The Snark, a counterexample for Church's thesis ?

Hannes Hutzelmeyer

Summary

In 1936 Alonzo Church put forward his thesis that recursive functions comprise all effectively calculative functions. Whereas recursive functions are precisely defined, effectively calculative functions cannot be defined with a rigor that is requested by mathematicians. There has been a considerable amount of talking about the plausibility of Church's thesis, however, this is not relevant for a strict mathematical analysis. The only way to end the discussion is obtained by a counterexample.

The author has developed an approach to logics that comprises, but goes beyond predicate logic. The FUME method contains two tiers of precise languages: object-language Funcish and metalanguage Mencish. It allows for a very wide application in mathematics from recursion theory and axiomatic set theory with first-order logic, to higher-order logic theory of real numbers and so on.

The concrete calcule <u>LAMBDA</u> of natural number arithmetic with first-order logic has been defined by the author. It includes straight recursion and composition of functions, it contains a wide range of so-called compinitive functions, with processive functions far beyond primitive recursive functions. All recursive functions can be represented in <u>LAMBDA</u> too. The unary Snark-function is defined by a diagonalization procedure such that it can be calculated in a finite number of steps. However, this calculative function transcends the compinitive functions and presumably the recursive functions. The defenders of Church's thesis are challenged to show that the Snark-function is recursive. Another challenge asks for an example of a recursive function that cannot be expressed as a compinitive function, i.e. without minimization.

They sought it with thimbles, they sought it with care;
They pursued it with forks and hope;
They threatened its life with a railway-share;
They charmed it with smiles and soap.

The Hunting of the Snark (An Agony in Eight Fits), Lewis Carroll 1876

Contact: Hutzelmeyer@pai.de
https://pai.de

1 FUME system of object-language and metalanguage

The author has put forward **FUME** a **precise** system of **object-language Funcish** and **metalanguage Mencish** that overcomes certain difficulties of predicate logic and that extends to a full theory of **types**. In order to describe an **object-language** one needs a **metalanguage**. According to the author's principle metalanguage has to be absolutely precise as well, normal English will not do. There are at least three levels of language:

English	supralanguage	natural	talks about everything
Mencish	metalanguage	formalized precise	talks about object-language
Funcish	object-language	formalized precise	language of mathematics

The essential parts of a language are its sentences. A sentence is a **string** of **characters** of a given **alphabet** that fulfills certain rules. This means that metalanguage talks about the strings of the object-language. The essential parts of the metalanguage are the metasentences (that are strings of characters as well). It is important to realize that the metalanguage talks about the strings of the object-language and nothing but. If one wants to comment on a certain mathematical system that is realized with the use of an object-language one has to take refuge to the supralanguage. As supralanguage is not a formal, precise language, there are no restrictions. One can comment on mathematical systems and one can talk in supralanguage specifically about metasentences, just as metalanguage talks about object-language.

On first sight Funcish and Mencish look familiar to what one knows from predicate-logic. However, they are especially adapted to a degree of precision so that they can be used universally for all kind of mathematics. And they lend themselves immediately to a treatment by computers, as they have perfect syntax and semantics. It is not the place to go into details. Both Funcish and Mencish have essentially the same syntax. Mencish, however, has strictly first-order logic. The **fonts-method** allows to distinguish between object-language (Arial and Symbol, normal, e.g. ∀Λ1[ ), metalanguage (Arial and Symbol, boldface italics e.g. ***Axiom***) and supralanguage English (Times New Roman).

Notice that Funcish and Mencish have a context-independent notation, which implies that one can determine the **category** of every language element uniquely from its syntax, 'wherefore by their *words* ye shall know them' (*fruits* according to Mathew 7.20). The reader may be puzzled by some expressions that are either newly coined by the author or used slightly different from convention. This is done in good faith; the reason for the so-called **Bavaria notation** is to avoid ambiguities.

There are some hints on the front of the author's homepage https://pai.de/ . You will find some a short description in chapter 1. of the pdf-download GeoO1.1.pdf that can be started from 'Geometries of O' on the homepage. There is also a description in the pdf.download GoodbyeAlonzo.pdf hat can be started from 'Church's thesis …' on the homepage. This publication from 2006, however, is not quite up to date in other respects. A complete description of Funcish and Mencish is forthcoming.

