15/20

A moderate amount of work needs to be done before submission in fall

15/20 x 2 = 30
+60
90

* Improve intro - need rationale + aim at start of paper
* Work on simplification of explanation

Lambda Calculus and the Y Combinator: Creating Functional Recursion in a Language Without It

Still very difficult to understand! ☺

Erik Boesen, Candidate gsg256

June 7, 2018

Ms. Jennifer Jayson & Mr. William Snyder

IB High Level Math

George Mason High School

Abstract

We explain the basics of the lambda calculus, an alternate system of mathematical representation and logic created in the 1930s by Princeton mathematician Alonzo Church. We take advantage of this new intuition to explain the Y combinator, a simple yet powerful construct for simulating the process of recursion in a mathematical language lacking such a concept or even the ability to name functions.

Contents

1 Where did the Lambda Calculus come from? 2
1.1 A brief history 2

2 Lambda Notation 3
2.1 Comparisons to conventional notation 3
2.2 The basics 3
2.3 Currying 4

3 Illustration: Defining Exponentiation 5

4 Implementing recursion in the lambda calculus 6
4.1 What is recursion? 6
4.2 The Y Combinator 7

5 Conclusion 8

6 Reflection 8

Intro needs to include your rationale (reasons for selecting this topic) and aim (goal of the paper)

make this more explicitly clear before writing— refer to conclusion you have stated both rationale + aim in your conclusion

1 Where did the Lambda Calculus come from?

The lambda calculus (sometimes notated "λ-calculus") is a system of mathematical notation and logic developed for the study of the elementary properties of functions [19].

1.1 A brief history

The notation and concept of the lambda calculus was first described in the 1930s by Alonzo Church [5].[1] Despite lacking many attributes of standard maths, including basic operators and even a native concept of numbers, the system can be used to represent any computation traditionally possible. Even complex computations such as traditional differential calculus [9][2] can be represented using the lambda calculus. So, it is equivalent in capability to a "Turing machine"—a mathematical notion of a computer. Both Church and Alan Turing, who proposed the idea of Turing machines, sought to solve the *Entscheidungsproblem*. Translating to "decision problem", this question was posed by logician David Hilbert in 1928, and asked whether an algorihm can be devised to take as input a first-order logical construction, and output a boolean (true or false, yes or no) stating whether that logical construction is universally valid [10]. Both Church and Turing proved that such an algorithm could not exist, but did so independently and in different countries. Church used the new construct of the lambda calculus to respond to the problem, while Turing used the competing—and comparatively simple—construct of the Turing machine.[3] The joint negatory answer given by the two computer scientists to the *Entscheidungsproblem* is encompassed under the famous "Church-Turing thesis" [17]. Unlike Turing machines, the lambda calculus never uses internal state to make decisions: functions cannot store data, and the same inputs will always yield the same outputs.

More recently, the system and its logical process has gained new notoriety as the basis for new forms of computer programming. The lambda calculus is a "universal programming language" [19], and can, on its own, represent and perform the computations of a Turing machine and vice versa, even without a computer to run it. The lambda calculus is the basis for functional programming languages, or those whose attributes include treating functions as data and using a declarative programming paradigm[4] [12].

[1] Church later published a revision [6] of the system after other researchers criticized it as logically inconsistent [14]. This revision brought the calculus to roughly its current form.

[2] This being the calculus learned by millions of students each year in high school classrooms.

[3] Interestingly, despite their sometimes conflicting ideas on computation, Turing studied as a Ph. D. candidate under Church from 1936 to 1938 [1].

[4] Declarative programming refers to the style of programming or mathematical notation where logic, not "control flow," of a program is defined [15]. Rather than specifying a sequence of instructions, a declarative programmer would specify what logic functions contain, not necessarily in a linear manner. In short, while non-declarative ("imperative") programming involves determination of what a function should *do*, declarative programming emphasizes what a function *is*.

The topic of functional programming is beyond the scope of this paper. Interested readers, however, are invited to investigate the topic through papers including [8] and [12].

