ADDENDUM ON TURING MACHINES TO:

Hinged Sets and the Answer to the Continuum Hypothesis

By R. Webster Kehr

March 15, 1999

(Note: This paper is an additional chapter to my other paper: “Hinged Sets and the Answer to the Continuum Hypothesis.” It will be assumed that the reader is completely familiar with the "Hinged Sets" paper, including all of its terminology, concepts, algorithms, mappings, etc.)

Chapter 23: Turing Machines

Introduction:

This chapter will consist of two major parts. First, the concept of Delineation Mapping and other important preliminary results will be presented. Second, there will be two independent proofs that there are a countable number of Turing Machines.

Delineation Mappings:

The cardinal number of R(0,1) and the power set of N are both defined to be "aleph one" or just "aleph." However, in previous chapters I have proven that aleph one and N/o (aleph nought) are equal.

Anyone familiar with Transfinite Set Theory would not be surprised, therefore, that there exist ways to map the power set of N into R(0,1), even though they

have the same cardinal number. A Delineation Mapping will map the power set of N into R(0,1). To do this each subset of the power set of N will be mapped to a unique element of R(0,1).

For Finite Subsets of the Power Set of N:

Because every element of N is defined to be finite, a member of the power set of N that contains a finite number of elements of N can be mapped to an element of Set Rt. For example, let Set pn be a finite set, which is a member of the power set of N:

Set pn = {2, 3, 5, 6, 7}

Let us take each element of Set pn and use this element to create a string. For example, for the third element in the set: a 5, we will create a string of 5 digits. These 5 digits will consist of 4 '1s' followed by a single '2', namely '11112'.

For each element of Set pn, with a value of say n, an element of N, we will create a string segment that consists of (n-1) '1's followed by a single '2'. These string segments will then be concatenated to form a single element of Set Rt. Set pn will then be mapped to this element of Set Rt.

Thus Set pn is mapped to:

Element of Set Rt = .12,112,11112,111112,1111112 (note: the commas are added only for convenience)

Note:
the 2 submaps to the ,12, portion of the number
the 3 submaps to the ,112, portion of the number
the 5 submaps to the .11112, portion of the number
the 6 submaps to the ,111112, portion of the number
the 7 submaps to the ,1111112, portion of the number

The '2' acts as a delineator, thus the term "delineation mapping." For simplicities sake zero will not be considered an element of N. Note that each element of the set is "submapped" to a string segment that consists of (n-1) '1's followed by a single '2'. These string segments are then concatenated to form a single element of Set Rt, to which the entire set maps. For example, a set with 6 elements in it will map to an element of Set Rt which has 6 2s in it along with a number of 1s. There must be 1s because redundancy is not allowed - meaning {2,2,2,2,2,2} is not a valid element of the power set of N.

Any string with a finite number of digits in it can be treated as a valid element of Set Rt. For example, the above element of Set Rt has 23 digits in it, making its width [23]. Likewise the sum of the elements in Set pn is also 23. The sum of

any finite set of finite numbers is finite. Therefore the element of Set Rt, created for any finite subset of the power set of N, will have a finite width, and will therefore be a valid element of Set Rt. That this is a one-to-one "into" mapping is trivial to prove by contradiction, but it is an “into” mapping (i.e. into Set Rt), not an “onto” mapping.

Notice that the order of the elements of the set is irrelevant. In fact every unique permutation of these elements maps to a unique element of Set Rt.

For Infinite Subsets of the Power Set of N:

Because Cantor has proven that the union of a countable number of countable sets is countable, then clearly the union of a countable number of finite sets is countable. Because every element of N is finite, if a countable number of finite strings are concatenated together, one behind the other, the resulting string will have a countable number of digits in it.

We will now map infinite sets of the power set of N into Set Rnt in much the same way we mapped the finite sets of the power set of N into Set Rt.

