Solution to the Halting Problem

MELAMPUS

Abstract

The Halting Problem is solved by complete induction on the number of states in any Turing Machine. It is inferred that mathematical induction is a species of synthetic reasoning.

Black's Academy Limited
Kington, England

Contents

Appendices

1. Introduction

Solution by mathematical induction is intuitively plausible

A machine of *n* states is designated T_n. *Figure 1* illustrates the solution for a five-state Turing machine. In the figure, we consider the four-state T_4 machine to have been made by the addition of one state to a three-state machine, T_3. This is illustrative of the iterative process that is involved in a proof by mathematical induction. Halting behaviour is a consequence of a *tape configuration*. Let $Q_0, Q_1, \ldots, Q_n$ denote states. Let S_k denote the input of a tape configuration at state Q_k. For example, by $S_1 = 000_1 10$ we denote an input to a machine; the subscript shows where on the tape the machine in state Q_1 is scanning the symbol 0; the whole input is a tape configuration. Suppose the instruction for Q_1 is $0:1$ then go to Q_2; then the output, which is a function of the program, the state and the tape configuration, is $001_2 10$; the machine has moved to state Q_2 An exit, denoted X_i, is any output that causes the machine to halt.

A *complete criterion* for an *n*-state machine is a specification of those tape configurations for any inputs to the machine that cause that machine to halt, specifying the exit at which it halts, and giving completely the tape configuration at any given exit. It is intuitive that a complete criterion can be written. (1) Finite and infinite strings of symbols represent finite information that can be expressed by finite symbols. (2) The number of permutations of steps through an *n* state machine is finite. Any loop can involve at most *n* instructions. The tape configurations that lead to any given exit are determined by finite symbols. (3) No loop is random. A machine that does not halt has entered a loop in which the same tape configuration will inevitably appear, even if the number of steps between states is large. There is no infinitely non-recurring sequence of (random) tape configurations in any Turing machine. The complete criterion for any machine may be written by a backward trace from any given exit. This is here called *the method of exits*. If the complete criterion is given for an *n*-state machine, then by the method of exits, a complete criterion may be determined for an $n+1$ machine. There is also a *method of inputs*, equivalent to the method of exits. The method of inputs is used in the proof of the induction step.

There is a standard presentation of the Halting Problem in Appendix 1. For readers not familiar with the theory of Turing Machines there is a summary in Appendix 2.

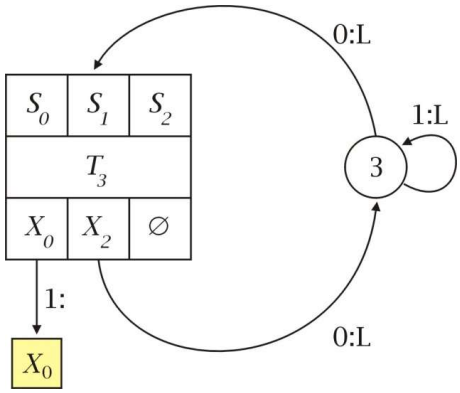


Figure 1

Diagram illustrating the iterative procedure needed for the inductive proof of the Complete Criterion Theorem.

Philosophical and cultural considerations

In would seem like insanity to attempt the proof of a problem that has already been proven to be impossible to solve. But an impossibility proof, like any other proof, has assumptions, and if the assumptions are false, then the conclusion does not follow.

The impossibility proof for the Halting Problem is a proof that within a certain mathematical system, one that is computable, there cannot be a solution to the Halting Problem. Hence, the impossibility proof is prima facie a proof that a computer cannot solve the Halting Problem and no more. It does not directly relate to the human capacity to solve problems, and only can be made to so relate if one adopts the premise that human beings can only solve at most all those, and only those problems that computers can solve.

Therefore, it may not be assumed without circularity that human reason is only capable of solving at most all those problems that a computer can solve. It may also not be assumed without circularity that mathematical induction is a species of computable algorithm.

In solving the Halting Problem by mathematical induction we not only demonstrate that the human mind can solve a problem that a computer cannot solve, we demonstrate that mathematical induction is not a computable algorithm. Therefore it cannot be assumed without circularity that a solution by mathematical induction is impossible. The impossibility proof is only a proof that a computer cannot solve the Halting Problem: it makes no reference whatsoever to mathematical induction. It does not circumscribe or limit human reasoning.

But that there could be such a solution to the Halting Problem must surely produce a cultural shock: *why in the history of the problem has such a simple solution never been considered?* The answer will be due to cultural forces that made it seem insanity to even consider such a solution.

Mathematical Induction and algorithms

Mathematical induction, also called complete induction, is an argument that proceeds from two premises to establish a property concerning all natural numbers. One of the most elementary results established by mathematical induction is the formula for the summation of the series: $1+2+3+...+n=\frac{1}{2}n(n+1)$. This algebraic formula is clearly algorithmic and it tells us how to compute the sum of consecutive numbers by means of a short-cut: the long summation on the left is converted to a simple multiplication on the right. The inductive argument establishes that we *know* that this algorithm correctly computes the summation up to any number *n*. Thus, prima facie, mathematical induction is a device for establishing *truth* and *knowledge* and is not an algorithm in itself. The knowledge that the formula is correct is established in the inferential movement from the first two steps in the argument to the conclusion.

