how to compute the tree at all or if it was even possible. However, since we do know that such paths are finite, we can say that it would equal something. It would just be harder to figure out what thing would be.

There is also an interesting thing here in that it can be shown that there exist lambda terms that generate all other lambda terms as subterms. Each lambda term is mapped to an index in some way and then reductions of the term produce the new terms. This in turn means that all of these Böhm trees are also merely subtrees of one master tree of infinite size.

Proposition 7. If M=N then $BT(M) = BT(N)$.

Proof. $\implies$ Suppose M=N. We will use induction to show that their Böhm trees are identical at each level and hence at all levels. We first let $lh(\alpha) = 0$, here we at the first level of the trees and $\alpha = \langle\rangle$. If M is unsolvable then so is N, and $BT(M)(\langle\rangle) = \perp = BT(N)(\langle\rangle)$. If both M, N are solvable then they have hnfs of the following form.

$$M \equiv \lambda x_1 \cdots x_n.xM_0 \cdots M_{m-1}$$

$$N \equiv \lambda x_1 \cdots x_n.xN_0 \cdots N_{m-1}$$

with $M_k = N_k$ for all $k < m$. By definition, $BT(M)(\langle\rangle) = \lambda x_1 \cdots x_n = BT(N)(\langle\rangle)$ because they have the same number of x_i by their being equal.

If $lh(\alpha) > 0$, then for some k, α' one has $\alpha = \langle k\rangle * \alpha'$. If M and N are unsolvable, then $BT(M)(\alpha) = \uparrow$ and $BT(N)(\alpha) \uparrow$. If M and N are solvable, then they must have hnfs satisfying the form of the previous case. Therefore, if $k \geq m$, then

$$BT(M)(\langle k\rangle * \alpha') = \uparrow \text{ and } BT(N)(\langle k\rangle * \alpha') = \uparrow .$$

If $k < m$, then

$$BT(M)(\langle k\rangle * \alpha') = BT(M_k)(\alpha') = BT(N_k)(\alpha') = BT(N)(\langle k\rangle * \alpha')$$

by the induction hypothesis. □

The problem here is that we cannot actually be guranteed to compute this because we only established the finitude of developments and not a bound upon them. Therefore, there is no number of head reductions that we can guarantee will result in the head normal form of the lambda term. Therefore, we cannot say that a term is unsolvable since we do not if a few more reductions will give us a normal form. This is the class of objects that are co-recursively enumerable. This means that we can enumerate all elements that are not in this set with normal computation. However, we can never say if an element is an element of the set. If one wants to know about such sets and their properties , Soare's book on Turing degrees is a good resource.

Because of this restriction, we will define a partially labeled tree instead of just a labeled tree where each node in the tree was forced to have an element of the alphabet assigned to it.

Definition 35. Let $\sum$ be a set of symbols.

1. A **partially labeled $\sum$-tree** is a partial map $\phi : Seq \mapsto \sum \times \mathbb{N}$ such that
   (a) $\phi(\sigma)\downarrow \wedge \quad \tau < \sigma \implies \phi(\tau)\downarrow$, and
   (b) $\phi(\sigma) = \langle a, n\rangle \implies \forall k \geq n \quad \phi(\sigma * \langle k\rangle)\uparrow$.
2. The **underlying tree** of a partial $\sum$-labeled tree ϕ is

$$T_\phi = \{\langle \quad \rangle\} \cup \{\sigma \in Seq | \sigma = \sigma' * \langle k\rangle \wedge \phi(\sigma') = \langle a, n\rangle \wedge k < n\}$$

.

3. Let $\sigma \in T_\phi$. If $\phi(\sigma) = \langle a, n\rangle$, then a is the label at node σ. If $\phi(\sigma) \uparrow$, then the node σ is not labeled.