For example, the set of all 'even' counting numbers, {2, 4, 6, 8, 10, 12, ...}, will be mapped to:

Element of Set Rnt =
.12,1112,111112,11111112,1111111112,111111111112,...
(note: the commas are added only for convenience)

In this case, because every element of N is a 'finite expansion,' meaning it has a finite number of digits, by definition, a countable number of finite expansions concatenated together as shown, can map to an element of Set Rnt, which has a countable number of digits in it. This element of Set Rnt will have a countable number of 2s in it, and a countable number of 1s (this is because redundancy of elements is not allowed, as above). It is obvious that for each infinite set within the power set of N, a unique element of Set Rnt can be mapped to, making this a one-to-one injection or "into" mapping.

A Delineation Mapping maps both the finite and infinite elements of the power set of N into a single countable subset of R(0,1).

Note that the subset of R(0,1) to which the power set of N maps into is an extremely "small" subset of R(0,1). This should come as no surprise to those who are familiar with transfinite mappings.

Other Comments on the Delineation Mapping:

Because elements of the power set of N are not allowed to be redundant, it is obvious that no element of R(0,1) that is mapped to by an element of the power set of N is a repeating decimal. Each element that is mapped to would be considered a non-repeating decimal.

However, suppose we did allow redundancy. To be more specific, suppose we allowed redundancy in a "modified" power set of N and we allowed each set to be ordered in any desired order, as long as each set in the modified power set was unique (taking order into account).

Since, by definition, each set within this modified power set of N must be unique, then the Delineation Mapping will remain a one-to-one mapping. In fact this modified power set of N would continue to map into a very "small" subset of R(0,1).

This redundancy and ordering allows elements of the modified power set of N to map to repeating decimals. For example: {8, 9, 7, 6, 4, 4, 5, 4, 4, 5, 4, 4, 5, …} will map to a repeating decimal. Note how the three elements (4, 4, 5} are an infinitely repeating pattern.

Set Rnt includes both repeating and non-repeating decimals. For example, the above set in the modified power set of N would map to:

.11111112,111111112,1111112,111112,1112,1112,11112,1112,1112,11112,...

Now we have a situation where the subset of R(0,1) that is mapped to includes some repeating decimals (this will be necessary to understand later in the chapter for Turing Machines that end up in infinite loops).

In fact given any element of R(0,1) which contains only 1s and 2s, and for which there is never an infinite number of consecutive 1s (this implies there are an infinite number of 2s by the FNP, but not necessarily an infinite number of 1s because redundancy is allowed), is mapped to by an element of the modified power set of N.

Throughout the rest of this chapter any reference to the "power set of N" will automatically include a reference to the "modified power set of N" depending on context.

Kehr's Extensions to Cantor's CUT:

In this section two theorems will be proven. These theorems are designed to be used with one of the proofs that the set of all Turing Machines (i.e. all Turing Machine programs) are countable.

In previous chapters it was shown that not all countable sets can be mapped to by N. Actually it was shown that for some countable sets a "new number" can always be created that is not mapped to by any specific element of N. This would be interpreted to mean that the DOU is false, in spite of the Pipe Mapping above.

Cantor's CUT uses the CDOU to justify that the row numbers and column numbers of the array are both N. For example, the row numbers of the CUT or DSM array are elements of N. Likewise, the column numbers of the CUT or DSM array are also elements of N. But earlier in this paper the DOU, and thus the CDOU, were proven to be false.

What happens if we use the elements of R(0,1) or the power set of N as the rows or columns of the CUT array? Is the CUT still valid? Two theorems will deal with this issue.

Kehr's First Extension of Cantor's CUT (KFE): "Given any two sets that can be shown directly or indirectly to be countable by the Creation Definition (CD) and a valid form of the Creation Algorithm (CA); if these sets are used as the rows and columns of a CUT array, then the CUT is still valid."

Proof:

Let us prove this theorem for the rows of a CUT array. It is trivial to extend such a proof to the columns or the rows and columns as well.