'Calcule' is the name given to a mathematical system with the precise language-metalanguage method FUME . 'Calcule' is an expression coined by the author in order to avoid confusion. The word 'calculus' is conventionally used for real number mathematics and various logical systems. As a German translation 'Kalkul' is proposed for 'calcule' versus conventional 'Kalkül' that usually corresponds to 'calculus'. Calcules are given names using some convention that relates to the Greek **sort** names of a calcule, e.g. concrete calcule LAMBDA with sort Λ .

A **concrete calcul**e talks about a **codex** of concrete **individuals** (given as strings of characters) and concrete **functions** and **relations** that can be realized by 'machines' (called calculators). An **abstract calcule** talks about **nothing**. It only says: if some entities exist with such and such properties they also have certain other properties. Essentially there are only 'if-then' statements. E.g. 'if there are entities that obey the Euclid axioms the following sentence is true for these entities'.

Mencish ih the language of the corresponding metacalcules, metasentences talk about ***sentence*** and other strings of Funcish calcules. It containns many metaproperties that classify strings of Funcish, but there are some metafunctions too. In section 5 it will be made use of metafunction string-replacement $\boldsymbol{(\Lambda\,;\Lambda\int\Lambda)}$ where $\boldsymbol{(\Lambda_1\,;\Lambda_2\int\Lambda_3)}$ gives the result of replacing all suitable appearances of the second string $\boldsymbol{\Lambda_2}$ in the first string $\boldsymbol{\Lambda_1}$ by the third string $\boldsymbol{\Lambda_3}$. Furthermor there is a binary metarelation $\boldsymbol{\Lambda\supset\Lambda}$ where $\boldsymbol{\Lambda_1\supset\Lambda_2}$ states that the string $\boldsymbol{\Lambda_2}$ is suitably contained in string $\boldsymbol{\Lambda_1}$.

Mencish allows for a precise defintion of what is usually called an ***Axiom*** scheme or **schema**. It is preferred to talk about a ***sentence*** **mater**. In sections 2 and 5 the metalingual expression ***scheme*** will be introduced and treated with a completely **different** meaning. *As mentioned before, so-called Bavaria notation has been chosen for good reasons. Although it may put up some hardship for the reader in the beginning, it will finally be realized that it gives so much more clarity.*

Funcish allows for higher-order logic by means of ***type*** strings, e.g. ***function-type*** $\Lambda(\Lambda)$ or ***property-type*** (Λ) that one could e.g. put into$\forall\Lambda_1(\Lambda)[$ ... or $\exists_1(\Lambda)[$ ... where the ***function-variable*** $\Lambda_1(\Lambda)$ and the ***relation-variable*** $_1(\Lambda)$ appear.

It is not absolutely correct to say that first-order logic is sufficient for calcule LAMBDA . Like for many other calcules one needs the **implicit definition of functions** . To this end one has to make a little detour to second-order logic, but one can return from that detour anytime. The detour means that one makes use of the purely logical ***Implicition-axiom*** matres allowing for the **implicit definition of functions**. They state the unique existence of functions so that they can be given names (i.e.***extra-function-constant*** strings); subsequently these functions can be used in normal fashion. Afterwards there occur no omnications with $\forall\Lambda_1(\Lambda)[$ or entications with $\exists\Lambda_3(\Lambda)[$ and therefore one again is in the safe world of first-order logic The method is based on ***UNEX-formulo***1) strings, that have to be introduced now.

As opposed to a ***formula*** that must not include the ***variable*** Λ_0 a ***formulo*** must include the ***variable*** Λ_0 . ***UNEX-norm-formulo***2) strings define relations that hold for exactly one value Λ_0 for every booking of the input ***variable*** strings Λ_1 , Λ_2 , … according to the arity of the ***UNEX-formulo*** . It is metadefined as follows in the unary case. This is the first appearance of a metasentence; remember that the boldface italics fonts belong to Mencish that talks about strings of Funcish that uses normal fonts. You also see that the same logic syntax is used in both Funcish and Mencish. Requiring the string $\forall\Lambda_0[\forall\Lambda_1[\boldsymbol{\Lambda_1}]]$ to be a ***sentence*** means that $\boldsymbol{\Lambda_1}$ is a ***formulo*** with exactly the free ***variable*** strings Λ_0 and Λ_1. The second condition means that ***variable*** Λ_2 does not appear bound in $\boldsymbol{\Lambda_1}$.

