9 God and Computability

9.1 Introduction

I am arguing that, if God exists then God is intelligible and able to be described in terms of the knowledge gained from the creation. I have claimed that God is the one being that can exist alone without any other thing. But if God exists then God is conceivable because if something exists it must be possible to say what it is that exists. It is pointless to make existence claims about things that cannot be described in any sense. What is it that you are claiming to exist? Therefore, associated with existence, is the knowledge to describe what it is that exists. Knowledge comes from the experience of discernible objects in the creation and the relations between these objects. Specifically, I wish to consider mathematical knowledge obtained from our knowledge of objects and their relations in the creation. I will consider mathematical knowledge as being expressed in the properties of sets, sequences and intervals. With the initial ideas of sets we develop ideas of numbers, sequences of numbers, concepts of infinity and intervals or bounded sequences of (infinite) numbers. Mathematics is important with respect to the creation (Nature) because we have findings about the creation obtained by mathematical means.

The question arises as to how the creation can indicate God? If thought arises from our experience of the creation then why can thought lead us to the Creator? Why must the creation relate to the Creator? There is a skepticism that would claim that the creation (Nature) only speaks of

Nature. Why should it be required to speak about anything else? My argument is not that Nature must speak about something beyond it, but rather that it can speak about something beyond it. The challenge I face is to show how talk about Nature and its mathematical properties can be used to talk about God in a way that is intelligible, consistent with the findings of mathematics and able to say something meaningful about God. To do this I use the construct of Creator and creation. I take this to mean that God exists and is the Creator (ex nihilo) of the creation, which consists of discernible objects and the relations between them. From the relations between distinct objects thought is developed which is expressible in some language. The creation provides the capacity for language and various language agents (including humans) devise languages and vocabularies to meet the need to manipulate the environment. There are not only physical, existing, created objects, but there are also conceivable objects, which are the objects of thought. Knowledge is about (universal) objects of thought, which can be readily applied to (particular) physical objects.

So we can go from the existence of God to the thought of God, as expressed by the knowledge obtained from the creation. However, can we go back the other way from the thought of God (as expressed from creation knowledge) to the existence of God? This process of going from thought to existence is what I am calling ontological thinking. Ultimately, I am saying that I cannot obtain the existence of an object from the thought of the object (existence from thought). The reasons for this are as follows. Existence

is a creation act of God. There is no a priori reason why something should (or has to) exist. Creation is ex nihilo so there are no pre-conceived conditions to existence. Another reason is that existence can only be indicated, or pointed to, not invoked by thought. However, there is a connection between thought and existence. Only the conceivable can exist. That which is inconceivable or contradictory cannot exist. (Inconceivable here means not, in principle, expressible in any language.) Therefore, what exists is conceivable in some language. If God exists then (the nature of) God is expressible in some language. That which exists is conceivable; that which is conceivable does not necessarily exist. It is my intention to show that God is conceivable, describable in a language, and that the idea of God is consistent with current mathematical thought. This does not give me the existence of God, but it gives me the indicators for the possible existence of an object. Such indicators are conceivability, intelligibility and plausibility. Briefly these three indicators mean the following. Conceivability means able to be expressed in a language. Intelligibility means that the object (in this case, God) being described in the language makes sense within the language and is a consistent interpretation within the body of knowledge. Plausibility means consistent across other bodies of knowledge.

I am also using Anselm's Definition as a specific example of thought about God. This is done because of what I regard as its essentially mathematical character.

The current topic of God and Computability is an attempt to relate thought about God in the form of Anselm's Definition

with the findings of the mathematical properties of
algorithms.

Having introduced one area of sequence and limit as a way
of thinking about the knowledge of a God, I want to
introduce the area of computability in a general
consideration of Church's Thesis and its relevance to
Anselm's Definition. There is not only an analogy between
Anselm's Definition and Church's Thesis, but the arguments
of computability can be taken as conceptual evidence to
support Anselm's Definition.

With respect to computability, Anselm's Definition may be
regarded as an effective enumeration of a set. The set is
the set of ideas, governed by the > relation, that leads to
the idea of God.

9.2 Church's Thesis

Church's Thesis is the claim that an effective procedure is
recursive (Kleene 1967, p232). Kleene explains (in a
footnote (ibid. p241)), that the converse of this
definition (that if a function is recursive then it is
effective) is often included in the initial definition as
Church originally intended.