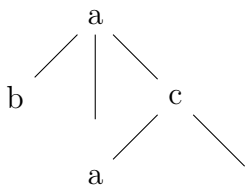
The difference here might only seem syntactic in that we are differentiating between labeling a node in the tree $\perp$ and not labeling it at all. However, the important difference is that for in the case of the regular labeled $\sum$-tree we force the function to output a value even when there is no stop condition. Hence, we assumed the use of an oracle when we were constructing the tree. However, now we are just saying that if ϕ never stops, we don't label it anything. We can do this because we can really just wait until a certain number of computations are complete and then let ϕ never terminate. Given this bound, the tree becomes partially computable, which means that on some inputs the function does not stop or return anything. Moreover, we are delaying the actual computation of the next stage to the next stage, which ensures that we minimize the computation needed at each stage. Lastly, we pair nodes of the tree with the number of children they have, which makes constructions easier as we don't have to look at the actual hnf-form of the element and instead only at the node in the tree.

Proposition 8. T_ϕ is a tree

Proof. If $\sigma \in T_\phi$ and $\phi(\sigma) \downarrow$, then by the definition of partially labeled trees, for all $\sigma' < \sigma$ $\phi(\sigma') \downarrow$, and hence $\sigma' \in T_\phi$. If $\langle\rangle \neq \sigma \in T_\phi$ and $\phi(\sigma) \uparrow$, then $\sigma' = \sigma * \langle k\rangle$ and $\phi(\sigma') \downarrow$. Hence for all $\sigma'' \leq \sigma'$ $\sigma'' \in T_\phi$, so $\sigma'' \in T_\phi$ for all $\sigma'' < \sigma$. Moreover, $\sigma * \langle k+1\rangle \in T_\phi$ then $\sigma * \langle k\rangle \in T_\phi$. □

Just as with labeled trees, we will defined partially labeled trees by capital letters and $|A|$ will denote the underlying tree T_A corresponding to A. We will write $\alpha \in A$ for $\alpha \in |A|$. Lastly we will say $A(\alpha) = \perp$ if $A(\alpha) \uparrow$ and $\alpha \in A$, and $A(\alpha) \uparrow\uparrow$ if $A(\alpha) \uparrow$ and $\alpha \notin A$. This just means that we will differentiate between elements that are a part of the tree and elements that are not part of the tree.

Example 13. Consider the following tree.



On this tree $A(\langle 0\rangle) \downarrow$, $A(\langle 1\rangle) \uparrow$, $A(\langle 2\rangle) \downarrow$, $A(\langle 2,0\rangle) \downarrow$, $A(\langle 2,1\rangle) \uparrow$, and $A(\langle 3\rangle) \uparrow\uparrow$.

Definition 36. Let $\sum_1$ be the set

$$\{\lambda x_1 \cdots x_n.y | n > 0, x_1, \ldots, x_n, y \text{ variables}\}.$$

The **effective Böhm Tree** of M (denoted$BT^e(M)$) is the partially $\sum_1$ labeled tree defined as follows

1. If M is unsolvable, then for all σ

$$BT^e(M)(\sigma)\uparrow$$

.

2. If M is solvable, and has the principal hnf $\lambda x_1 \cdots x_n.yM_0 \ldots M_{m-1}$, then

$$BT^e(M)(\langle\rangle) = \langle \lambda x_1 \cdots x_n.y, m\rangle$$

and for all σ

$$BT^e(M)(\langle k\rangle * \sigma) = BT^e(M_k)(\sigma) \text{ if } k < m$$
$$BT^e(M)(\langle k\rangle * \sigma = \uparrow \text{ if } k > m$$

This tree is clearly a partially labeled $\sum$-tree. Moreover, the underlying tree is the same as in the labeled Σ-tree situation.The first difference between the effective Böhm tree and the regular Böhm tree is that there is extra information at each node saying how many children it has, which does not change the structure of the tree. Secondly, we do not label the unsolvable terms in the effective Böhm tree, which also does not change the structure either, so we still have the same underlying tree.

From this point on we will mean $BT^e(M)$ whenever we say $BT(M)$.

Definition 37. 1. A **Böhm-like tree** is a partially $\sum_1$-labeled tree with $\sum_1$ as in the previous definition. $\mathfrak{B}$ is the set of all Böhm-like trees.

2. $\Lambda\mathfrak{B} = \{A \in \mathfrak{B} | \exists M \in \Lambda \quad (BT(M) = A)\}$

3. $A \in \mathfrak{B}$ is $\perp$-free if and only if $\forall \alpha \in A \quad (A(\alpha)\downarrow)$.