$$\forall\boldsymbol{\Lambda_1}[[[\textbf{\textit{sentence}}(\forall\Lambda_0[\forall\Lambda_1[\boldsymbol{\Lambda_1}]])]\wedge[\textbf{\textit{sentence}}(\forall\Lambda_0[\forall\Lambda_1[\forall\Lambda_2[\boldsymbol{\Lambda_1}]]])]]\rightarrow$$
$$[[\textbf{\textit{UNEX-norm-unary-formulo}}(\boldsymbol{\Lambda_1})]\leftrightarrow$$
$$[\textbf{\textit{TRUTH}}(\forall\Lambda_1[\exists\Lambda_0[[\boldsymbol{\Lambda_1}]\wedge[\forall\Lambda_2[[(\boldsymbol{\Lambda_1};\Lambda_0\int\Lambda_2))]\rightarrow[\Lambda_2=\Lambda_0]]]]])]]]$$ 3)

Talking about the arity of ***UNEX-formulo*** strings the ***variable*** Λ_0 is not counted. A nullary ***UNEX-formulo*** string has no other ***variable*** , a unary ***UNEX-formulo*** string has one free, a binary ***UNEX-formulo*** string has two other free ***variable*** strings and so on.

Logical ***Axiom***4) of **implicit definition of unary functions** by ***UNEX-formulo***

$$\forall\boldsymbol{\Lambda_1}[[[\textbf{\textit{sentence}}(\forall\Lambda_0[\forall\Lambda_1[\boldsymbol{\Lambda_1}]])]\wedge[\textbf{\textit{sentence}}(\forall\Lambda_0[\forall\Lambda_1[\forall\Lambda_2[\boldsymbol{\Lambda_1}]]])]]\rightarrow$$
$$[\textbf{\textit{Axiom}}([\forall\Lambda_1[\exists\Lambda_0[[\boldsymbol{\Lambda_1}]\wedge[\forall\Lambda_2[[(\boldsymbol{\Lambda_1};\Lambda_0\int\Lambda_2)]\rightarrow[\Lambda_2=\Lambda_0]]]]]]]\rightarrow$$
$$[\exists\Lambda_1(\Lambda)[[\forall\Lambda_1[(\boldsymbol{\Lambda_1};\Lambda_0\int\Lambda_1(\Lambda_1))]]\wedge[\forall\Lambda_2(\Lambda)[[\forall\Lambda_1[(\boldsymbol{\Lambda_1};\Lambda_0\int\Lambda_2(\Lambda_1))]]\rightarrow[\Lambda_2(\Lambda)=\Lambda_1(\Lambda)]]]]])]]$$

1) the capital letters indicate that ***UNEX-formulo*** is not a metaproperty that is effectively decidable like e.g. ***formulo***

2) ***norm*** means ***variable*** strings Λ_0 and consecutive Λ_1 , Λ_2 , Λ_3 …

3) the capital letters indicate that ***TRUTH*** is not a metaproperty that is effectively decidable like e.g. ***sentence***

4) the only initial capital letter indicates that metaproperty ***Axiom*** is related to ***TRUTH*** but decidable

2. Concrete calcule LAMBDA for pinitive functions

Concrete calcule LAMBDA of decimal pinitive arithmetic uses the following alphabet which is not the shortest possible one, but it is tried keep as close to conventional logic language as possible:

Arial 8, petit-number for variables										*Arial 12, normal size numbers for decimal individuals*									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
Symbol 12, general logic symbols,														*special calcule symbols*					
=	≠	¬	∨	∧	→	↔	∃	∀	[	]	(	)	;	∗	#	≤	Λ		□