The term 'recursion' can have two related meanings. The
more basic and historic meaning is iterative procedure,
whereby using, repetitively, a set of equations, values or
outputs from one evaluation of the equations is fed back as
input values for new evaluations from the same set of
equations. A more particular meaning is that a set is
recursive if it is decidable. This means a mechanism exists

whereby it can always be decided whether a given element is or is not a member of the set. It is important to note that one can determine if the given element is a member of the set. Sometimes it cannot be decided that a given element is not in the set. If it can only be decided that a given element is in the set, then the set is not recursive but only recursively enumerable.

Computability is the mathematical and logical study of computation by machine (Rogers (1967), Cutland (1980)). Computability is the study of what machines can and cannot do. This study commenced in the 1930's before electronic computers were devised. However, the same results apply to current computers as to any mathematical machines no matter how fast or capable such machines can be. The set of instructions (and their execution) is finite but unbounded.

Some of the first results in computability were limitative results, showing what a machine cannot do. The best known and earliest result of this kind is the Halting Problem. This is a proof, discussed later, that one machine, (a Universal machine given a copy of another machine) cannot tell in general whether the given machine will halt. One machine can tell when another machine has halted, but it cannot determine if the machine will halt in the future.

This indeterminacy in deterministic devices like machines is surprising.

Similarly, it was thought that axiomatized mathematics could be 'mechanically' generated. The formalist and finitary program of Hilbert was conceived to formally construct all true and only true propositions of

mathematics. It was envisioned that a correctly programmed (axiomatized) machine could generate and prove all the truths for a specified formal system. Gödel's Incompleteness Theorems showed this programme to be impossible. We can demonstrate that we cannot prove all of mathematics.

This can be interpreted as a limitative result of machines. So how are we to handle machines mathematically? On the one hand mathematics is formal and produces proofs, which guarantee the results. On the other hand, machines are intuitively constructed, and there is a limited guarantee on what will happen.

Church's Thesis is an attempt to link the formal mathematics with the mechanical algorithm, the formal with the intuitive, the determinate with the indeterminate, the totally defined with the simply enumerated.

Church's Thesis is essentially an hypothesis. It cannot be proved because the total behaviour of machines cannot be formally or fully determined.

However all attempts, so far, to characterise algorithmic processes have isolated the same set of computable functions. Kleene has summarised the equivalent sets of computable functions as general recursiveness, λ-definability, Turing computability, reckonable functions and canonical and normal systems (Kleene (1952), p320). But there is no guarantee that an algorithmic process will not be devised that is not recursive. So far a non-recursive effective procedure has not been demonstrated, which seems to support Church's Thesis. But, in keeping with the

character of computability, there may be examples not yet
discovered or invented.

A recursive function, f, is a function that is defined for
all values of the function argument $x \in A$. This is usually
expressed by a characteristic function X that characterizes
the recursive set R by determining whether any element is
or is not an element of R. This can be symbolized for any
element a of **N**, the set of natural numbers, and the
recursive set R as

$$X_R(a) = 1 \text{ if } a \in R$$

$$= 0 \text{ if } a \notin R$$

This means the characteristic function X describes what is
and what is not in R. This function is totally defined.
That is, for any a, X can determine whether a is in R or
not. R is also said to be decidable if X is a recursive
function.

However, consider the characteristic function of a set S
that is determining whether, for a set M of machines, a
given machine $m \in M$ halts or not

$$X_S(m) = 1 \text{ if } m \text{ halts and is a member of } S$$

$$= \text{Undefined otherwise.}$$

This function is only partially defined, X only has a value
for certain m.

How does this relate to Anselm's Definition? As Church's
Thesis links the (formal) recursive mathematics with the
intuithve (informal) algorithm, even so Anselm's Definition
links the idea of God (formal) with the (informal) idea of
a thought sequence or algorithm. Both algorithms (the

machine algorithm and the thought sequence) may be well defined, but they do the work of producing interesting results. The machine algorithm defines (the range of) a mathematical function, and the thought algorithm defines or approaches a definition of God. The intuition that God exists is not just asserted but specifically indicated by a directed thought sequence. Even as Church's Thesis tries to grasp the result that an algorithm cannot be fully guaranteed, similarly Anselm's Definition is attempting to grasp what ultimately cannot be fully understood (namely God). Anselm's Definition, although I am calling it a definition and it contains the phrase 'God is …', does not describe God as such, but offers a method or algorithm for attaining knowledge of God.