Now let us consider the situation that arises in the proof that of the impossibility of a solution to the Halting Problem. What this proof of impossibility establishes is a relationship between two Turing machines. It does not state that the Halting Problem is insoluble for all machines, or even any particular machine, but merely establishes that there could be no one machine that could solve all Halting Problems, including its own. It also establishes that if Turing machine *A* provides a solution to the halting problem of machine *B*, then *A* must have more states than machine *B*. Thus, another formulation of the conclusion is that any machine that could solve the halting problems for all machines would have actually infinite states. Of course, that would not make it a machine, since all machines have finite states. But is this situation really so different for algorithms in general? The expression $1+2+3+...+n=\frac{1}{2}n(n+1)$ is true for all natural numbers, but it does not follow that there ever has been built a machine that actually has computed the sum of all natural numbers, and given that the collection of all natural numbers is infinite, any machine that applied this formula to all of them would also require infinite states. The impossibility proof for the Halting Problem (Appendix 1) makes only a common situation more explicit in the case of the Halting Problem, because it constructs a relationship between two machines, and says any machine *A* that could solve the problem for machine *B* is "bigger" than machine *B* – that is, has more states.

Mathematical Induction

Mathematical Induction is the argument used in Number Theory and throughout mathematics that has the following form.

Particular Result: the result is true for $k=0$ (or for some other starting value).

Induction Step: if the result is true for the number *k* then the result is true for $k+1$.

Conclusion: the result is true for all numbers *n* (or for all *n* greater than the starting value).

In symbols:

$$\frac{P(0)\qquad P(k)\rightarrow P(k+1)}{(\forall n)P(n)}$$

The impossibility of a solution the Halting Problem

A standard version of the impossibility proof to the Halting Problem is given in Appendix 1.

The Church-Turing Thesis

The standard result that all computable algorithms are Turing machines is given in Appendix 2.

Mathematical induction is a species of inference that establishes *truth* and *knowledge*. An algorithm is a mechanical procedure that on occasion is used to apply the fruits of such inferences to compute results. No computer has been constructed that could churn out algorithmically all the results that could be established by mathematical induction, and only the existence of such a machine would suffice to prove that mathematical induction was itself an algorithm. That some inductions may be simulated by computers and even some new results established by such simulations does not prove that induction itself is algorithmic. Simulation may be nothing much more than a species of typing.

But these remarks are only intended to demonstrate that as a *possibility* mathematical induction is not algorithmic. It would be as circular to assume this result as it would be to assume its opposite, because it is conceivable that there is a mechanical and material substratum to the human mind at which level all operations of the mind are algorithms. However, if there is an inductive solution to the Halting Problem, then such a result would rule out that consideration.

2. The method of exits

A Turing tape is a linear tape, marked into squares, which is potentially infinite in both directions. Each square of the tape contains either the symbol 0 or 1.

Here we have been explicit in defining the tape to be *potentially* infinite. This is in contrast to the description of the tape offered by Boolos and Jeffrey, which refers to an "actually infinite" tape.

A tape configuration (or "configuration" for short) is an assignment of the symbols 0 and 1 to a portion of the Turing tape. (We shall also identify in the tape configuration the state the Turing machine is in, and which symbol it is scanning.) Since the tape is potentially infinite, we must have devices for describing potentially infinite portions of the Turing tape.

The expression $\overline{\overline{0}} = ...000...$ is an assignment of 0 to the whole potentially infinite tape: in it, every square of the tape both to the left and right contains the symbol 0. The double over-bar represents an assignment to a potentially infinite portion of the tape. We allow that the meaning of a symbol is unambiguously modified by context. Thus, the configuration $\overline{\overline{0}}1010\overline{\overline{0}}$ indicates that a portion of the tape contains the symbols 101 and that to both the left and right of that configuration the tape is assigned a potentially infinite sequence of 0s. The expression $\bar{0}$ is an assignment of 0 to a finite but indeterminate portion of the Turing tape. It is a finite string of 0s on the tape, but the actual number of 0s is not known. It may be zero.

When an explicit configuration is given, for example, 010110, it is presumed that the assignment to the left and right of this configuration is undetermined: that is, *we do not know* what the symbols to the left and right of this portion of the tape configuration are. Thus, an assignment is a determination of an otherwise undetermined tape. The whole tape may be determined or undetermined. For example, the configuration $\overline{\overline{0}}1010\overline{\overline{0}}$ determines the whole tape. It states that there is a portion of the tape definitely containing the configuration 101, and that to both the left and right of this portion there is a potentially infinite sequence of 0s. The Halting Problem is solved by determining what configurations on an otherwise undetermined tape cause a given Turing machine to halt at a given exit.

The potential and actual infinite

The potential infinite: no matter how large a (natural) number we have reached it is always possible to count to a higher one by adding one more. Counting is inexhaustible.

The actual infinite: the entire process of counting forms a completed totality. A completed collection of infinite objects is given actually in its entirety.

Of these two concepts it is that of the actual infinite that is problematic. The potential infinite is implicit in direct experience, but the actual infinite is a construct of metaphysics or (mathematical) science required to define the idea of a limit terminating in a real number. In the history of these concepts, Aristotle legislated against the use of the actual infinite, though he accepted the potential infinite, and the consequence was that Greek science never developed the calculus. In mathematics, the potential infinite is primarily a concept of number theory, and the actual infinite is primarily a concept of analysis.

From Boolos and Jeffrey

"We suppose that the computation takes place on a tape, marked into squares, which is unending in both directions – either because it is actually infinite or because there is a man stationed at each end to add extra blank squares as needed." (Boolos and Jeffrey [1980] pp.20 – 21.)

The gloss after the introduction of the actual infinite here confirms that it is only the potential to add one more square to the tape that is required for the Turing tape.