Let us consider the creation of one of these two sets by some form of the CA. N, or some proper subset of N, can be listed side-by-side with the steps of the CA, by definition. Let us list the elements of N (and/or Set Nv), used by the CA, down the rows of a CUT array, with the evolving elements of a set such as R(0,1) to the left of the elements of N.

For example:

```
R(0,1)      Nv      N

.0          v0      1
.1          v1      2
.2          v2      3
.3          v3      4
and so on
```

Note that by doing this, the elements of N in column #3 above can now be considered the row numbers of the CUT array. In other words, even though N cannot map onto R(0,1) after it is created (for paradoxical reasons), a simple listing of N can be used to create all of the elements of R(0,1).

Thus in the rows of the CUT array, N can simultaneously be used to create R(0,1) and be the row numbers of the CUT array. In essence, the algorithm steps simultaneously are used to create R(0,1) and map to the rows of the CUT array.

By the CD the cardinality of the set created by the CA cannot exceed the cardinality of N, the set that creates it. Thus there is no concern that there exist rows of the array after all of the elements of R(0,1) are created by N or Set Nv. In other words, there is no concern that there are rows of the array after N is exhausted.

We are now in a position where we can "create" the set within the CUT array.

We will use the Diagonal Serpentine Mapping (DSM) with the row numbers (which are now N) and the column numbers to map N onto the cells of the array. However, instead of assuming N maps to the elements in the cells of the array, we will use the CD and CA and use these elements of N to **create** a set that is countable by the CD and CA.

In other words, the DSM mapping, which uses N, is actually using N with the CD and CA to create a set derived from this N. By the CD this newly created set is countable. This makes the CUT valid. **QED**

Kehr's Second Extension of Cantor's CUT (KSE): "Given any set that can be shown to be countable by KFE, the power set of this set is countable."

Proof:

(Note: This proof would work on any set that is countable by the CD and CA, however, one of the Turing Machine proofs below will use a CUT array so the theorem is designed for CUT arrays.)

Let us begin by modifying what we did in the proof of KFE. We will list the elements of N that are used by the DSM side-by-side with whatever set it is creating within the array. We are essentially "straightening-out" the DSM. We will use the same column numbers as above.

The creation of the set in column #1 (such as R(0,1)) is an infinite process which does not have an "end." The important thing to remember is that it is driven by the N of the DSM. In other words, nothing happens after all of the elements of N are executed as step numbers.

For example, in the Pipe Mapping, the elements of N are exhausted before all of the elements of R(0,1) are mapped to (i.e. certain nonterminating decimals are not mapped to). But with the CA, the elements of N are totally sufficient to completely construct all of the elements of R(0,1) (note: R(0,1) is being used as a sample of the use of the CA). No elements of R(0,1) are left to be created after N is exhausted.

This same fact is true for the N that creates the set resulting from the CUT array with KFE, call it Set CUT. The elements of the N used by the DSM are totally sufficient to completely construct all of the elements of Set CUT.

The key question is this, can the cardinality of the power set of Set CUT be greater than the power set of the N that is used to create Set CUT?

It is obvious that the power set of the N that is used to create Set CUT is countable, because there is only one definition of N (by the ACN), thus the power set of one N is the same as the power set of any other N.

Thus the only way that the power set of Set CUT can be uncountable is if the power set of Set CUT is greater than the power set of the N that is used to create it.

By the CD, the cardinality of Set CUT cannot exceed the cardinality of the N that is used to create Set CUT. If the cardinality of Set CUT cannot exceed the cardinality of this N, then clearly the cardinality of the power set of Set CUT cannot exceed the cardinality of the power set of this N (i.e. by applying the

power set operation to both sides of the "Card(Set CUT) <= Card(N)" formula), which is countable. Thus the power set of Set CUT cannot be uncountable. **QED**

Turing Machines:

In my discussion of Turing Machines, I will use the terminology of the book: **Computability and Logic**, Third Edition, by Boolos and Jeffrey (Cambridge University Press).