2 Lambda Notation

2.1 Comparisons to conventional notation

The notation used to describe lambda calculus expressions seldom bears intentional resemblance to more traditional systems of mathematical notation. To describe, for instance, the identity function (in which a single parameter is given as input and is then returned without modification) in traditional notation, one would likely use some variation upon:

$$f(x) = x$$

To make the function anonymous—remove its name—in traditional notation, one could use a "map" expression [20], which simply lists the parameters to a function and, following the map symbol $\mapsto$, gives the function's internal logic:

$$x \mapsto x$$

Functions in traditional mathematics can easily be translated to use anonymous mapping. For example, the function for finding the length of a three-dimensional vector

$$L(x, y, z) = \sqrt{x^2 + y^2 + z^2}$$

could become

$$(x, y, z) \mapsto \sqrt{x^2 + y^2 + z^2}$$

2.2 The basics

Understanding this syntax, the transition to lambda calculus notation is relatively intuitive. Let us return to our identity map $x \mapsto x$ from above. In the lambda calculus, an anonymous identity function would be notated thus:

$$\lambda x.x$$

In addition to variables, which, as in conventional mathematics, can be any symbol, only two operational symbols in the lambda calculus are defined. These are λ and . (some other symbols, such as parentheses, are used intuitively for grouping and other purposes). The former is used to represent a function definition, whereas the latter separates parameters from that function's internal logic. The sum of those two components is referred to as an "abstraction," which is roughly a synonym for a function definition.

The lambda calculus also defines "application," the passing of a function to a parameter to a function [11]. Lambda calculus expressions associate left-to-right. So, lambda abstractions can be applied simply by parenthesizing the abstraction and appending a parameter afterward.

After one constructs a lambda expression, one can solve it through the process of "reduction." The most prevalent form of reduction is the process of substituting a parameter into a lambda function. That process is referred to as β-reduction (beta-reduction) [13]. The first is α-conversion (alpha-conversion), in which one variable name is substituted for another to prevent naming collisions, as in $\lambda x.B[x] \rightarrow \lambda y.B[y]$, where B represents the

It would be helpful to do some very simple examples to illustrate to the ignorant reader how this process is applied - then expand upon the idea - constantly showing examples...

lambda body [13]. [5] The final reduction operation, and the most complex of the three, is η-conversion, which states that for a bound variable x, $\lambda x.Bx \rightarrow B$. Stated more transparently, if there exists a lambda abstraction which does nothing beyond applying another abstraction to its single parameter, the outer abstraction can be removed in favor of a direct use of the wrapped one [3].

As an example of the reduction process (β-reduction), to apply our identity function to some constant C, we would perform the following lambda computation:

$$(\lambda x.x)C$$
$$= x[x := C]$$
$$= C$$

a few more examples please :)

You will notice that we here make use of the syntax $x[x := C]$. This is an idiom used during β-reduction to make clear the process of substitution being undertaken.

A final important distinction in the lambda calculus is that between bound and free variables. A bound variable is one which is a parameter to the lambda abstraction, whereas a free variable is any other variable. So, in the expression

$$\lambda\theta.\theta\xi$$

θ is a bound variable, and ξ a free variable because while θ is a parameter to the abstraction given, ξ is not [18].

[5] The need for α-conversion may be mitigated through use of a De Bruijn index, which we will for the moment ignore [4].

2.3 Currying

The lambda calculus' syntax for representation of multi-parameter functions is potentially surprising. Purely speaking, there is no notion of multi-parameter functions. Rather, the lambda calculus makes use of "currying," a technique named after mathematician Haskell Curry. In this process, a function which would typically require multiple parameters transforms into a series of functions requiring only one parameter each. So,

$$\lambda xy.xy$$

would expand to

$$\lambda x.\lambda y.xy$$

These two expressions are equivalent, assuming that the impure syntax used in the first represents one function with three parameters x, y, and z. Let us see how this expression would be applied:

$$(\lambda x.\lambda y.\lambda z.x(yz))ABC$$

The outer abstraction would be applied first, and the single parameter it accepts, x, would be substituted into the body of the enclosed function. So, the above expression would first simplify as follows:

?

$$(\lambda y.\lambda z.A(yz))BC$$
$$(\lambda z.A(Bz))C$$
$$A(BC)$$

need explanations as you go through step by step. This is not obvious to someone new to topic

3 Illustration: Defining Exponentiation

The important thing to note about the aforementioned syntax is that in pure lambda calculus, there are no other symbols or syntatical constructions used. This leads to some occasionally verbose definitions for otherwise simple concepts, which must be redefined before they can be used. One must go through a frankly absurd number of operations in order to perform the simplest of mathematical processes. To illustrate, let us walk through the process of defining exponentiation. To any remotely experienced mathematician the operation is doubtless well known. In the lambda calculus, however, we must do a great deal of work to define this capability.