But difficulties or barriers of incomprehension or lack of definability do not mean we cannot approach what we can attempt to define. Church's Thesis is a parallel of Anselm's Definition. Both algorithm and thought sequence deal in a defined way with objects that elude strict definition.

A link between Church's Thesis and Anselm's Definition appears also in a certain lack of definition in both. This link will be explored as follows. A machine may be viewed a finite set of instructions. When a machine halts, an input value for the function is defined by the value output by the function. If the machine does not halt the function is not defined for that input value, because the machine does not output a value. A procedure as a set of instructions is well-defined. However, given a set D of input values there is no guarantee that the procedure will produce a value for

every member of D. Consider D ⊆ **N**. If a value is produced for all members of D then the procedure is recursive and the set D is decidable. The set D may be considered as a completed list. Because the list is complete, the effective procedure when presented with any number is able to determine whether that number is or is not on the list or in the set. But we can also have an incomplete list of numbers. When the effective procedure is presented with any number it may or may not find that number on the list. The procedure (or machine) may continue searching the list forever. We still do not know whether the number is on the list or not. This indeterminacy keeps Church's Thesis a thesis rather than a proof. One way around this indeterminacy is to seek the least element in a list that meets certain requirements. But this is still no guarantee an entry does or does not exist.

Similarly, with Anselm's Definition. The thought sequence developed in Anselm's Definition does not have a final element or thought. No matter what thought is produced in the sequence, there is always one thought greater. So we cannot say precisely which thought indicates God. We can only ever indicate the existence of the thought as a limit and not the thought itself. A way around this problem is to follow Cantor and postulate the first infinite ordinal ω. Although ω has a successor it does not have a predecessor so we have at least a candidate for a thought that concludes the thought sequence and indicates God. The indeterminacy in the thought sequence keeps Anselm's Definition as a mechanism indicating a thought about God

rather than as a proof of the existence of what the thought indicates.

Anselm's Definition does not require us to go beyond the first limit ω, so we do not pursue the transfinite. The reason for this is that Anselm is approaching only one limit. He does not require more than one limit for his thought sequence to work. We can also use the Lowenheim-Skolem theorem which states that if any set of statements in a first order language are simultaneously satisfiable they are simultaneously satisfiable in a domain of individuals at most enumerable (Runes (1960), p 184).

The link between Church's Thesis and Anselm's Definition may be seen in the following diagram

	Anselm's Definition	Church's Thesis
Intuitive:	Thought sequence	Algorithm
Formal:	ω-thought	Recursion

Here I am comparing the intuitive with the formal or mathematical. Just as in Church's Thesis the intuitive algorithm is related to the formalistic recursion, so the intuitive Thought sequence is related to the ω-Thought. Even as Recursion formalizes and structures the algorithmic, the Thought sequence formalizes and structures the ω-Thought. Even as Church's Thesis makes the conceptual jump between the informal and the formal even so Anselm's Definition makes the conceptual jump between the thought sequence and the ω-Thought indicating God.

It is worthwhile to explore the indeterminacy further.
Kalmar in an analysis of Church's Thesis argues against
certain features of it (Kalmar, p74f). Kalmar introduces a
theorem of Kleene which states that the recursive function
φ can be used to generate a non-recursive function ψ as
follows

$\psi(x) = \mu y(\ \varphi(x,y)=0 \ ) =$ the least y for which $\varphi(x,y)=0$

$\qquad\qquad\qquad\qquad = 0$ if there is no such y

which says that, given x, if the least value of y such that
$\varphi(x,y)=0$ is found, then $\psi(x)$ takes that value of y. If no
least value for y can be found such that $\varphi(x,y)=0$, then
$\psi(x)$ takes the value 0 (which is understood as undefined).
The above characteristic equation has two conditions.

1. To prove such a y exists. This is done for a given p by
   using φ as a recursive function to generate the sequence
   $\varphi(p,0)$, $\varphi(p,1)$, $\varphi(p,2)$…, until a q is found such that
   $\varphi(p,q)=0$. Then $\psi(p)=q$.