List of 38 (plus 1 ***extra*** *) characters for ontological* **basis** *of calcule* LAMBDA

sort ::	Λ	
sort-array ::	***sort*** ¦ ***sort-array*** ; ***sort***	
decimal :: number ::	0 ¦ 1 ¦ 2 ¦ …	*correct definition see section 5*
basis-ingredient ::	***sort*** ¦ ***decimal*** ¦ ***basis-function-constant*** ¦ ***basis-relation-constant***	
basis-function-constant ::	Λ( ) ¦ Λ(***sort-array***) ¦ (Λ∗Λ)	*pinitive functions, decimal synaption*
basis-relation-constant ::	#Λ ¦ Λ≤Λ	*pinity, minority*
pinon-catena ::	***pinon*** ¦ ***pinon-catena pinon***	
pinon-array ::	***pinon*** ¦ ***pinon-array*** ; ***pinon***	
pinon ::	0 ¦ 1 ¦ 2 ***pinon pinon*** ¦ 8 ***pinon pinon-catena*** 9	*only 4 cases*

pinon strings are natural numbers that **code** primitive functions, when they replace Λ in ***basis-function-constant*** string Λ( ) or Λ(***sort-array***) resp. : 0 codes the zero function, 1 the succession function. The third case 2 ***pinon pinon*** codes straight recursion, where the left ***pinon*** of intrinsic arity *m* gives the initial value and the right ***pinon*** of intrinsic arity *n* gives the iteration function (the intrinsic arity of the new ***pinon*** is *max(m+1,n-1)*). The last case 8 ***pinon pinon-catena*** 9 codes composition of functions with any intrinsic arity: the left ***pinon*** is the function where the ***pinon*** strings of the ***pinon-array*** are plugged in. The PINITOR calculator that does the calculating is not described here, neither the basic true sentences.

The ***basis-function-constant*** (Λ∗Λ) gives the decimal synaption of two strings, which is basically concatenation, except that no leading 0 is admissible. Actually the definition among the ***basis-ingredient*** strings is redundant, as could be given by a ***pinon*** . The same is true for ***basis-relation-constant*** strings #Λ and Λ≤Λ as they can be defined using some ***pinon*** strings Λpiny[1)] and Λemiy resp. .

Primitive recursive functions are obtained by ***pinon*** strings, these precede as codes the ***basis-function-constant*** strings Λ( ) and Λ(***sort-array***) . If a number is not a ***pinon*** string the primitive function with this code is simply put to 0 for all input.. Very few examples for coding of primitive recursive functions by decimal numbers are given here (they will be contained in a full publication on the concrete calcule LAMBDA). It is a funny observation that pinitive functions have a Janus face. They have been designed to represent primitive recursive functions

22011(Λ_1;Λ_2)	the addition of two numbers with ***pinon*** Λadd=22011 e.g. 22011(1;1)=2

But the following is defined too and gives a funny function:

Λ_1(0)	the value for all codes at 0 where the result is put to 0 if Λ_1 is not a ***pinon*** code.

The strange functions that can be obtained by putting variables into code position can be generalized to so-called **processive** functions. One realizes that ***scheme*** strings that are obtained from ***function-constant*** strings by inserting ***number*** and ***variable*** strings and compositions thereof represent functions (conventionally they are called *general terms).* The world of processive functions is very rich, e.g. it comprises straightforwardly **Ackermann function** and other **hyperexponentiation**s.

The fact that one does not need minimization for the construction of non-primitive effectively calculative function encouraged the author to look for a counter-example for Church's thesis.

1) one can introduce ***number-constant*** as names by adding a ***medium-letter-word*** subscript to the ***constant*** Λ ; a string a ***pinon*** can be referred to both in Mencish and Funcish, e.g.by ***Λupr*** or Λupr resp.

3. Primitive and minimitive recursive functions

Concrete calcule LAMBDA of decimal pinitive arithmetic allows to define what is meant by a recursive unary function by its representation as a ***UNEX-recursive-norm-unary-formulo***(Λ_1) . A ***UNEX-norm-unary-formulo*** Λ_1 contains exactly ***variable*** strings Λ_0 and Λ_1 and fulfills the condition ***UNEX*** wich means that for every Λ_0 there exist exactly one Λ_1 ; uniqueness is obtained by choosing the smallest possible value (minimization). It is called ***recursive*** if its either ***primitive*** or ***minimitive*** :