First, we must define what numbers are.[6] An easy way to use numbers in the lambda calculus is through "Church numerals." Church numerals are actually themselves lambda abstractions, which take in a function and compose it n times. Church numerals are notated with an underline [2].

$$\underline{0} = \lambda f.\lambda x.x$$

$$\underline{1} = \lambda f.\lambda x.fx$$

$$\underline{2} = \lambda f.\lambda x.f(fx)$$

$$\underline{3} = \lambda f.\lambda x.f(f(fx))$$

$$\underline{4} = \lambda f.\lambda x.f(f(f(fx)))$$

$$\ldots$$

[6]Yes, seriously.

The first step, having now invented numbers, towards exponentiation is the "successor" function. This lambda expression takes a function f as input, then composes it so as to return, for input n, $n+1$.

$$\lambda n.\lambda f.\lambda x.f((nf)x)$$

Building upon the successor function, we can define addition as follows:

$$\lambda m.\lambda n.\lambda f.\lambda x.(mf)((nf)x)$$

Put simply, we execute succession n times to add n to the numeral m.

Building off this definition of addition, we can define multiplication as simply repeated addition. We'll need to use α-conversion, renaming variables in the aforementioned addition function:

$$\lambda a.\lambda b.a((\lambda m.\lambda n.\lambda fx.(mf)((nf)x))b)\underline{0}$$

This expression is clearly convoluted and difficult to interpret. This is a key flaw within the pure lambda calculus. Without the ability to name abstractions, simple expressions quickly become lengthy and inconvenient to deal with.[7]

[7]It should be noted that some sources, including [?][19] and others, as well as in most functional programming langauges based upon the lambda calculus, mitigate this issue by naming lambda expressions and substituting in the expressions when solving. So, rather than the convoluted expression $\lambda a.\lambda b.a((\lambda m.\lambda n.\lambda fx.(mf)((nf)x))b)\underline{0}$, one could define $ADD = \lambda m.\lambda n.\lambda fx.(mf)((nf)x)$ and then write multiplication as $\lambda a.\lambda b.(ADDb)\underline{0}$. However, this is not pure lambda calculus.

The relationship between addition and multiplication is roughly the same as that between multiplication and exponentiation. We simply need to repeat multiplication:

$$\lambda x.\lambda e.x((\lambda a.\lambda b.a((\lambda m.\lambda n.\lambda f x.(mf)((nf)x))b)0)e)1$$

In mathematical literature, occasionally conventional mathematical syntax will be mixed into lambda expressions. Strictly speaking, pure lambda calculus cannot use even the most basic of operators like $+$, $-$, $\times$, $\div$, exponents/roots, etc. However, for legibility and terseness, these symbols are often used. Reasonably so; it is far easier to understand this equivalent of the three-dimensional vector length formula:

$$\lambda x.\lambda y.\lambda z.\sqrt{x^2 + y^2 + z^2}$$

Conveying this operation as a pure lambda expression would require a prohibitively large quantity of space and writing.

4 Implementing recursion in the lambda calculus

4.1 What is recursion?

As has been previously alluded to, one of the most bizarre characteristics of the lambda calculus is its lack of natural support for naming functions. A consequence of this deficit is that the process of "recursion" is not as easy as one might be accustomed to.

Recursion is a technique used frequently within the field of computer science, but it finds much use within mathematical operations as well. A classic example of recursion is a redefinition of the factorial operator:

$$factorial(x) = \begin{cases} x \times factorial(x-1) & x > 0 \\ 1 & x = 0 \end{cases}$$

This function may appear initially confusing. An illustration may help. Let us take the factorial of the integer 4. First, we begin with our initial call to the function:

$$factorial(4)$$

As $4 > 1$, the first case is met, so:

$$factorial(4) = 4 \times factorial(3)$$

From here, we can clearly see how the expression will expand.

$$factorial(4)$$

$$= 4 \times factorial(3)$$

$$= 4 \times 3 \times factorial(2)$$

$$= 4 \times 3 \times 2 \times 1 \times factorial(0)$$

Since evaluating $factorial(0)$ uses the special "base case" of $factorial$, the expression will evaluate to:

$$4 \times 3 \times 2 \times 1 \times 1 = 24$$

Knowing, however, that the lambda calculus does not allow us to name abstractions, we must use an alternative construction known as the "Y combinator" to achieve our end in defining recursion.