A Turing Machine is represented by a machine or box that sits on top of a tape that has an infinite number of squares in both directions. From a cardinality standpoint, the number of symbols that are available to be put onto each square is not important, as long as it is finite. Thus, suppose each square is either blank (i.e. S0) or contains a '1' (i.e. S1). Before the Turing Machine program is executed, each square of the tape is either blank, or it has a '1' written on it.

The Turing Machine program, when viewed as a table, consists of n rows, that are called "states" or "instructions" or "rows" and two columns (in this case). This means there are 2*n or 2n cells in the table. For each cell of the table there are five possible "acts" of the machine or box that is on top of one square of the tape:

1) Halt the computation (i.e. a blank cell in the table)
2) Move one square to the right
3) Move one square to the left
4) Write S0 (i.e. blank) in place of what is in the square
5) Write S1 in place of what is in the square

(Note: Of course if the current symbol in the square is an S1, for example, and the instruction says to write an S1 on the square and go to the same state number, this would immediately put the Turing Machine in an infinite loop. I suspect this possibility would be frowned upon, but to be thorough I will consider such an option to be legal.)

It is well known that for a two-symbol (S0 and S1) Turing Machine with n possible states of the reading head, there are exactly (4n + 4)^2n distinct programs that can be written.

Suppose a third party observes the actions of the machine or box, but has no clue what the Turing Machine table looks like. In other words, the third party can see the machine move, write and halt, but has no clue what state the machine is in.

If the third party observed every possible program, of the (4n + 4)^2n distinct programs, acting on every possible initial configuration (i.e. initial tape

configuration); he or she might observe that, for every initial configuration, two or more Turing Machine programs behave exactly the same and thus result in the same ending or resulting tape configuration. Thus, in some instances this third party would not be aware that two distinct programs were used.

If we look only at the functionality of the Turing Machines, meaning only the initial configuration, the actions of the machine or box, and the resulting tape configurations, then the formula of (4n + 4)^2n may contain some redundancy and thus be too large. This will be better understood in a moment.

An "act" is an overt or physical "act" of the machine or box. Physical "acts" include only the writing to a square, the moving to another square or halting. These are the physical "acts." The mental part of the program is keeping track of which state is being executed and which column to use. The third party does not "see" the mental parts of the program.

The "cell" of the Turing Machine table that is executed is determined by the row (i.e. the state) and the column (i.e. what symbol is already on the square). For example, if the machine sees an "S0" on the square, the table cell may tell it to move one square to the right and go to state 58. The actual physical "act" in this sequence is to move one square to the right. Deciding on which row and column to execute and on which state to execute next is a mental or program issue, not a physical "act" that the third party can see.

The Turing Machine "halts" when it runs out of instructions, usually meaning it hits a blank cell. If the Turing Machine halts, it must halt after a finite number of "acts." If the Turing Machine never halts, then it performs an infinite number of "acts." This can be caused by an infinite loop or an infinite number of states.

A "Turing Machine" is defined to be the table of states, independent of what is initially on the tape. In other words, a "Turing Machine" is a table of instructions, or "program," that can act on any initial configuration of the tape.

How Many Initial Configurations Can the Tape Have?

Before determining the number of possible Turing Machines (programs) let us first determine how many possible initial configurations there are for the tape.

Let us number the squares on the tape. We will number the initial or beginning square as '1' (i.e. the square where the Turing Machine sits at the beginning). We will number the squares to the right of the initial square consecutively with "even" counting numbers: {2,4,6,8,10,...}. We will number the squares to the left of the initial square consecutively with "odd" counting numbers, beginning with 3: {3,5,7,9,11,...}.

We can therefore represent any initial configuration as a subset of N. For example, suppose the squares that are non-blank (i.e. the squares that contain a '1') are squares: {5, 10, 15, 20, 25}. This list of non-blank squares is a subset of N.

Whether the tape is allowed to contain a finite or a countable number of '1's doesn't matter; each unique initial configuration of '1's can be represented by a finite or an infinite subset of N.