$$\forall\Lambda_1[\,[\,\textit{UNEX-primitive-norm-unary-formulo}(\Lambda_1)\,]\leftrightarrow[\,\exists\Lambda_2[\,[\,\textit{pinon}(\Lambda_2)\,]\wedge[\Lambda_1=\Lambda_0=\Lambda_2(\Lambda_1)]\,]\,]\,]$$

$$\forall\Lambda_1[\,[\,\textit{UNEX-minimitive-norm-unary-formulo}(\Lambda_1)\,]\leftrightarrow$$
$$[\,\exists\Lambda_2[\,\exists\Lambda_3[\,[\,[\,[\textit{pinon}(\Lambda_2)\,]\wedge[\,\textit{pinon}(\Lambda_3)\,]\,]\,]\wedge[\,\textit{TRUTH}(\ \forall\Lambda_1[\exists\Lambda_2[\Lambda_2(\Lambda_1;\Lambda_2)=0]]\ )\,]\,]\wedge$$
$$[\,\Lambda_1=\exists\Lambda_2[[[\Lambda_2(\Lambda_1;\Lambda_2)=0]\wedge[\forall\Lambda_3[[\Lambda_2(\Lambda_1;\Lambda_3)=0\ ]\rightarrow[\Lambda_2\leq\Lambda_3]]]]\wedge[\Lambda_0=\Lambda_3(\Lambda_2)]]\ ]\,]\,]\,]$$

It was shown by Kleene that **one minimization** suffices. The definition of ***UNEX-minimitive-norm-unary-formulo*** strings shows that they are denumerable (as finite strings of characters) but not enumerable (meaning effectively denumerable), as it cannot be decided in general if the primitive recursive function ***scheme*** $\Lambda_2(\Lambda_1;\Lambda_2)$ has a zero Λ_2 for all arguments Λ_1 . Therefore recursive functions are not enumerable - and thus do not lend themselves to diagonalization. However, one can say e.g. 'for all unary minimitve functions' as they are given by Λ_3 and Λ_4 with **unary-regularity-condition** $\forall\Lambda_1[\exists\Lambda_2[\Lambda_3(\Lambda_1;\Lambda_2)=0]$

It is sufficient to consider ***UNEX-minimitive-norm-unary-formulo*** strings as ***UNEX-primitive-norm-unary-formulo*** with a ***pinon*** Λ_3 can be expressed as ***UNEX-minimitive-norm-unary-formulo*** strings with the trivial choice: Λ_2 = 82201202201201220120120 19 (that is ***pinon*** Λjsub for the primitive function subtraction *x-y*) and the given Λ_3 .

Onr can define corresponding **minimitive functions** with a ***minimitive-norm-unary-function-constant*** using the logical ***Axiom*** of implicit definition of unary functions by a ***UNEX-norm-unary-formulo*** .

4. Church's thesis

Church's thesis says that all effectively calculative functions are recursive. Whereas recursive functions are precisely defined (as they were defined in the two preceding sections) so that the definition fulfills the criteria of every mathematician, effectively calculative functions are not defined with the precision that is requested by mathematicians, they are not defined within FUME . Church's thesis is not a sentence that belongs to either object-language or metalanguage. It is a suprasentence, meaning that it belongs to supralanguage (in our case English[1]).

One can talk about the plausibility of Church's thesis in (unprecise) supralanguage English. But this is a **never-ending stor**y; as long as only plausibility reasons for the thesis or for its negation are discussed.

To make the story **ending** one has to leave supralanguage. The only way is to put forward a counter-example (one way or the other) for whose correctness all mathematicians can agree on upon. It must obey the criteria that guarantee that a calculation comes to an end after a finite number of **steps** (but keep in mind, that the steps have no general definition either).

Say, somebody has put forward a counter-example by a function (□Λ) , then a special contradiction of Church's thesis reads:

$$\neg\,[\,\exists\Lambda_1[\,[\,\textit{UNEX-minimitive-norm-unary-formulo}(\Lambda_1)\,]\wedge$$
$$[\,\textit{TRUTH}(\ \forall\Lambda_1[\forall\Lambda_0[[\Lambda_1]\leftrightarrow[\Lambda_0=(\square\Lambda_1)]]]\,)\,]\,]\,]$$

It is funny to note that one cannot write down Church's thesis in FUME in general nor its general negation. But one can write it down for a **special counterexampl**e. This is what is meant in the beginning of this section: the never-ending story can be cut short by one counterexample, that can be expressed in proper FUME .Now the full power of FUME is applied to construct a counter-example function.