The expression $\bar{0}101\bar{\bar{0}}$ indicates that the leftmost part of the Turing tape is undetermined; there then follows a finite but unspecified number of 0s; this is followed by a sequence of symbols 101; finally the rightmost part of the tape is completely determined by a potentially infinite sequence of 0s. The expression $\bar{0}$ encompasses the possibility that $n=0$, and that there are in fact no 0s on that part of the tape to which this symbol is assigned. The expression $0\bar{0}$ indicates a finite sequence of *n* 0s, but one where there is at least one 0; that is, $n\geq 1$. The expression $\overline{00}$ indicates an undetermined, finite, even sequence of 0s on a portion of the tape; the expression $0\overline{00}$ indicates an undetermined, finite, odd succession of 0s. These symbols are required to describe the action of instructions of given states, as will be made clear below in the examples. The expression $\overline{001}=(001)_n$, $n\geq 0$ indicates the finite but indeterminate repetition of the sequence 001 on a part of the Turing tape. The expression $\left\{\begin{matrix}\overline{01}\\ \overline{111}\end{matrix}\right\}$ indicates a finite but indeterminate repetition of the sequences 01 or 111 in any (random) order.

By *standard configuration* is meant a situation in which the machine either starts scanning a zero on an otherwise blank tape or ends scanning the leftmost of a block of 1s on an otherwise blank tape. The *Halting Problem* is the problem of designing an effective procedure for identifying Turing machines which never halt, once started in their lowest-numbered states on blank tapes. In our notation the starting standard configuration is expressed by $\bar{\bar{0}}_0$, and it is this that is referred to as "starting in standard configuration". The Halting Problem cannot be solved in isolation from a larger problem, that of what happens when a Turing machine is started for *any* given configuration in any of its numbered states. We consider a potentially infinite number of *non-standard* starting configurations. Any such non-standard starting configuration may be considered as an *input* to a state scanning a given symbol of a given tape configuration. We designate the states of an *n*-state Turing machine by $Q_0, Q_1, \ldots, Q_{n-1}$, and inputs at each of these by $S_0, S_1, \ldots, S_{n-1}$. Thus, the input of the standard configuration to state Q_0 is designated: $S_0=\bar{\bar{0}}_0$. An example of a non-standard input is: $S_1=\bar{0}10_1 1$; this indicates that while in state 1 the machine is scanning a 0, immediately to the left of this is a 1, then a finite but indeterminate string of 0s; beyond that, the tape is undetermined – we do not know what it contains.

Repeating tape configurations

As indicated in the text, the expression $\left\{\begin{matrix}\overline{01}\\ \overline{111}\end{matrix}\right\}$ indicates a finite but indeterminate repetition of the sequences 01 or 111 in any (random) order. In this, every possible permutation being allowed. Thus, this expression has many instances. Examples are:

01 111 01 111	01 111 111 01
111 01 111 01	111 111 111 01

Here the gaps between the different blocks are introduced solely for the sake of clarifying the meaning. These expressions are needed to describe the effect of loop constructions in a Turing machine upon a tape configuration. There are manipulations of these symbols that are direct consequences of their definitions. For example, $100\overline{00}0=10\overline{00}0=1\overline{00}00$.

A Turing machine is specified by a collection of instructions, where every instruction comprises an ordered quadruple: (1) the number of the state it is in; (2) the number of the state it next moves to; (3) the symbol that is being scanned; (4) the action to be taken: L, R, $0, 1$ (Move left, move right, write a 0, write a 1). Since there are only two symbols used on the tape, to every state there is assigned at most two instructions. If there is no instruction assigned for a state scanning a given symbol, then the machine halts. An *exit*, denoted X_i, is any output that causes the machine to halt. Flow graphs, which are diagrammatic representation of Turing machines and their instructions, are vital heuristics, and make everything clear.

A *trace* is any record of the step-by-step effect of a given input. The input of systematically generated permutations of the tape to generated traces is called the *method of inputs*.

Complete criterion

A *complete criterion* for an *n*-state Turing machine is a specification of those tape configurations for any inputs that cause that machine to halt, specifying the exit at which it halts, and giving completely the tape configuration at any given exit.

Complete Criterion Theorem

For any finite Turing machine of *n* states, it is possible to write a complete criterion. The solution to the Halting Problem is an immediate corollary of this theorem. The theorem is proven in section 5. When we do so, we will show that complete criteria can always be written by the method of inputs, but in practice the most efficient method for determining a complete criterion is given by the *method of exits*.

The method of exits

This involves tracing backwards from a given exit by using an *if-what* procedure: if the machine exited and halted at such-and-such a state, then what tape configurations at given inputs could have led to this outcome? On an *if-what* basis, we determine at each step all the possible tape configurations that could have led ultimately to that exit. The method of exits generates a structure known as a tree. At each stage (or *level*) of the tree there is the possibility of branching. Since a machine is finite, no state can have infinite inputs from other states.

Flow graphs. Flow graphs are diagrams of Turing machines introduced by Boolos and Jeffreys [1980].

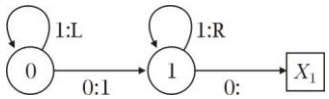


Figure 2. Example of a flow graph

Example of a trace. The trace for input $S_0 = 0011_0 0$ is:

$S_0 = 0011_0 0$ $001_0 10$
$00_0 110$
$01_1 110$
$011_1 10$
$0111_1 0$ $X_1 = 01110_1$

Method of exits. The following is a backwards trace for the machine whose flow graph is given in figure 2.