By using a Delineation Mapping, where each initial configuration is considered an element of the power set of N, it has already been shown that the number of possible initial configurations is countable. In this case redundancy is not allowed (i.e. an initial configuration cannot list the same square twice) and the elements are assumed to be in order by size (e.g. 2 is listed before 8 and 8 is listed before 12).

An "initial (tape) configuration" will always have the starting position of the Turing Machine program as its number "1" square. In this way we have a clear reference point for numbering the squares on the tape and thus what constitutes a "unique" initial configuration.

(Note: Using a delineation mapping it is also possible to prove that there are a countable number of resulting (tape) configurations. This should be noted for a comment later in this chapter.)

The "Pure Listing of Acts" (PLOA):

Given an initial configuration of the tape, and given a Turing Machine table or program, a given listing of physical "acts" that are performed by the machine or box can be isolated.

In other words, once the initial configuration of the tape is shown to us, and the program is shown to us, we could list all of the physical "acts" of the machine or box that occur in the order that they occur. If a tape with a different initial configuration is used with the program, then it is quite possible that a different list of consecutive, sequential "acts" of the machine or box would be generated.

For example, suppose that we are given an initial configuration of the tape and a Turing Machine table such that the following events occurred:

Simplified Diary of an Initial Configuration/Turing Machine Combination:

1) Execute state 1 (mental look at row 1),
2) An S1 is in this square (mental look at the current square)
3) Look at row 1, column S1 (mental look at row 1/column S1),
4) Write S0 on the square (physical act based on cell instructions),
5) Go To state 1 (mental look at row 1),
6) An S0 is in this square (mental look at the current square)
7) Look at row 1/column S0 (mental look at row 1/column S0),
8) Move one square to the right (physical act based on cell instructions),
9) Go To state 58 (mental look at row 58),
10) An S0 is in this square (mental look at the current square)
11) Look at row 58/column S0 (mental look at row 58/column S0),
12) Write S1 on the square (physical act based on cell instructions),
and so on.

Note that in the above list of 12 items, 9 of them are "mental look at" items the third party cannot see, and three of them are "physical acts" performed by the machine or box that the third party can see. All of the "look at" items are performed "mentally" by the program, but the "acts" are physically performed by the machine itself.

We could isolate and list the "acts" performed by the machine in this way:

A Pure Listing of (sequential and consecutive) Acts (performed by the machine):

1) Write S0 on the square (item #4 above),
2) Move one square to the right (item #8 above),
3) Write S1 on the square (item #12 above),
and so on.

These are the items that the third party can see. Note that the "Pure Listing of Acts" (PLOA) list above is a function of the original configuration of the tape and the Turing Machine program or table. If either the original configuration of the tape changes or the Turing Machine table changes the "Pure Listing of Acts" may change.

How Many Possible PLOAs are there?

Given that each square can only contain a '0' or a '1', each physical "act" of a PLOA must be one of the five "acts" mentioned above:

1) Halt the computation
2) Move one square to the right
3) Move one square to the left
4) Write S0 (i.e. blank) in place of what is in the square
5) Write S1 in place of what is in the square

Note that the PLOA just listed above contains three "acts," each of which is equal to one of these five possible "acts" of the machine or box.

Now let us consider Set Nv in base 5 (i.e. each digit must be a 0,1,2,3 or 4). We can easily create a base 5 tree for Set Nv. The nth level of the Set Nv base 5 tree has 5^n branches.

Let us now consider a similar tree for the above 5 "acts" of a machine. Let '0' be the "halt," '1' be "move one square to the right," and so on. This makes the Turing Machine tree comparable to the Set Nv tree.

The set of all possible PLOAs with one act is 5^1. However, the set of all possible PLOAs with exactly two acts is less than 5^2. It is less than 5^2 because if the first "act" is a halt (i.e. a zero), there are no second "acts" attached to this branch. This is a little different than Set Nv because with Set Nv a number such as v30212033302 is allowed. But with a Turing Machine the first "zero" (i.e. the "halt") halts the machine. There cannot be two "zeros" or "halts" in a PLOA.