2. To prove such a y does not exist. If this can be done in
   a finite number of steps, then there exists a p such
   that $\psi(p)=0$. This means that the recursive sequence
   $\varphi(p,0)$, $\varphi(p,1)$, $\varphi(p,2)$…, was continued until no y was
   found. This must mean that the set of values of p
   generating the values of φ was exhausted. But the
   theorem states that ψ is not recursive so we are not
   dealing with a decidable set of values under ψ. This

means it is not possible to show that such a y does not
exist, only that such a y does exist.

Kalmar claims that the condition of proving that such a y
does not exist ($\neg\exists y\ \varphi(x,y)=0$) is undecidable, not in the
Godelian sense, but in the yes/no result of a finite
calculation sense. Kalmar concludes that the pre-
mathematical such as 'effective procedure' should be left
as pre-mathematical.

This excursion into the indeterminacy or lack of definition
represented by Church's Thesis and Anselm's Definition is
an attempt to show their similarity in structure and
function.

The following table summarizes the parallel between
Church's Thesis where mathematics is formally attempting to
describe algorithm and Anselm's Definition where thought is
formally attempting to describe God.

Church's Thesis: Recursion captures Effectiveness.

Mathematics	Algorithm
mathematical	mechanical
recursion	calculation
formal	intuitive
proof	procedure
sets	numbers (number-theoretic)
domains	inputs
ranges	outputs
Turing machine	machine
existential	computable / recursive iteration / constructive
Uses the infinite	Remains finite

determinate	indeterminate
limit	process
actual infinite	potential infinite

I explain this chart as follows. The mathematical is compared to the mechanical in this chart. The mathematical deals with recursion, describes the formal, determines proof and uses sets with their domains and ranges. By comparison the effective procedure is characterized by the mechanical, the calculable, the intuitive, the procedural, the use of integers and machine inputs and outputs. Again on the mathematical side, mathematics devises the Turing machine as a conceptual machine as compared with ordinary machines. (Turing in devising his machine actually explained his theory by visualizing how a person processes data (Turing in Davis, p117f)). Mathematical claims are often existence claims where an object's possibility or existence is demonstrated, but how the object is obtained or constructed is not or cannot be given. The mathematical uses the actual infinite whereas the algorithmic is finite and constructible and only approaches the infinite potentially. The sense of the completed or actual infinite is expressed in the mathematical idea of Limit whereas the algorithmic remains essentially as process or sequence.

The comparison with Anselm's Definition is in the following chart.

Anselm's Definition: Thought captures God

(recursive)	**(algorithmic)**
(mathematical)	**(mechanical)**
God (ω–thought)	**Thought sequence**

Limit possible	Sequence only
Existence asserted	constructive only
Completed infinite	Potential infinite
Thought as a boundary	Thought as element only
Infinite set	Finite set
Conceptual	Intuitive
Abstraction	Experience

The above table may be explained as follows. The idea here
is that thought expressed by Anselm's Definition helps us
to understand our intuitive experience of God. We can
decide to remain in the intuitive, constructible,
understandable expression of God or we can decide to use
explicit thought structures developed for mathematics, to
attempt to structure or conceptualize our intuition of
God's existence. In dealing with the idea of God there is
always the temptation to rely on a lack of definition and
retreat into inscrutability. By means of Anselm's
Definition, thought attempts to build a way that approaches
an understanding of the existence of God. Thought attempts
to capture the intuitive with the conceptual, in the manner
of Church's Thesis. So in the above chart, mathematical
thought supplies us with limits, existence statements,
completed infinity, thought as a boundary beyond which
thought cannot go and the Infinite as a limit to the
Finite. This is a response to a purely intuitive expression
of God's existence. This way of thinking is typical of
Intuitionism which (constructively) remains in the
sequence, emphasizes the constructive, deals only with
potential infinity, and treats thought as always reducible
to more basic elements.

God is described by the approaching thought sequence but is
not fully defined. God is conceived intuitively,
experientially and finitely. It is interesting to note that
thoughts about God are finite descriptions but they are
arrived at by thought sequences in-the-limit. As we proceed
through the thought sequence, ideas about God become more
complex and, presumably, indicate God more accurately. So
the infinite property claimed refers to the thought process
not to the nature of God. To describe God as infinite
actually describes how we think about God not how God is.

This explains how we can speak about the unspeakable or
describe the indescribable. All language about God is the
language of the definable thought sequence approaching God.
All language about God is approaching God or approximating
to God.