Exit at X_1
$X_1 = 0_1$
$1_1\bar{1}0$ 0:1
$0_0\bar{1}0$
$01_0\bar{1}0$ $0\bar{1}1_0\bar{1}0$ $0\bar{1}\bar{1}_0 0$
END END END

On an *if-what* basis, we determine at each step all the possible tape configurations that could have led ultimately to that exit. Since there are two instructions leading into state Q_1 we must consider that *either* might have led to the outcome. In the first instance since we have $X_i = 0_1$ then we could not have entered this state and configuration by having immediately read the instruction $0:1$, so this is an impossibility in the backward trace and we show that CONTRADICTION by marking that option $0:1$ in red. The expression $1_1\bar{1}0$ is a standard way of representing the action of the looped $1:R$ instruction for Q_1. In order to have reached the exit, the machine may have started scanning the leftmost 1 of a block of 1s; the tape to the left being undetermined. In the backward trace, the only next possibility is the instruction $0:1$; the trace takes us back to $0_1\bar{1}0$. Finally, we may have reached this state from the action of the instruction $1:L$, but in considering this there are three possibilities, as represented by $01_0\bar{1}0$, $0\bar{1}1_0\bar{1}0$, $0\bar{1}\bar{1}_0 0$ because the initial scanned 1 may appear anywhere within a finite block of 1s. From this point the backward trace cannot continue, and this is marked in red by END.

To prove that the Halting Problem is soluble we have to demonstrate (a) that the *length* of each branch of the tree is finite – that is, each branch terminates either in an END, a LOOP or a CONTRADICTION; (b) the *width* of the tree is always finite – the tree only ever has finite branches. The term LOOP should already be intuitively clear. In a computer program we often write loop instructions; for example, *while ... , do ...* . The idea here is to infer from the repetition of a tape configuration that a loop has been entered.

The result of the application of the method of exits to any machine enables us to write its complete criterion. The complete criterion records all the possible halting configurations; from it we may infer all the non-halting configurations. Since any machine either does or does not halt, the set of all configurations of the Turing Tape is partitioned by the machine into two sets: the set of configurations that halt, and the set of configurations that do not halt. Thus, if a configuration is not a halting one, then it is non-halting.

Figure 3 is a diagrammatic representation of the complete criterion for any machine of *n* states. This representation treats a Turing machine in a manner akin to the "black box" principle in computer science: a Turing machine is seen herein as merely a device for computing outputs (here exits) from given inputs – a machine whose "internal workings" from this point-of-view are otherwise irrelevant. The internal workings, which are by no means irrelevant, are fully given in the description of the internal states of the Turing machine, which are represented diagrammatically in the flow graph. In the diagram the symbol $\varnothing$ signifies the infinite non-halting LOOP configurations of the Turing machine.

The solution to any halting problem is straightforward in theory – though in practice, the computation could be "hard". If in fact every tree generated by the method of exits is finite – that is, if in fact all its branches terminate – then the complete criterion for any Turing machine may be written, and this expresses everything determinate about the machine, and everything that could be needed to be known about it. Then: if, and only if, the standard starting configuration $S_0 = \overline{\overline{0}}_0$ appears somewhere in the tree of halting configurations does the machine halt for $S_0 = \overline{\overline{0}}_0$. Only those machines that have no exits at all never halt; all other machines have halting configurations, and among these the configuration $S_0 = \overline{\overline{0}}_0$ may sometimes appear, and our task is merely to determine the complete criterion to find out whether that is so.

The Complete Criterion

The result of the application of the method of exits to the machine of *figure 2* enables us to write its complete criterion.

S_0		S_1	
$0_0\bar{1}0$	$H = 1\bar{1}\boxed{0_1}$	$1_1\bar{1}0$	$H = 1\bar{1}\boxed{0_1}$
$01_0\bar{1}0$	$H = 1\bar{1}\boxed{0_1}$		
$0\bar{1}1_0\bar{1}0$	$H = 1\bar{1}\boxed{0_1}$		
$0\bar{1}1_0 0$	$H = 1\bar{1}\boxed{0_1}$		

This records all the possible halting configurations; from it we may infer all the non-halting configurations. In this complete criterion, we also occasionally find it helpful to mark the symbol scanned at the exit (halting configuration) by putting it into a box; but this is a presentational device only. If a configuration is not a halting one, then it is non-halting. The non-halting configurations for this machine are $\overline{\overline{1}}_0$ and $\overline{\overline{1}}_1$.

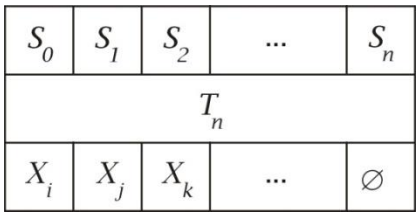


Figure 3. The Complete Criterion

3. Solving unsolved five-state hold-outs

According to the papers published by Kellett [2005] and Ross et.al [2006], after certain standardisations have been performed to ensure duplicate and mirror machines have been only counted once, there are 7,491,189 Turing machines with 5-states – an impressively large number of machines. (There are approximately 7.6 billion 6-state machines.) After running certain screening procedures executed by algorithms run on powerful computers, Kellett [2005] and Ross et al. [2006] claim to have solved the Halting Problem for all but 98 Turing machines, which they designate "hold-outs".

It is a claim here that by the methods established in the preceding section the Halting Problem for all 98 "hold-outs" is solved, and that in fact all 98 are non-halting. Some of these problems may be classified as "hard" and require hours of patient work, but most can be solved by the method of exits in two stages in approximately 20 to 25 minutes. The most difficult problem encountered by this author was in fact to verify that the Kellett "champion" five-state machine did in fact halt for starting standard configuration. This champion machine computes the productivity of the best five-state machine to be 11; that is, it halts in standard configuration scanning the left-most of eleven 1 symbols. If the 98 "hold-outs" are indeed non-halting, this confirms that the productivity of a five-state machine is 11.

We now proceed to illustrate (a) the method of exits, (b) the iterative procedure used to establish the general solution to the Halting problem for machines of any size whatsoever, through a series of annotated examples.

Example 1

Holdout machine no. 60 (also designated Kellett [2005] B.9).

It is straightforward to demonstrate that no. 60 does not halt for standard configuration. It has only one exit at X_4, where it halts on a 0. A backward trace by the method of exits constructs a tree with minimal depth.