Clearly the Set Nv base 5 tree and the tree that can be created from the set of 5 possible "acts" (the "base 5 PLOA tree") are very similar trees.

Clearly, for all finite PLOAs, as with Set Nv itself, the set of all of these acts is countable because Set Nv is countable.

But what about PLOAs that have an infinite number of acts? By using the Creation Algorithm we can clearly create all of these sets within a countable number of algorithm steps. This set is clearly similar to R(0,1) in quasi-base 4 because there is no "halt" symbol in these sets. Each digit of an ISD in quasi-base 4 and each of the non-halt acts above are significant.

We now have shown that the set of all of the sets of acts that halt is in bijection with a subset of Set Nv and the set of all of the sets of acts that do not halt is in bijection with R(0,1) in quasi-base 4. By KFE (instead of the CUT), therefore, the set of all possible PLOAs is countable (i.e. the union of two countable sets is countable).

Just as the base of R(0,1) does not affect its cardinality, as long as the base is finite (actually the base - or number of symbols - could be countable, but I have

not taken the time to prove that); similarly as long as the number of available symbols is finite, the number of possible PLOAs is countable.

A Functional Definition of a Turing Machine:

Turing Machines are typically defined by a unique table of states. Given any specific initial configuration of the tape, and given the table, a specific ending configuration of the tape will result.

What if we are less concerned with the "Go To" commands in a Turing Machine table and instead focus on the functionality of the machine or box? In other words, we are interested in how the machine or box rearranges the initial configuration of tape symbols rather than all of the possible logic jumps the program goes through.

For example, let us take every possible initial configuration of the tape and a given Turing Machine program. For each possible initial configuration let us calculate the PLOA based on this single Turing Machine. For this one Turing Machine we essentially have a table which, as its first column, contains all of the possible initial configurations, and in the second column contains the PLOA generated by that combination of initial configuration and Turing Machine program.

If we do this for every possible Turing Machine, and make sure that all initial configurations are in exactly the same order, then for every possible Turing Machine program we can generate a two-column table where column #1 contains every possible initial configuration and column #2 contains the calculated PLOAs for each initial configuration/Turing Machine combination.

Thus, we have a new and different way of representing each Turing Machine table, which I will call the: "PLOA Table." The PLOA Table is essentially a table with an infinite number of rows and two columns. A PLOA Table can be created for each and every Turing Machine program.

Whether we count the number of unique Turing Machine programs or whether we count the number of unique PLOA Tables is irrelevant, they are the same thing.

However, before going further, it is necessary to talk about redundancy.

Redundancy of PLOA Tables:

Given any unique initial configuration of squares, and given any unique pure listing of acts (i.e. given a unique PLOA Table), suppose we add a column #3 to

represent the resulting tape configuration. Given two PLOA Tables, If columns #1 and #2 are identical for both tables, then column #3 will be identical for both tables.

What if more than one Turing Machine program generates identical PLOA Tables? To say this another way: suppose that given the same initial configurations, two different Turing Machine programs result in the same PLOA Table and thus the same resulting configuration of the tape (column #3)? What would be accomplished by counting both of these PLOA Tables?

To understand why nothing would be accomplished, consider this: if I wrote a 1 megabyte computer program to generate an econometric model, what would be accomplished if someone else wrote a 1 gigabyte computer program that resulted in the exact same model? There is no benefit by writing two programs that do the same thing. Our interest is in the number of functional Turing Machines, not in how many Turing Machines we can create that do the same thing.

In fact, multiple Turing Machine programs can generate the same PLOA Table. Looking at Turing Machines in a purely functional way, counting the number of possible PLOA Tables is actually a more efficient way of counting Turing Machines because functional redundancy is ignored.

We could even take this a step further (but I won't) and consider a Turing Machine to be a unique table of initial configurations (column #1) and ending configurations (column #2). One such table could be created for each Turing Machine. Obviously there would be redundancy in this case also.