9.3 Anselm's Definition and Recursion

Another result in Computability is the description of the
recursive procedure. This becomes a way to describe the
thought sequence.

Recursion, following Gödel, is the mathematical description
of the processes of calculation and symbol manipulation
with their presentation. Gödel (Gödel (1992), p51) lists 45
primitive recursive functions including divisibility, prime
number, factorial, n^{th} prime in a list, n^{th} term in a list,
length of a number sequence, variable, elementary formula.
All of these allowed Gödel to build an undecidable formula
in a consistent system. Cutland (Cutland, p51) defines the
primitive recursive functions as the smallest class of
functions that contains the basic functions of zero,

successor, and projection which is closed under the operations of substitution and recursion. The set of general recursive functions is the set of primitive recursive functions plus the operation of minimalisation (that is, $\mu y(f(x,y)=0)$ which means the least y such that $f(x,y)=0$).

On the one hand it is the mathematical that gets us to the ω–Thought, the purely constructive cannot do it. If Anselm's Definition is left at the constructive level we never move into the infinite because the infinite is not constructible. This means that we are dealing with finite, constructible thought. Any thought produced in such a finite thought sequence can always have a greater thought produced. None of these finite thoughts are indicators of God (according to Anselm), because there is always a greater thought, and God is beyond the greatest thought. Moving into the infinite, at least to the first infinite ordinal ω, gives us the ω–thought, which, although in one sense is conceivable and potentially has a greater thought, at least it does not have a predecessor thought (a characteristic of ordinals of the second kind (Sierpinski 1958, p274)).

However, the algorithm of the Thought Sequence, although regarded initially as intuitive, is amenable to a recursive treatment. This is an example of Church's Thesis that the algorithmic is recursive. Initially I have explained Anselm's Definition in terms of the > relation in the context of an ordinal sequence. This ordinal sequence leads to ω and, by the meaning of Anselm's Definition, to the ω–

thought denoting God. However, this > relation, which as a relation simply compares numbers, can be replaced by an operator in an application to recursive functions. I will replace the > relation with the GT (for 'greater than') operator. When we considered the cumulative hierarchy, we used the power set operator P, which constructed the power set P(A) of any given set A. (That is, the set of all subsets of A.) However, we need not use the power set operator. By GT I will mean any operator that constructs a set or concept (in the thought sequence) that can be regarded as greater than the previous set or concept. This may be greater beauty, greater goodness or greater glory.

Kleene (Kleene (1952), p219) defines a function as primitive recursive if the function is derived by means of the following five functions:

The Successor function	$S(x)=x'$,
The Constant function	$C(x)=k$,
The Projection function	$U_i(x_1,...,x_n)=x_i$,
The Substitution function	$h(x)=f(g_1(x),...,g_n(x))$,
The Recursion function	$h(0,x)=k$,
	$h(y+1,x)=g(y,h(y,x),x)$.

Using the input of the natural numbers, **N**, these functions have the following meaning. The Successor function outputs the next number; the Constant function always outputs a constant (for example zero); the Projection function outputs a number from a list; the substitution function allows substitution of functions in functions; and the recursion function describes how a sequence of values can be generated. This suits Anselm's Definition as follows:

```
        T(0) = t₀

        T(y+1) = GT(y, T(y))
```

where t_0 is the initial thought

        T is the Thought Sequence function

        GT is the 'greater than' operator generating the
next term of the thought sequence.

So it seems that Anselm's Definition is expressible as a
recursion. If, by a standard process, we regard thoughts
as able to be coded as numbers, then Anselm's Definition
could be open to other recursive properties. By coding the
thoughts as numbers we are being consistent with treating
the relationship between thoughts in Anselm's Definition as
ordinal.

9.4 Anselm' definition and the Halting Problem

One of the earliest results in Computability theory was the
Halting problem. The problem is: can a machine x, given a
description of another machine y, always tell whether
machine y will halt or not, given a particular input I? If
we treat this machine as a function y, we can ask if we can
always determine whether a function will be defined or not.
Function definition is equivalent to halting or converging.
Therefore when a machine halts the function (which the
machine is calculating) is said to converge, an output is
printed and the function is defined. If the machine does
not halt the function being calculated is said to diverge,
nothing is printed out and hence the function is not
defined for that input value. With respect to an algorithm
(or a machine), when the algorithm receives an input value,

this value is processed by the algorithm and an output value may be obtained and the function defined for that input value. The algorithm outputs (prints) the output value and then stops. When the algorithm outputs a value for a given input value and stops, then the algorithm (function) is defined for that input value. If, however, the machine does not output a value and stop, the algorithm (function) is not defined for that input value. If the machine does not stop, we cannot, in general, determine whether it will or will not stop in the future.