Productivity

Let T be a Turing machine of n states using only the symbols 0 and 1. Initially T scans only a blank tape. The machine T either halts in "standard configuration" - that is scanning the leftmost of an unbroken string of 1s on it otherwise blank tape, or it does not. If it does not it may either not halt at all, or halt scanning some other configuration. The **productivity** of T is defined to be:

$$p(T) = \begin{cases} \text{the length of the string that } T \text{ scans if it} \\ \qquad\qquad \text{halts in standard configuration} \\ 0 \text{ otherwise} \end{cases}$$

This is a function defined for each Turing machine. From this we may derive another function, $p(n)$ which is defined to be the productivity of the most productive n-state Turing machine.

Caveat emptor

Any solution by hand is subject to human error. The method of exits involves reversing symbols – reading a "move left" as a "move right" and vice-versa; marking a scanned state and a scanned symbol, and so forth. These reversals are mentally tiring and prone to error. No fundamental principle is involved should a casual error arise. Hence, subject to this caveat, the Halting Problem for all five-state machines is solved, and the productivity of five-state machines is 11.

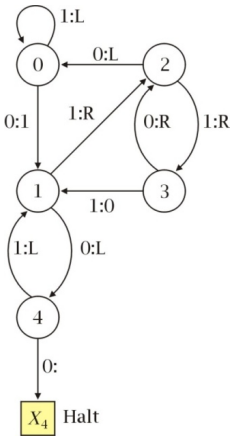


Figure 4. Holdout no. 60

There is no path backwards from this exit at X_4 to the standard configuration. In fact, this machine loops—is non-halting—for almost every tape configuration.

Exit at X_4
0_4
00_1
01_3 $0:1$ 001_4
$1:R$ 0010_1
0011_3 $0:1$ 00101_4
001_21 $1:R$ $00\overline{1}010_1$
$1:R$ $00\overline{1}01_40$ $00\overline{1}011_3$
LOOP LOOP

The solution to this problem is direct. There is no need to use an iterative method to break the machine down into a simpler machine of four-states to which an additional state is added. The method of exits establishes immediately that there are in fact only three non-halting inputs whatsoever.

There are approximately 7.5 million five-state Turing machines (after redundancies have been discounted). By running certain algorithms on these 7.5 million Turing machines, Kellett [2005] reduced the number of "hold-outs" to a mere 98 – a remarkable result. The fewer the halting configurations, the more likely the given machine is to exhibit apparently un-analysable behaviour. It is difficult to identify non-halting behaviour merely from the trace of a Turing machine. Having generated the trace through a substantial number of steps Kellett concluded that no. 60 is a holdout.

In the literature, a procedure akin to the method of exists is called *backtracking* and many of those machines classified as non-halting have been done so by application of a backtracking algorithm run on a computer. On considering the formulation of backtracking in Kellett's paper, we observe that the same notation for tape configurations that has been independently developed here is also utilised. For instance, considering a specific case, Kellett traces halting behaviour back to a configuration 11_3 - exactly our own notation. But from this situation (a) Kellett's algorithm only explicitly checks the combination 1_3 and hence does not use all the information available, namely the whole configuration; (b) the tree structure generated by method of exits is not identified. The discussion of backtracking offered by Kellett is instructive.

(1) It is an error to assume that the backtracking algorithm implemented on one machine could not solve the halting problem for a class of other finite machines, as stated in the quotation from Machlin & Stout referenced in Kellett [2005]. A backtracking algorithm implemented on a finite machine albeit "large", could solve the problem for a machine with a smaller number of states. Therefore, no contradiction is implied in principle.

(2) It is true that the algorithm may "infinitely backtrack" however, it is not true that the non-halting behaviour cannot be identified as a consequence. The algorithm infinitely backtracks because it enters a LOOP configuration – one that can be identified from the method of exits, as shall be illustrated below. The information contained in the loop is finite, and may be expressed as such by finite symbols. Any repeated sequence is determined by the machine itself. Hence, by the introduction of the over-bar and double over-bar symbols as well as the concept of a LOOP, the method of exits does generate a finite tree with branches that terminate. All the non-halting configurations of a finite machine may in principle be identified by the method of exits, and its halting problem resolved.

(3) An error lies in the understanding of the nature of the impossibility proof for the halting problem. It does not prove that any given halting problem is unsolvable, only that *if* machine *A* solves the problem for machine *B*, *then* machine *B* has more states (is "larger") than machine *A*. Furthermore, it says nothing directly whatsoever about human reasoning. It shows that any algorithm must be implemented on a machine with actually infinite states – which, as already remarked – means that it is not a machine. The limitation imposed by the fact that all machines have *finite* states has been overlooked.

(4) Any limitation on solving the halting problem for a given Turing machine of *n* states is practical, not theoretical. That is, in principle, the problem can be solved, but in practice the solution may require more time than the history of the universe will allow. But this is not a situation unique to the halting problem – it is the same situation that applies to all algorithms whatsoever – that there will be a number that is too big for either a computer or the human mind to manage. In other words, this situation arises from the nature of infinity, and is nothing unique to the halting problem. This is another underlying conceptual error in the history of this problem.

Back tracking formalization

Now that we have seen how the backtracking algorithm works in practice, we can define the concrete algorithm implemented in our program. This algorithm ... is adapted from that put forth by Machlin & Stout (1990) and as they so gracefully put it: "While backtracking can be useful, it cannot be guaranteed to always stop since otherwise it would supply a solution to the halting problem." Intuitively, therefore, some non-halting Turing machines cannot be proven as such by this procedure. Attempts to apply this procedure to these machines causes the algorithm to "infinitely backtrack." As a result of this problem, we are forced to specify a step limit pertaining to how far the procedure is allowed to "backtrack". If this limit is reached, the results are also inconclusive.