4. If $A \in \mathfrak{B}$, then $d(A) = sup\{lh(\alpha) | \alpha \in A\}$. A could be infinite though, such as in the previous example with the Turing fixed point combinator.

Definition 38. 1. Let $A \in \mathfrak{B}$ and $\alpha \in Seq$. If $\alpha \in A$, then **the subtree of A at node** α (denoted A_α) is

$$A_\alpha = \lambda\!\!\lambda\, \beta A(\alpha * \beta)$$

where $\lambda\!\!\lambda$ just makes this a function. Therefore, the subtree is just all extensions of α in the original tree A.

2. $BT_\alpha(M) = (BT(M))_\alpha$

3. Let $M \in \Lambda$ and $\alpha \in BT(M)$. We define $M_\alpha \in \Lambda$ by induction on $lh(\alpha)$.

$$M_{\langle\rangle} \equiv M$$

$$M_{\langle i\rangle * \alpha} \equiv (M_i)_\alpha \quad \text{if } \lambda \overrightarrow{x}.yM_0 \cdots M_{m-1} \text{ is the principal hnf of M}$$

.

0.9.1 Purpose of Böhm Trees

The idea of these Böhm trees may seem like a strange one, but it is really quite natural because of the key role that trees play in logic overall. For example, trees play a very important role in the construction of certain degrees of computability. In the context of Lambda-calculus, it allows one to search through the subterms of a term in a clear way. Without the concept of these trees, we had no way to index the subterms of a given lambda term or even identify the whole class of meaningful subterms given that new subterms could arise from substitution. Specifically, with Böhm trees one can implement the Böhm-out technique which allows one to access the subtree of a given term. This then leads to the separability theorem.

Definition 39. Let $\mathscr{F} = \{M_1, \ldots, M_p\}$ be a set of λ-terms. If $\mathscr{F} \subset \Lambda^0$, then $\mathscr{F}$ is called **separable** if

$$\forall N_1 \cdots N_p \in \Lambda \quad \exists F \in \Lambda \quad FM_i = N_i$$

The separability theorem states that as long as the λ-terms are distinct then the set is separable.

A much more conceptual idea is that of the topology of the Böhm tree. With the Böhm tree construction we have a way of actually ordering all of the λ-terms in a meaningful way. This in turn means you can add a topology to them. The one most used is the Scott topology.

Definition 40. Let D be a set with a partial order $\sqsubseteq$.

1. A subset $X \subseteq D$ is **directed** if

$$\forall x, y \in X \quad \exists z \in X[x \sqsubseteq z \wedge y \sqsubseteq z]$$

2. D is a **Complete Partial Order**(cpo) if

   (a) there exists a least element $\perp \in D$ called the bottom, and

   (b) for every directed $X \subseteq D$ the supremum $\sqcup X \in D$ exists.

3. Let (D,$\sqsubseteq$) be a cpo. The **Scott Topology** on D is defined as

$$O \subseteq D \text{ is open if}$$

   (a) $x \in O \wedge x \sqsubseteq y \implies y \in O$, and

   (b) $\sqcup X \in O$, with $X \subseteq D$ directed $\implies X \cap O \neq 0$.

This is quite an interesting result all on its own, but is rather limited given that this space is only a T_0 or Kolomogrov space. This means that it only satisifes the axiom that given two points one can construct a set that does not include both points. Namely here one would choose the 'greater' point as the starting point of the space, which would not include the smaller point. However, that is the only type of differentiating set that you can make. You can not create two sets that each contain only one of the points, for example.

0.10 Other Topics in Lambda Calculus

What has been covered here are the essential aspects of Lambda calculus that cover two major things, the types of proof and certain questions that one wants to deal with. However, since its creation in the 1930s, Lambda Calculus has developed in many different directions.