4.2 The Y Combinator

Since lambda expressions are anonymous, no strategy for programatic recursion using the notation is immediately evident. However, it is actually possible to create recursion even without naming any functions. To do so, we must implement the Y combinator [16]. The Y combinator originated in the lambda calculus as a useful tool for simulating recursion even in a calculus lacking names or otherwise not allowing for recursive computation. This technique was discovered by mathematician Haskell Curry, whom you may remember as the namesake of the process of currying [7].

To understand how the Y combinator works, we need to recall how lambda expressions associate. Lambda expressions are "left-associative," meaning that the leftmost abstraction in an application will be evaluated first, and expressions to the right of one expression will be used as parameters to that on the left. *Any* expression can be substituted as a parameter to a function. So:

$$(\lambda x.xx)\lambda x.x$$

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

$$= \lambda x.x$$

The Y combinator may be expressed as follows:

$$\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

To demonstrate how the Y combinator creates recursion, take the lambda abstraction $\lambda\omega.\omega$ (which, through α-conversion, is equivalent to our identity function $\lambda x.x$ from earlier).

We start by applying the Y combinator, as above notated, to our abstraction.

$$(\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))(\lambda\omega.\omega)$$

In this instance, the Y combinator $(\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))$ accepts a single parameter of our lambda abstraction $\lambda\omega.\omega$ as parameter f. We can use β-reduction to reduce the expression by substituting $\lambda\omega.\omega$ for f where it occurs within the Y combinator.

$$(\lambda x.f(xx))(\lambda x.f(xx))[f := (\lambda\omega.\omega)]$$

$$(\lambda x.(\lambda\omega.\omega)(xx))(\lambda x.(\lambda\omega.\omega)(xx))$$

At this point we need to substitute the second outer lambda abstraction (not our identity) as the parameter x to the first.[8]

$$(\lambda\omega.\omega)(xx)[x := (\lambda x.(\lambda\omega.\omega)(xx))]$$

$$(\lambda\omega.\omega)((\lambda x.(\lambda\omega.\omega)(xx))(\lambda x.(\lambda\omega.\omega)(xx)))$$

At this point, you will notice that we are left roughly where we were before our second β-reduction, the only difference being that our reductions have left us with a copy of our identity abstraction, which we passed into the Y combinator as parameter f.

Through this process, the Y combinator is able to achieve recursion in a the lambda calculus—no naming required.

To further understand the Y combinator, a fascinating exercise for the reader would be

[8] It is not necessarily to perform α-conversion in this instance because the substitution of our new x replaces all instances of that variable in the expression being applied. Therefore, there is no naming collision.

to reimplement the *factorial* function previously described. Doing so requires implementation of boolean values (true and false), conditional operators, and subtraction. These topics are described extensively in a variety of sources, including [18][2].

5 Conclusion

Through this paper we reviewed the basics of the lambda calculus, a new system of mathematical notation and logic. To summarize what we covered:

- The lambda calculus includes three syntactic rules for constructing "lambda expressions." We can define variables, use "abstractions" to define lambda functions, and use "application" to apply those functions to parameters.
- We can solve lambda expressions through reduction operations, especially β-reduction, or substitution of parameters into the body of a lambda expression.
- We can use these simple rules to define any mathematical operation which can be run by a computer.
- We can implement recursion through a lambda expression known as the Y combinator.

There are quite a few other fascinating techniques encompassed within the area of lambda calculus which are worthy of study. We did not have time to touch on the fascinating topic of boolean logic, which is a fascinating concept crucial in the construction of higher-level abstractions around the lambda calculus. Nor did we discuss the formulation of a type system, a topic crucial to using lambda calculus in the context of functional programming.

The lambda calculus uses a varied system of notation which sometimes involves, as previously mentioned, names for functions. In an effort to keep notation simple for reader understanding, we did not delve into this topic. There is, however, a great deal of literature regarding consistent names for functions which may be used to build larger and more complicated expressions [18]. The same is done for functional programming, which requires naming to abstract confusing logic so that, despite a basis in the lambda calculus, functional programming does not require excessive logic.

We also did not detail the topic of computability, despite this being the problem the calculus was designed to solve. The aforementioned Church-Turing thesis is a fascinating topic worthy of deep study, as well as Church and Turing's work on the infamous halting problem.