Kellett [2005], Section 4.2.1.2 page 53.

Example 2

Holdout machine no. 54 (also designated Kellett A.79). The solution to this machine is given in Appendix 4.

This example illustrates a slightly more demanding situation; we solve the problem in two stages, by first constructing a four-state machine by removing one of the states, and then adding that state back. This example thus illustrates the iterative method for solving halting problems. Additionally, the loops in the program lead to loop configurations, and we illustrate how we deal with those. The loop configuration is identified and then represented as finite information by use of the over-bar notation. This example introduces the basic manner in which steps of the program through the loop structure represented by the over-bar symbol are dealt with. The program steps through a loop structure as if it were a single instance – the whole point being that in a loop the same process always recurs.

However, when using the methods of exits – which is a backward trace – a very important caveat applies – the backward trace generates a branching tree structure – so it is important to consider the situation where the juxtaposition of two repeat units of the loop structure creates a new permutation from which the program can exit. Thus, branches may be hidden within a loop.

It is vital to consider whether a loop could contain a potentially infinite number of permutations from which the program might exit while applying the method of exits. This is the sole way in which theoretically the method of exits could develop that situation assumed in the remark quoted above from Machlin & Stout [1990] of backtracking never ending. ("While backtracking can be useful, it cannot be guaranteed to always stop since otherwise it would supply a solution to the halting problem.") It is essential to the proof that backtracking always generates a finite tree with branches that terminate in LOOP, END or CONTRADICTION. We demonstrate that the tree is always finite when we prove the Complete Criterion Theorem in Section 5.

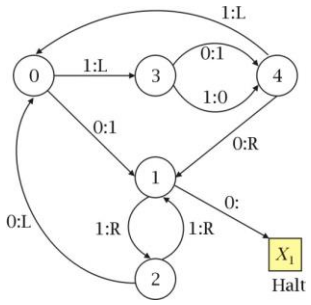


Figure 5. Holdout no. 54

Rendering loop configurations finite

The same process always recurs within a loop. This is illustrated by the following trace for holdout no. 54 used to verify one of the halting configurations for the four-state sub-machine.

$$01\overline{10}11_4 \quad 01\overline{10}1_01 \quad 01\overline{10}_311 \quad 01\overline{11}_411$$
$$01\overline{1}_0111 \quad 01_3\overline{1}111 \quad 00_4\overline{1}111 \quad 0_101\overline{1}111$$

One will see in the above trace that the shifts through states 3 to 4 to 0 in the loop recur for as long as the loop recurs; hence, we don't have to write out a potentially infinite sequence of such states – a single instance under the over-bar will suffice.

Hidden permutations

Suppose we have the LOOP structure represented by $\left\{\begin{matrix}\overline{11}\\ \overline{100}\end{matrix}\right\}$. Recall that this symbol represents the finite permutation of any repeated sequence of either structure. One instance of this structure is $11\ 11\ 100\ 11\ 100\ 100$. Here the repeat units have been spaced apart for clarity only. Now suppose in the complete criterion we have an input such as: $S_1 = 0_110,\ H = 11_20$. What this says is that if the tape configuration is ever 11_20 then that may have been reached from an input of $S_1 = 0_110$. In our example there are two instances of the permutation 110: $11\ 1\boxed{1\ 10}0\ 1\boxed{1\ 10}0\ 100$. If the program is in state 2 such that 11_20 at one or both of these instances, then a backward trace to $S_1 = 0_110$ becomes mandatory in the method of exits. Thus, the application of the method of exits can be demanding of patience.

Example 3

Holdout machine no. 20 (also designated Kellett [2005] B.4). The solution to this problem is given in Appendix 4.

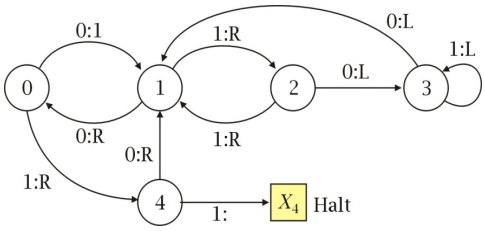


Figure 6. Holdout no. 20.

While we found most of the 98 hold-outs to yield solutions readily to the method of exits, no. 20 may be regarded as a difficult problem. However, there is an immediate direct proof that it is non-halting and we give that first. Then the solution is broken down into three stages: we construct firstly a three-state machine, T_3, then a four-state machine T_4, then finally the five state machine, $T_5 = \text{no. } 20$. The difficulty of the problem is illustrated by the introductory remark we make to the solution: there are four inputs to state Q_1, so direct solution of this machine, though possible, would lead to a tree with very many branches. A large number of non-halting configurations are introduced by the loop of period 1 given by the instruction $1 : L$ at Q_3. The effect of this instruction is to permute tape-configurations, and we call it a *permutator*. This will be explained in section 4. There are also loops of periods of 2, 3 and 4 states; there are double cycles between states Q_0 and Q_1 and between Q_1 and Q_2. The writing of the complete criterion for this machine is a difficult problem. This is because the width of the tree, though finite, is very wide.

We write the complete criterion for T_3 and then apply the method of exits to T_4 to obtain its tree. At that point a simple argument confirms the earlier conclusion that $T_5 = \text{no. } 20$ is non-halting. In the solution we omit the complete criteria of both T_4 and T_5.

The method of writing complete criteria shall be firmly established, and the finite nature of the tree for no. 20 is also implied. Since it is demonstrably non-halting we omit the tedious task of writing out all the halting configurations for T_4 from its tree; and then generating the tree for T_5. Actually, there is no inherent interest or purpose served by doing so. The individual halting problems have no practical or even theoretical application. The whole import of the topic – its significance cannot be underestimated – lies solely in the question: can the Halting Problem be solved or not?