In summary there are three ways to count Turing Machines:

1) count each unique Turing Machine table,
2) count each unique PLOA Table, and
3) count each unique combination of initial configuration/ending configuration.

In the long proof that there are a countable number of Turing Machines I will count the second item in this list. In the short proof that there are a countable number of Turing Machines I will count the first item in this list. I will also trivially deal with the third item after the long proof.

The Long Proof there are a Countable Number of Turing Machines:

Theorem: There are a countable number of unique PLOA Tables

Proof:

Let us consider a CUT array where the rows of the array are every possible initial tape configuration and the columns are every possible PLOA.

It has been shown that the initial tape configurations can map into R(0,1) by the delineation mapping. Thus these configurations can be used as the rows of a CUT array.

It has also been shown that the PLOAs with an infinite number of "acts" are similar to a CA for R(0,1) in quasi-base 4. It has also been shown that the PLOAs that halt can map into Set Nv in base 5. Thus it is obvious that this set can map into R(0,1) in base 5 (this is not quasi-base 5). Thus this set can be used as the column numbers of a CUT array.

Thus both KFE and KSE can be used on this CUT array.

Noting that each cell of this CUT array is a combination of an initial tape configuration and a PLOA, let us consider the power set of this CUT array. By KSE the power set of this CUT array is countable. Thus the power set of the set of all possible "initial configuration/PLOA cells" is countable.

However, every unique PLOA Table is clearly an element of the power set of this CUT array because a PLOA Table is nothing but a set of "initial configuration/PLOA cells." Thus since the power set is countable, a proper subset of the power set (i.e. the set of all possible PLOA Tables) must also be countable. QED

This proof is interesting in that it does not use the Axiom of Choice because it does not mention or care that each PLOA consists of one cell per row. However, if we were to actually construct a PLOA from the CUT array, it would require the AC.

That was the long proof. By using similar logic it is quite trivial to prove that if a Turing Machine is defined only by its unique initial and resulting tape configurations (item #3 above), that this definition would also lead to a countable number of possible Turing Machines.

The Short Proof there are a Countable Number of Turing Machines:

Let "N/o" be the symbol for "aleph nough," the cardinal number of N.

Let "N/i" be the symbol for "aleph one" or simply "aleph," the cardinal number of R.

It has been shown in previous chapters that N/o and N/i are the same cardinal number.

As was mentioned above, given all two-symbol Turing Machines with n possible states, there are exactly (4n + 4)^2n distinct programs. Functionally, some of these are redundant, but that doesn't matter.

Obviously, the set of all Turing Machines with a finite number of states is countable by the "Into-Definition."

With KSE in mind, the question arises about Turing Machines with an infinite number of states. The rest of this discussion uses standard Transfinite Set Theory mathematics (e.g. see Set Theory and Logic, by Robert R. Stoll).

Let us substitute N/o for n in the above formula. The formula becomes:

(4*N/o + 4)^(2*N/o)

The term (4*N/o+4) reduces to N/o. The term (2*N/o) reduces to N/o.

The cardinal number of the set of all possible Turing Machine programs thus reduces to the term: N/o^N/o

Now consider these formulas:

N/i = 2^N/o (Creation Algorithm in quasi-base 2 - this is a well established formula)

N/i^2 = (2^No)^2 (by substitution) = 2^(2*N/o) = 2^(N/o) = N/i

N/i^N/o = (2^N/o)^N/o (by substitution) = 2^(N/o^2) = 2^N/o (by the CUT) = N/i

N/i = 2^N/o <= N/o^N/o <= N/i^N/o = N/i (see above), and hence N/o^N/o = N/i

which has been shown to be countable.

Thus the set of all finite state Turing Machines is countable and the set of all infinite state Turing Machines is also countable. Thus by KFE the set of all possible Turing Machines is countable. QED

End of Chapter

Webster Kehr: webster.r.kehr@mail.sprint.com