Before we proceed to the proof of the Halting problem I want to explain my reference to it. I briefly want to look at a criticism of Anselm's Definition, which I will call Reductionism. Reductionism is typical of Logical Positivism, Empiricism, Analytic Philosophy, Intuitionism, Constructivism. Because Anselm's definition moves the thought sequence to the limit, reductionist (and constructivist) thinking can refuse to admit this process. The Reductionist simply will not leave the sequence. The jump to the limit is unacceptable as indicating anything real. The grounds on which the reductionist resists the 'in-the-limit' thought sequence is that all acceptable thought has to be constructible, defined and verifiable, or, at least, reducible to acceptable or verifiable elements. To be acceptable usually means demonstrable or provable. If a thought is not provable or constructible with a given set of assumptions it can be treated as meaningless or false. Gödel's Incompleteness Theorems drove a wedge between truth and provability. Any true, provable system of ideas (capable of expressing arithmetic) will

have at least one truth not provable in the system. So that truth is not always guaranteed by proof. The Halting problem is another demonstration of the result that a defined (determinate) system, algorithm or function is not always provable although it may be true.

The reductionist is arguing that only defined (empirically provable) thoughts are admissible and, in particular, thoughts defined by assumptions of their own choosing. This places strong controls on what thoughts are acceptable and what are not acceptable. There is a reductionist 'cut' that refuses any thought not meeting their standard of truth.

The Halting problem demonstrates that not all thought processes (algorithms, machines) can be defined for all input values. An algorithm (which is, in effect, a machine executing a thought sequence) is not always definable, and so is not fully determinate.

Reductionism is an attempt to restrict the domain and range of thoughts to an always defined or verifiable position. It is assumed (with Hilbert) that (mathematical) thought can be totally formalised, strictly controlled, fully defined and axiomatically contained. Gödel's Incompleteness Result showed that this was not possible. There is always a truth lying outside the formally defined system. So a reductionist campaign to reduce all meaningful or admissible thought to a vigorous, formal, totally defined corpus cannot contain every truth although all that it does say may be true (consistent).

Can we always determine what exists by thought? The reductionist wants to restrict what can exist to their

theory. What exists must be perceivable or verifiable. This is theory-driven and theory-restricted existence which I have called ontological thinking. I claim that thought is evidence for existence but not proof. This is because everything that exists is conceivable. I also maintain that what exists is the result of an arbitrary (non-theoretic) ex nihilo, creative act obtained by a creative agent (God).

The Halting Problem is another way to demonstrate this incompleteness in theorizing. The argument goes as follows (following Brainerd and Landweber, p45). Consider the function h defined by

$$h(x) = 1 \quad \text{if } x \in \text{dom } f_x$$

$$= 0 \quad \text{otherwise.}$$

That is, f is defined for all values of the domain of f which are arguments for h; h is any total function and we are assuming all the total functions can be enumerated, f_0 f_1 f_2…. We are considering total functions of one variable.

We shall show that h is not computable.

Assume h is computable. Then we have a g such that

$$g(x) = f_x(x)+1 \pmod 2 \text{ if } h(x) = 1$$
$$= 0 \qquad\qquad\quad \text{if } h(x) = 0$$

That is, if h is computable then g is computable also. To compute g(x) we first compute h(x) by the first equation. When h(x) = 1 then $f_x(x)$ is defined and the machine stops. Then we can calculate $g(x) = f_x(x)+1$. If, however, h(x) = 0 then g(x) = 0. Since g is computable there is a y such that $g = f_y$. That is, g must occur somewhere in the list f_0 f_1

f_2…. Let it be at f_y. (We are assuming that all the total functions in 1 variable can be listed.) Using y as the variable we obtain

$$f_y(y) = g(y) = f_y(y)+1 \pmod 2 \text{ if } h(y) = 1$$
$$= 0 \qquad\qquad \text{if } h(y) = 0.$$

Since f_y is total, $y \in \text{dom } f_y$ so $h(y) = 1$. This gives $f_y(y) = f_y(y) + 1$ which is a contradiction. We conclude therefore that g and therefore h is not computable and that not all total functions are listable or decidable, which is a version of the statement that not all algorithms stop and are defined.