Example 4

Kellett [2005] B5 Champion. Discussion of this machine is given in Appendix 5.

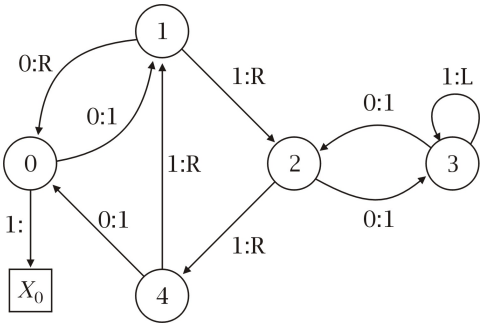


Figure 7. B5 Champion

The B5 Champion is a machine that computes a well defined productivity of 11. If all other 98 hold-outs are non-halting this demonstrates that the greatest productivity of any five-state machine is 11. That is, starting in standard configuration in state Q_0 the machine ends scanning the rightmost of a block of eleven 1s.

It is just as essential to the proof in this paper that those machines that halt for standard configuration should also be demonstrated. In the method of exits, we must prove that the tree is finite – that is, have finite depth and width – branches that terminate after a finite number of steps, and a finite number of branches. This is proven in section 5, when we prove the Complete Criterion Theorem. In our preliminary discussion the B5 Champion we write the complete criterion for a T4 sub-machine, and demonstrate that the starting standard configuration appears on a finite branch of the tree generated by the method of exits. The tree of halting configurations has enormous width as we shall subsequently explain, and to write it by hand is a practical task of no interest.

But this example does raise the possibility that (a) some among these branches are non-terminating, being potentially infinite in length, and/or (b) at some point in the generation of the tree structure a non-finite (potentially infinite) number of branches is generated. In either case, the tree would then become infinite, and the method of exits would fail to terminate. Then, the tree for such a machine could not be written, and its halting problem would not be solvable. Therefore, proving that the tree is always finite is essential. In the proof of the Complete Criterion Theorem, which is based on the method of inputs, we must prove that the permutations that are needed as inputs are finite.

One understands that finding the finite tree of this particular B5 Champion machine would not prove the result for *all* machines.

4. Permutators, loops and the method of inputs

We begin by illustrating the problem introduced at the conclusion of the previous section by further investigation of the case of the B5 champion. At several occasions in the generation of the tree by the method of exits, we encounter the following situation. The machine has entered state Q_3 scanning a 0 to the right of which there is an indeterminate block of 1s. Because of the 1:*L* instruction for Q_3 the program might have entered that state from any one of those 1s. This is shown in the trace of the tree for the B5 Champion as follows.

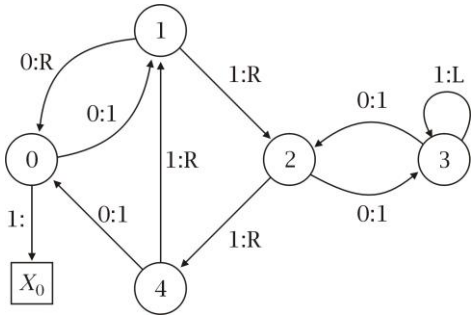


Figure 8. The B5 Champion

$0_3 1 1 \overline{111} 1 1 1 0$

$0_3 1 1 \overline{111} 1 1 1 0 \quad 0 1_3 1 \overline{111} 1 1 1 0 \quad 0 1 1_3 \overline{111} 1 1 1 0 \quad \ldots \quad 0 1 1 \overline{111} 1_3 1 0 \quad 0 1 1 \overline{111} 1 1_3 0$

Since the block represented by the over-bar symbol $\overline{111}$ is a block of three 1s repeated finitely and indeterminately, has not the width of the tree become potentially infinite at this point? This is the consideration we must eliminate, if our Complete Criterion Theorem is to go through. We also have to show that the length of every one of those branches is finite. To solve the problem about the width of the tree, we must show that the number of permutations that need to be considered at any stage of the generation of the tree is finite.

During the execution of its program a Turing machine may enter loops and cycles of instructions, but these loops and cycles always represent finite alterations to the tape configuration, and hence only finite permutations of configurations need to be tested. The testing is done by the *method of inputs* – input of all possible permutations of symbols of length sufficient to cover all possible alterations to the tape by the loop or cycle.

The over-bar notation is a means of dealing with the LOOP structures identified during execution of the method of exits. The possibility of this approach arises because in fact the width of the tape configuration – the number of 1s and 0s in any permutation – is circumscribed by what shall be defined below as the *period* of the machine given by the *method of inputs*.

$\overline{111} = 111\ 111\ 111\ 111\ \ldots$

Reducing potentially infinite to finite

When in a program we reach a line such as: $0_3 1 1 \overline{111} 1 1 1 0 = 0_3 1 \overline{111} 1 1 1 1 1 1 0$ note firstly that this has already determined something about the halting configuration – namely, that it comprises at least four 1s marked off by one 0 at each end. (The minimal instance of this configuration is $0_3 1 1 1 1 0$. To deal with the over-bar, we would now proceed to the following tape configurations.

$0 1_3 1 1 \overline{111} 1 1 0 \quad 0 \overline{111}_3 1 1 \overline{111} 1 1 0 \quad 0 1 1 \overline{111}_3 1 1 1 0$

$0 1 1_3 \overline{111} 1 1 0 \quad 0 \overline{111} 1_3 1 \overline{111} 1 1 0 \quad 0 1 1 \overline{111} 1_3 1 1 0$

$0 1 1 1_3 \overline{111} 1 1 0 \quad 0 \overline{111} 1 1_3 \overline{111} 1 1 0 \quad 1 \overline{111} 1 1 1_3 1 0$

$0 1 \overline{111} 1 1 1_3 0$

This table of 10 configurations covers all the possible ways in which the configuration $0_3 1 1 1 1 \overline{111} 1 1 0$ might have been reached. Hence, we illustrate that at this stage the width of the tree really is finite.