0.10.1 Typed Lambda Calculus

The most important expansion to λ-calculus is the idea of types. These types allow one to restrict things in a much nicer way than with the untyped version of λ-calculus. Theoretically, with a typed version, one could remove the pathological example of Ω. Since it can be shown that Ω is equivalent to any other unsolvable λ-term, it does not add anything, but does prevent one from saying that all β-reduction paths are finite for example. Moreover, it allows one to restrict what can be applied to what, which in turn makes the analysis much easier. One does need to worry about the most general case when proving things about terms. It also allows one to represent data types in the computation much more easily.

0.10.2 Functional Programming

Functional Programming is a programming paradigm directly inspired by λ-calculus. One of the major benefits that people see with functional programming is that things are much more mathematical than in object oriented or procedural based programming, the other two major programming paradigms. Every single object is an atomic object which you then deal with. With the other programming paradigms the objects are mutable, and you can be forced to deal with the actual hardware of the computer and not just abstracted software. A great example of this is all of the memory problems people have with C. Because C allows you to access the hardware almost directly, you can have lots of problems where the memory is not handled in a safe way. This can introduce many security problems and for most programs does not offer any improvement. Computers have largely gotten fast enough that usability and the maintainability of code can be a much bigger factor in the usefulness of a language than the sheer speed of a language.

The 'purest' programming language to play around with the ideas of functional programming is LISP, which was first made in 1958 and is still in use today. This makes it one of the oldest currently used programming languages. However, there are many others such as Haskell and Scheme.

0.10.3 Reduction Techniques

A question that arises from the idea of β-reduction is which reduction method is best or takes the least amount of steps to reduce the λ-term to head normal form. This is a topic that has been covered to a certain extent in this overview, but it is much deeper. This is where the labeling techniques previously used have the greatest relevance.

This topic can also extend into what type of reduction rules are useful. The given β-reduction has been very useful and is able to do a lot of things, but that does not mean that other reduction rules are useless. Here is also where we can get an expansion into not just

λ-calculi but to other calculi made to model different situations. The most famous of these is the π-calculus, which is used to model more parallel situations. This is useful because it can be shown that in some sense λ-calculus is inherently serial. Hence, it was useful for modeling the ideas of previous computing paradigms, but that is less so the case to do because of the massive amounts of parallelism in current computers. The general topics of this area is the area of process calculi.

0.10.4 Models

This is the most mathematical of the different areas that branch off form lambda calculus. Here, we explore different structures that have the applicative structure of regular lambda calculus. Here what is stressed is the applicative structure of lambda calculus and not the computational structure. There are many different known structures that map the behavior.

One especially important example is the combinatory logic of Curry. Here instead of the variable being the key building block, we have the term itself being the building block. Therefore, we have a few axiomatic terms and these combine in different ways to make new terms. Therefore, there is no abstractor or sense of a free or bound variable at all here. There is only the term. Specifically, for Curry's version there are only three terms named $S, K,$ and I, from which one can do all computation. Along with the axioms of who these three terms interact with each other and a few extra axioms one can essentially have an equivalence with λ-calculus in terms of an isomorphic translation between the two.

0.11 Further Reading

The content of this has been directly adapted from *Lambda Calculus: Its Syntax and Semantic* by Henk Barendregt, which is seen as the core text of this field of study. For untyped lambda calculus, it includes all of the major topics. Another treatement is *Lambda Calculus and Combinatorics: An Introduction* by J. Roger Hindley and Jonathon P. Seldin, which also talks about typed lambda calculus. If one wants to focus more on typed lambda calculus in general, then the sequel to Barendregts first book on the topics *Lambda Calculus with Types* proves a definitive covering of that topic. *Domains and Lambda-Calculi* by Roberto M. Amadio and Pierre-Luois Curien provides a category theoretic approach to the topic. If one does wish to delve more into lambda calculus, learning category theory is quite important. This is becuase the abstract structure can be expressed well in terms of category theory. The main books for this are *Categories for the Working Mathematician* by Saunders MacLane.

Bibliography

[1] Barendregt, H. P. H. P. *The lambda calculus : its syntax and semantics*, rev. ed.. ed. Studies in logic and the foundations of mathematics ; v. 103. North-Holland ; Sole distributors for the U.S.A. and Canada, Elsevier Science Pub. Co., Amsterdam ; New York : New York, N.Y., 1984.