This type of argument can only be used on decidable predicates, where both items in a set and items not in a set can be listed. The way to avoid this kind of argument is to back off the strong form of decidability (recursion) and choose recursive enumerability, where items in a set can be listed but items not in the set cannot be listed.

What is the significance of the Halting problem for Anselm?

The Halting problem is a limitative result. It is telling us what a machine cannot do. Consistently with other arguments such as ordinal sequences, diagonalisation, Gödel's Incompleteness results, we have the fact of undecidable systems, that is, formal systems generating true statements that are neither provable or disprovable within the system. Anselm's Definition can be viewed as a sequence of statements but the assertion concerning God always lies beyond the list. In a sense Anselm's Definition

is a limitative result in that it indicates what cannot be
said within the sequence.

There are two views here about Anselm's Definition. Can we
have finite description of God or can we only have infinite
description of God? Or, can God be described by a finite
language or only by an infinite mechanism of language? This
can be collasped to a two step definition or an infinite
step definition. A two step definition means that whatever
(finite) description or thought I come up with, God is
greater than that thought. We may say God is good and
whatever idea of God you have then God is greater than that
idea of goodness. This does not involve a significant
thought sequence. However, if we wish to imply that no
matter how far you go in your thinking, then you still will
not reach a thought adequate to describe God, something
akin to an infinite process is required. And mathematics
gives us the language to use in this instance. So we may
say that we have minimal and maximal cases of Anselm's
Definition. The minimal case is the two step case. The
maximal is the infinite (limit) step case.

The incompleteness of consistent, axiomatized systems
appears to be a strong criticism of reductionist programmes
that attempt to place all true statements of a theory in a
provable set of statements. Usually the statement is said
to be true because it is provable (testable or verifiable)
in some sense acceptable to the theory.

These limitative results indicate that a consistent formal
system cannot contain all truths as provable and that a
truth can lie outside the proof of the system. With respect

to Anselm's Definition, the limit thought denoting God is not contained in the thought sequence but, as a truth indicated by the system, God, as a truth, can lie outside the sequence (system) as a limit object.

9.5 Anselm's Definition as Recursive

By demonstrating the recursive nature of Anselm's Definition, I wish to show the way Anselm's Definition uses this way of thinking. Recursion is the main technique used (indeed innovated) by Gödel in his Incompleteness Theorems (see 11.1). Recursive Functions are the basic construction in the mathematical presentation of Computability (Cutland, p32). Anselm's Definition can be described as recursive and, if we allow for the numerical coding of thoughts, similar results may be obtained as the results associated with Diagonalisation for Anselm's Definition (see 10.4).

I regard this as evidence for the plausibility of Anselm's Definition arising from associated or similar forms of thought. Questions arise as to how certain objects can be thought about. Considering such an object (or term) as God, can one have anything to say? Anselm's Definition says that we can and by its structure invokes similar language from associated arguments, namely from mathematics and logic.

Basic to the idea of Recursion is the idea of iteration. Iteration is the procedure (algorithm) where a value is obtained as output and then that output value is used as the next input value for the procedure. This looping effect is continued until a certain pre-determined condition is met.

Analogously the thought sequence of Anselm's Definition can be seen as a previous thought being used to generate the next thought in the sequence.

Gödel introduced the idea of Recursion in his 1931 paper (Gödel (1992), p46). Here recursion is used, in a formal system, to build up more complicated formulae from more simple or basic formulae. Recursion, in fact, is a form of proof, in that, if you can show how a formula can actually be constructed it can be accepted as a well-formed formula and may be proven for that formal system. This enables Gödel to build a formula, recursively, from previous simpler formulae which, although constructible in that system, actually can neither be proved nor disproved. This is because proof as a constructible statement can be recursively defined. Hence Gödel claims that this can be done in any system that allows numeric coding (arithmetic) and is sophisticated enough to be able to encode the notion of proof (B and Bw in Gödel's coding (see 11.1)). Hence Proof is by construction and construction is by Recursion.