6 Reflection

I first chose to research the lambda calculus due to my existing interest in the field of computer science. I've had past experience with the functional programming paradigm, the study of which was a truly transformative

Develop a clear aim-goal and make sure this is clearly articulated in the introduction of your paper. The conclusion may help with this!

Rationale! Make sure this is clearly stated (along with the aim of the paper) in the intro.

experience. Conventional programming—with languages like C, Java, Python, etc.—is structured around traditional mathematical structures. When I learned about functional programming, I occasionally heard references to the lambda calculus, an apparently mysterious and confusing alternate form for mathematical syntax and logic.

Through this Internal Assessment, I had an excellent opportunity to dive into the new and fascinating notion of this calculus, and the incredible idea that any computation can be represented using only two predefined symbols.

I was at times challenged in comprehending the bizarre manner in which the calculus represents operations. It was tempting to impulsively use conventional syntax and operations like addition to build expressions! I eventually, though, figured out how to define exponentiation in terms of multiplication, multiplication in terms of addition, addition in terms of the successor function, all requiring knowledge of an esoteric way to define the concept of Church numerals.

It has been often said that learning functional programming makes one a better programmer. If this is so, I believe that learning and understanding the lambda calculus certainly makes one a better mathematician. I intend to continue researching the lambda calculus and to find new ways to apply it in my work within the fields of Computer Science and mathematics.

Question I never after reading this... Is lambda calculus used frequently? If no - why - it seems overly complicated! If yes - why - Elaborate on the benefits and advantages of lambda calc. over other methods.

References

[1] Alonzo church papers, 1924-1995: Finding aid. Archived 2010-06-07.

[2] CS 312 recitation 26: The lambda calculus.

[3] η-reduction, 2003.

[4] Nicolaas Govert De Bruijn. Lambda calculus notation with nameless dummies: A tool for automatic formula manipulation, with application to the church-rosser theorem. *Elsevier, Indagationes Mathematicae*, 34:381392.

[5] Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):346–366, 1932.

[6] Alonzo Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5(2):56–68, 1940.

[7] Computerphile. Functional programming's y combinator.

[8] D.A.Turner. Total functional programming. *Journal of Universal Computer Science*, 10(7):751–768, 2004.

[9] Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. 2001.

[10] D. Hilbert and W. Ackermann. *Grundzüge der theoretischen Logik*. 1928. Principles of Theoretical Logic.

[11] Susan B. Horowitz. Lambda calculus (Part I).

Well written - but almost too formal - remember - this should be an exploration of your learning in addition to a research paper - add more personal engagement and reflection w/in the paper.

[12] Paul Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3), 1989.

[13] Paul Hudak. A brief and informal introduction to the lambda calculus. 2008.

[14] S. C. Kleene and J. B. Rosser. The inconsistency of certain formal logics. *Annals of Mathematics*, 36(3):630–636, 1935.

[15] J.W. Lloyd. Practical advantages of declarative programming. *Joint Conference on Declarative Programming*, pages 3–17, 1994.

[16] Ayaka Nonaka. The Y combinator (no, not that one), 2014.

[17] Stanford Encyclopedia of Philosophy. The Church-Turing thesis, 1997.

[18] Stanford Encyclopedia of Philosophy. The lambda calculus, 2012.

[19] Raúl Rojas. A tutorial introduction to the lambda calculus.

[20] Harold Simmons. *An Introduction to Category Theory*. 2011.

Eric B.

MATHEMATICAL EXPLORATION

WHAT DIFFERENTIATES EACH LEVEL? WHAT ARE WE LOOKING FOR?

Criterion A: Communication

This criterion assesses the organization and coherence of the exploration. A well-organized exploration includes an introduction, has a rationale (which includes explaining why this topic was chosen), describes the aim of the exploration and has a conclusion. A coherent exploration is logically developed and easy to follow.
Graphs, tables and diagrams should accompany the work in the appropriate place and not be attached as appendices to the document.