1) or what the author considers to be English, as his native language is German

5. The Snark

Concrete calcule LAMBDA contains **compinitive** functions, i.e. primitive **and** processive functions. The **Snark-function** (□Λ) is constructed along the idea of outdiagonalizing the compinitive functions. This can be done as the set of compinitive functions is enumerable, meaning effectively denumerable (by the way as opposed to the set of recursive functions that are not enumerable as was mentioned in section 3) . One has to start with an exact definition of ***norm-unary-scheme*** strings.

nu-pattern ::	***number*** ¦ Λ_1 ¦ ***nu-pattern*** ( ) ¦ ***nu-pattern*** (***nu-pattern-array***)
nu-pattern-array ::	***nu-pattern*** ¦ ***nu-pattern-array*** ; ***nu-pattern***

$\forall\Lambda_1$[[***norm-unary-scheme***(Λ_1)] ↔ [[***nu-pattern***(Λ_1)] ∧ [Λ_1 ⊃ Λ_1]]]

A ***norm-unary-scheme*** is built from 13 characters 0 1 2 3 4 5 6 7 8 9 (;) and the14th part Λ_1 ; it can be considered as a ***quadro-decimal*** string constructed from 14 parts: 0 and 13 ***deq-cipher*** characters.

deq-cipher ::	1 ¦ 2 ¦ 3 ¦ 4 ¦ 5 ¦ 6 ¦ 7 ¦ 8 ¦ 9 ¦ (¦ ; ¦) ¦ Λ_1
nonneg-deq ::	***deq-cipher*** ¦ ***nonneg-deq deq-cipher*** ¦ ***nonneg-deq*** 0
quadro-decimal ::	0 ¦ ***nonneg-deq***
de-cipher ::	1 ¦ 2 ¦ 3 ¦ 4 ¦ 5 ¦ 6 ¦ 7 ¦ 8 ¦ 9
nonneg-de ::	***de-cipher*** ¦ ***nonneg-de de-cipher*** ¦ ***nonneg-de*** 0
decimal ::	0 ¦ ***nonneg-de***

Quadrodecimal numbers can be mapped bijectively to decimal numbers. There are two metafunctions Gödel-dedeq-translation *Λ⇑⇑(Λ)* and Gödel-dedeq-cislation *Λ⇓⇓(Λ)* between ***decimal*** and ***quadro-decimal*** strings. Only very few ***quadro-decimal*** strings actually corerspond to ***norm-unary-scheme*** strings. There is a characteristic[1)] primitive function with ***pinon*** Λunsy [2)] that specifies ***norm-unary-scheme*** strings. One has to describe the synaption for the ***norm-unary-scheme*** strings as given above using 14 parts now for decimal numbers that are coding quadrodecimal numbers.

$\forall\Lambda_1$[[***number***(Λ_1)] → [[[***TRUTH***(Λunsy(Λ_1)= 0)] ↔ [***norm-unary-scheme***(Λ⇑⇑(Λ_1))]] ∧
[[***TRUTH***(Λunsy(Λ_1)= 1)] ↔ [¬ [***norm-unary-scheme***(Λ⇑⇑(Λ_1))]]]]]

A choice for the **Boojum-function** (Λ□Λ) where the first position is input and the second codes the ***norm-unary-scheme*** is metalingually defined so that it gives the calculation of this ***scheme*** for the input:

$\forall\Lambda_1$[$\forall\Lambda_2$[$\forall\Lambda_3$[[[[[***number***(Λ_1)] ∧ [***number***(Λ_2)]] ∧ [***number***(Λ_3)]] ∧
[***TRUTH***([[Λunsy(Λ_2)=0]→[Λ_3=(Λ⇓⇓(Λ_2) ; Λ_1∫Λ_1)]]∧[[Λunsy(Λ_2)=1]→[Λ_3=0]])]] →
[***TRUTH***((Λ_1 □Λ_2)=Λ_3)]]]]

One can construct a function that cannot be given by a ***norm-unary-scheme*** string. The Snark-function (□Λ) out-diagonalizes all functions that are given by ***norm-unary-scheme*** strings, as it gives the successor of the diagonal. Therefore it is **not a compinitive** function.