The actual recursive definition that Gödel gives is as follows (simplifying to one argument x).

$$\phi(0,x) = \psi(x)$$

$$\phi(k+1,x) = \chi(k, \phi(k,x), x)$$

where $\psi(x)$ is the initial value for the recursion, $\phi(k,x)$ is the recursive function, k is the index counting the steps of the recursion and χ is the construction function, which constructs the current k^{th} value of ϕ.

Taking Gödel's definition of recursion we may apply this to Anselm's definition. In Anselm's definition we have an initial thought t_o. As explained in 8.2, the well-ordering of the ordinal sequence of thoughts has a minimal element. t_0 is our minimal element.

Let the thought sequence be S. S holds the values, t, of the recursively generated thought sequence. Let GT(k,t) be the construction operator which takes thought t and constructs a greater thought as expressed by the GT (greater than) operator with k as the index. The GT operator can be regarded as generating an ordinal scale where we formally use ordinal numbers as isomorphic to the gradations of a specified attribute such as beauty or glory.
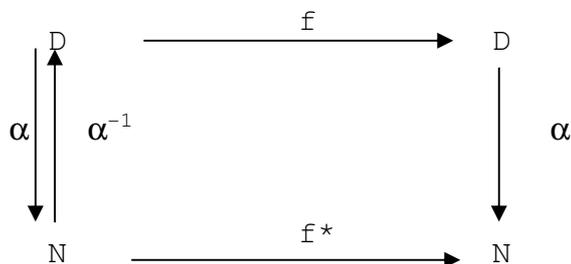

We then have
$S(0, t_0)$ = t_0 :initial thought of sequence
$S(k+1, t) = GT(k, S(k,t), t)$: the recursion over k.

This recursion is designed to work with numbers and computability is used with the integers. These results are called number-theoretic. Computability theory does allow for non-numeric data. Gödel's work itself was basically non-numeric in meaning. However, Gödel worked out a system, now called Gödel numbering, whereby symbols, formulae and proofs could be expressed as numbers. This arithmeticizing of the formal system was the innovation in Gödel's Incompleteness proof.

This raises the possibility of the numeric coding of non-numeric information. Coding of non-numeric objects

(thoughts) could be as follows. We are mapping thoughts
onto numbers. Consider a domain D of objects. The coding is
the enumeration α:D $\rightarrow$ N. Here the object d$\in$D is coded by
the natural number α(d). If f is a function f:D $\rightarrow$ D
(between objects of D) then f* is a coding of f where f*: N
$\rightarrow$ N gives

$$
\begin{array}{ccc}
D & \xrightarrow{\ \ f\ \ } & D \\
\alpha \Big\updownarrow \alpha^{-1} & & \Big\downarrow \alpha \\
N & \xrightarrow{\ \ f*\ \ } & N
\end{array}
$$

which commutes (as a natural transformation).

Here α^{-1}: N $\rightarrow$ D is a decoding.

Following the arrows this shows how f*= $\alpha \circ f \circ \alpha^{-1}$ and f can
be transformed into f*.

So we can certainly code non-numeric data. However, how
plausible is it to numerically code thoughts or objects?
Anselm's thought sequence is defined in terms of the >
relation which is usually numeric. The thought sequence
models the ordinal sequence. It would seem that if the
thought sequence can be formalised Gödel's conclusions and
numbering/coding techniques would apply.

It should be noted that the numeric coding of thoughts is
done only at the ordinal level. If in the thought sequence
S, S(k) comes before S(2k), this does not imply that

thought S(2k) is twice as complicated as thought S(k). The scale is an ordinal scale not a ratio or interval scale.

A powerful example of coding is as follows. To get a formula inside itself Gödel devised recursion using arithmeticization. (This kind of procedure will be discussed in chapter 11.)

I offer the following as another way to express this kind of result. A Gödel numbering is a unique representation of a formula or proof as a number, namely a product of powers of primes.

Let [a] be the Gödel numbering form of an arbitrary formula a. Let b be another formula and let P be a proof involving these two formulae.

P([a],[b])   says a proves b.

P([a],[a])   says a proves itself.

¬P([a],[a])   says a does not prove itself.

If p = ¬P([a],[a]) then P([p],[p]) asserts its own unprovability.

This is an attempt to show how ideas and proofs can be coded as numbers and made part of a number/thought sequence.