Achievement Level	Descriptor	SUCCESS CRITERIA
0	The exploration does not reach the standard described by the descriptors below.	I can't understand it.
1	The exploration has some coherence.	Some coherence - it makes sense
2	The exploration has some coherence and shows some organization.	Not all of introduction, aim, rationale, conclusion included or not coherent. However, there exists an introduction **or** an aim **or** a rationale **or** a conclusion **and** there is some coherence.
3	The exploration is coherent and well organized.	Contains clear introduction, aim, rationale, **and** conclusion. The conclusion answers the aim. Coherent - the teacher doesn't need to pause to figure out what you mean. All diagrams, graphs and figures are appropriately placed and relevant. Diagrams, graphs and other forms of representations have titles and captions.
4	The exploration is coherent, well organized, concise and complete.	Concise - avoid repetition Complete - Student did everything they thought about (no loose ends) There is proper referencing

ANNOTATED COMMENTS BY EXAMINER ON EXAMPLE EXPLORATIONS:

Your writing is...
- organized
- coherent

You clearly included and identified...
- an introduction
- a rationale (which includes explicitly explaining why this topic was chosen)
- describes the aim of the exploration
- has a conclusion

need @ start of paper

Included ...
- graphs
- tables
- diagrams

Has it been...
- proofread by at least one classmate

Criterion B: Mathematical presentation

This criterion assesses to what extent the student is able to:

-use appropriate mathematical language (notation, symbols, terminology)
-define key terms, where required
-use multiple forms of mathematical representation, such as formulae, diagrams, tables, charts, graphs and models, where appropriate.
Students are expected to use mathematical language when communicating mathematical ideas, reasoning and findings, where appropriate.

Students are encouraged to choose and use appropriate ICT tools such as graphic display calculators, screenshots, graphing, spreadsheets, databases, drawing and word-processing software, as appropriate, to enhance mathematical communication.

Achievement Level	Descriptor	SUCCESS CRITERIA
0	The exploration does not reach the standard described by the descriptors below.	No mathematical notation
1	There is some appropriate mathematical presentation.	There is some mathematical notation or terminology or key terms defined (an attempt is made to use relevant mathematical notation or diagrams)
2	The mathematical presentation is mostly appropriate.	Significant progress made toward achieving a level of 3 but mistakes, triviality or inappropriate multiple representations preclude an achievement level of 3. A consistent notation error may result in this award level.
3	The mathematical presentation is appropriate throughout.	Substantial evidence of correct and appropriate mathematical notation throughout Key terms are defined Terminology is correct and clear Multiple representations used to enhance communication Minor errors do not preclude this award level

compare/contrast w/a simpler method to enhance reader understanding

ANNOTATED COMMENTS BY EXAMINER ON EXAMPLE EXPLORATIONS:

You used...
- appropriate mathematical language
- appropriate notation
- appropriate symbols
- appropriate terminology
- define key terms

Can...
- your classmate understands the point of your exploration and explain it to you?
- NO typos or errors!

← This is a very difficult topic - your paper needs to be written so that the info is more accessible !!

Criterion C: Personal engagement

This criterion assesses the extent to which the student engages with the exploration and makes it their own. Personal engagement may be recognized in different attributes and skills. These include thinking independently and/or creatively, addressing personal interest and presenting mathematical ideas in their own way.

Achievement Level	Descriptor	SUCCESS CRITERIA
0	The exploration does not reach the standard described by the descriptors below.	Student did not do the exploration. Student did not express the exploration creatively. Student did not express personal interest of the topic.
1	There is evidence of limited or superficial personal engagement. need a stronger personal	At least one example of a link to a personal experience or thought. There is limited evidence. Student did not explain theorem used or the mathematics.
2	There is evidence of some personal engagement. engagement voice throughout paper.	Able to express idea in personal language or use personal context. The mathematical ideas and examples (1 or 2) might appear, but the student did not really explain it. vague explanations. Complex explanations – need to simplify
3	There is evidence of significant personal engagement.	Able to express idea in personal language or use personal context. Created some examples or presented some ideas that were satisfactorily explained
4	There is abundant evidence of outstanding personal engagement.	The work is original. Worky may be from historical idea and real world situations, for example socio-economic, political awareness. Created some examples or presented some ideas that were explained in depth.

ANNOTATED COMMENTS BY EXAMINER ON EXAMPLE EXPLORATIONS:

Used personal language
◦Indicated personal relevance or relation to own life → work on this.
◦creates and uses own/personal examples
◦shows the qualities of IB learner profile...
-principled
-open-mindedness
-caring
*Displays independent thinking
* Displays Creative thinking ✓
* Addresses personal interest ✓

Criterion D: Reflection

This criterion assesses how the student reviews, analyses and evaluates the exploration. Although reflection may be seen in the conclusion to the exploration, it may also be found throughout the exploration.