∀Λ_1[(□Λ_1)=1((Λ_1□Λ_1))] *For the Snark was* ***almost*** *a Boojum, you see.*

And everything was envisaged by Lewis Carroll in his poem ***The Hunting of the Snark*** of 1876:

In the midst of the word he was trying to say,
In the midst of his laughter and glee,
He had softly and suddenly vanished away—
For the Snark ***was*** *a Boojum, you see.*

[1)] a characteristic function has only values 0 and 1 representing truth and falsehood or the corresponding relation

[2)] one of these days I perhaps will write down ***pinon*** Λunsy in full beauty as a ***decimal*** . It will be relatively complicated programming, needing many auxiliary functions. Forgive me for being too lazy to do it now.

6. Two challenges

Hello defenders of Church's thesis! You are challenged to show that you can calculate the Snark-function (□Λ) . With the definition of recursive functions in Kleene normal form you have to show:

First challenge:

$$\exists\Lambda_2[\exists\Lambda_3[\forall\Lambda_1[\exists\Lambda_4[[[\Lambda_2(\Lambda_1;\Lambda_4)=0]\wedge[\forall\Lambda_5[[\Lambda_2(\Lambda_1;\Lambda_5)=0]\rightarrow[\Lambda_4\leq\Lambda_5]]]]\wedge[(\square\Lambda_1)=\Lambda_3(\Lambda_4)]]]]]$$

The best thing would be to write down such ***pinon*** strings Λ_2 and Λ_3 . But a proof of existence of Λ_2 and Λ_3 would do as well.

The situation of Church's thesis can be best demonstrated in the following diagram. The **metacursive** functions are those that can be generated **for** calcule LAMBDA by metalingual methods. The **transcursive** functions are those that can be generated in some calculative fashion **for** calcule LAMBDA without taking refuge to implicit definitions via ***UNEX-formulo*** strings, which is necessary for minimitive functions, but including metalingual methods. The **progressive** functions are those that can be represented **in** calcule LAMBDA including implicit definitions via ***UNEX-formulo*** strings.

One should be honest: the expressions **metacursive** and **transcursive** are just **heuristic** ones like **calculative**, as they are not precisely defined as the good ones: primitive, minimitive, processive, recursive, pinitive, compinitive and progressive.

***calculative** functions*				
recursive				
primitive ***A***	*minimitive* ***B***	*minimitive and processive* ***C***	*processive* ***D***	***metacursive E***
		transcursive		
		compinitive		
progressive				

Classification of calculative functions with respect to concrete calcule LAMBDA

One can formulate in Mencish the metasentence that all minimitive functions are processive, meaning that **B** would be empty. However, the answer to this is besides Church's thesis which is the present topic.

Church's thesis has two meanings in the actual context, that

-	there are no metacursive functions	***E*** is empty, the Snark is in ***B***
-	there are no processive functions that are not minimitive	***D*** is empty

Hello again defenders of Church's thesis! You are challenged to show that ***D*** is empty.

Second challenge:

Put forward a ***pinon*** Λ_2 with intrinsic arity 2 and the proof that it has a zero for all input Λ_1 as it is necessary for the definition of a unary minimitive function (see section 3) . And prove that no ***scheme*** exists in LAMBDA to allow for the calculation of that minimitive function.

$$\exists\Lambda_2[[[\mathrm{pinon}(\Lambda_2)]\wedge[\mathrm{TRUTH}(\forall\Lambda_1[\exists\Lambda_2[\Lambda_2(\Lambda_1;\Lambda_2)=0]])]]\wedge$$
$$[\neg[\exists\Lambda_1[[\mathrm{unary\text{-}norm\text{-}scheme}(\Lambda_1)]\wedge[\mathrm{TRUTH}(\forall\Lambda_1[[\Lambda_2(\Lambda_1;\Lambda_1)=0]])]]]]]$$

*Nomini **Alonzo Church** satiram non scribere difficile est. Why, for heaven's sake, has Church this ecclesiastical name ? And Gödel's name has a certain proximity to divinity too. God bless.*