Achievement Level	Descriptor	SUCCESS CRITERIA
0	The exploration does not reach the standard described by the descriptors below.	No reflection related to exploration
1	There is evidence of limited or superficial reflection.	Any evidence of reflection relating to the task undertaken
2	There is evidence of meaningful reflection.	Should include the following: reflection relates to the purpose of the task and is meaningful reflection on what has been learnt reflection in conclusion only?
3	There is substantial evidence of critical reflection.	Student must, ideally have continuous reflection throughout the task Should include the following: identify and address issues as piece develops limitations of the work where applicable ideas for extensions reflection on the significance of the findings

ANNOTATED COMMENTS BY EXAMINER ON EXAMPLE EXPLORATIONS:

Your writing ...

- reviews
- analyses
- evaluates *...the exploration*

**Student questions him/herself*

why is Lambda calc useful
what are your thoughts/reactions
to this as compared to other
methods... is it a useful method
or just a fun exercise?

Criterion E: Use of mathematics

This criterion assesses to what extent and how well students use mathematics in the exploration.
Students are expected to produce work that is commensurate with the level of the course. The mathematics explored should either be part of the syllabus, or at a similar level or beyond. It should not be completely based on mathematics listed in the prior learning. If the level of mathematics is not commensurate with the level of the course, a maximum of two marks can be awarded for this criterion.

Achievement Level	Descriptor	SUCCESS CRITERIA
0	The exploration does not reach the standard described by the descriptors below.	
1	Some relevant mathematics is used. Limited understanding is demonstrated.	Not commensurate with the level of the course and no understanding or very little understanding is shown.
2	Some relevant mathematics is used. The mathematics explored is partially correct. Some knowledge and understanding are demonstrated.	Not commensurate with the level of the course but some knowledge is shown and demonstrated.
3	Relevant mathematics commensurate with the level of the course is used. The mathematics explored is correct. Good knowledge and understanding are demonstrated.	Commensurate with the level of the course Mathematics is part of the syllabus or at a similar level or beyond. The Mathematics is **not** completely listed in the prior learning Occasional minor errors may exist but they do not detract from the flow of the mathematics or lead to an unreasonable outcome. Mathematcis used is on HL syllabus **and** also on SL syllabus or at the level of the HL syllabus **and** SL syllabus.
4	Relevant mathematics commensurate with the level of the course is used. The mathematics explored is correct and reflects the sophistication expected. Good knowledge and understanding are demonstrated.	Mathematical sophistication identified. This might include: - applying what you've learned in new situations - testing a conjecture - reflecting on what something makes sense - using logical arguments and counterexamples Mathematics used is on HL syllabus or at HL level only
5	Relevant mathematics commensurate with the level of the course is used. The mathematics explored is correct and reflects the sophistication and rigour expected. Thorough knowledge and understanding are demonstrated.	Mathematical rigor identified. This might include: - a formal proof - using formal definitions, assumptions, and reasoning in a easy to follow manner.

6	Relevant mathematics commensurate with the level of the course is used. The mathematics explored is precise and reflects the sophistication and rigour expected. Thorough knowledge and understanding are demonstrated.	The work is Precise. This means that: - it needs to be completely error free - the student did not make any illogical leaps in their steps - the student "got to the point" and did not get off topic

challenging math - you clearly understand

The mathematics can be regarded as correct even if there are occasional minor errors as long as they do not detract from the flow of the mathematics or lead to an unreasonable outcome.
Sophistication in mathematics may include understanding and use of challenging mathematical concepts, looking at a problem from different perspectives and seeing underlying structures to link different areas of mathematics.
Rigour involves clarity of logic and language when making mathematical arguments and calculations. Precise mathematics is error-free and uses an appropriate level of accuracy at all times.

Eric B

Grading Guide – Juniors Research Project Discussion

Criterion	Score
(5) Topic Discussed in an organized manner. It was clear that the student had good knowledge of the subject.	4 – have some structure in mind before you start speaking.
(5) Use of Time. Class time during this unit was used in a productive and constructive way to complete the assignment. A good rough draft was complete on the day of the discussion.	5

Total: 9 /10 good draft 10:46–10:52

Teacher Comments:

Lambda Calculus – Alan Turing

Construction / Reduction

λ .

Alonzo Church v. Alan Turing

✓

Academic Partners / Remises ...