Loops and cycles

To illustrate what is meant by the *period* of the machine and the *method of inputs*, consider a sub-machine of the B5 Champion.

In *figure 9, t*he sub-machine in question is the loop marked in red. This is a loop of three states each connected by the instruction $1:R$. A machine will only move through this loop more than once if the tape configuration requires it to do so. For instance, to exit at state Q_1 the input configuration is $\left.\begin{matrix}1_4\\1_2 1\\1_1 11\end{matrix}\right\}\overline{1110}$. The *shift number* of a loop is the number of separate states in the loop. The sub-machine will always exit somewhere if there is a 0 on the tape to the right of any scanned symbol. The *period* is the number of squares on the tape required to be read when the machine passes once through the loop. The *step number* is the number of steps the program runs through when executed. The step number is greater than the shift-number when the same state is invoked repeatedly during execution: in this example, when at state Q_3 we have the $1:L$ instruction.

We will define an *n-cycle* to be a concatenation of *n* loops. Thus, a loop is a 1-cycle. See *figure 10* for an instance of a 2-cycle in the B5 Champion.

The method of inputs

The *method of inputs* is to input test data at each input equal to every permutation of the period of a loop or *n*-cycle. Thus, in the above example, if every permutation of three symbols fed into the red-loop at each input will suffice to characterise all the outputs of that loop. From this the complete criterion for the red-loop can be written. At this point the over-bar notation is reintroduced to represent the possibility of repeated instances of a tape configuration corresponding to the period.

It will be immediately seen that the method of inputs is very "expensive" in terms of the use of resources. Instead of constructing one tree by the method of exits, we must work forward through 2^{ρ} permutations, where ρ is period of the cycle, and do this for *n* states, where *n* is the number of states in the cycle. The number of steps required can grow inordinately, as shall be shown shortly.

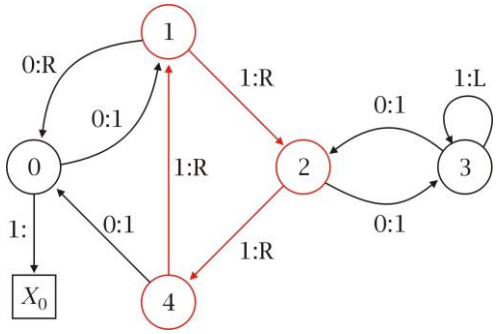


Figure 9. B5 Champion. Red loop.

Period and shift number

In this example the shift number and the period are the same; if we were to alter one of the $1:R$ to $1:0$ then the period would be one less than the shift number, that is, 2. If we were to replace one $1:R$ to $1:L$ then the period would be just 1.

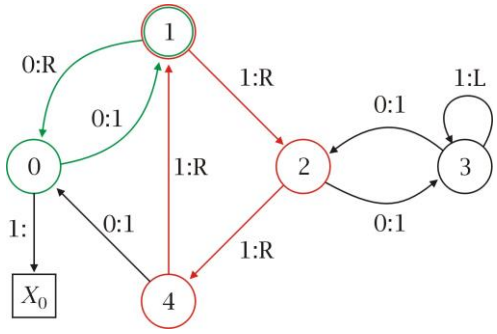


Figure 10. B5 Champion. Red-green 2-cycle.

Stepping through the cycle

The program can (and does) move through the red loop followed by the green loop back to the red loop in a cycle whose shift number is 5 and period is 4 (four *R* moves).

Inputting permutations

To write the complete criterion of the 2-cycle defined by the red and green loops combined requires input of all permutations of four symbols at each input, because the period given by the four *R*-moves is 4. There are 2^4 inputs, and since the number of states is 4, then $4^2 \times 2^4 = 1024$ traces are required – not, apparently, an excessive number. The length of each trace is given by the shift number, which in this case is equal to the step number; the shift number being 5, this makes $5 \times 4^2 \times 2^4 = 5120$ steps.

Routine of the method of inputs, ghost loops, changing the permutation

Run instances of each permutation starting at each input through the step number. If the program returns to the same permutation, it is a loop; otherwise, it is a halting configuration and exits somewhere. Check to see what the output of the routine is. The over-bar notation is introduced at the completion of the method of inputs. Until now, we have considered the effect on the method of inputs of combining two loops into a 2-cycle – the red-green cycle of the B5 Champion has period 4 and shift number 5. To determine the complete criterion, we must test at each input (state) each of 2^4 permutations to a depth of 5 steps, here equal to the shift number. One thing to observe about the B5 Champion is that there is a *ghost loop* between states Q_4, Q_0 and Q_3. The reason why this is an illusory loop (a "ghost" loop) is because at Q_4 the instruction is $0:1$; hence, if the machine reaches state Q_0 from Q_4 then it must already be scanning a 1, and cannot move to state Q_1. We redraw the flow-diagram to express this observation (*figure 11*).

Now we must consider the effect of the addition of the loop between states Q_2 and Q_3. In *figure 11* we have marked this loop in blue. What is the effect of this blue loop on the tape configuration? We have just seen that the complete criterion for the red-green cycle combined is fully described by inputs of every permutation of four symbols. To enter the blue cycle the program must leave such a permutation having moved right onto a 0 scanned at state 2. So, the tape looks like the following – in the empty four boxes there is a permutation of 1s and 0s.

$\square\ \square\ \square\ \square\ \boxed{0_2}$

On leaving state Q_2 the last 0 is converted to a 1 and we enter state Q_3.

$\square\ \square\ \square\ \square\ \boxed{1_3}$

The instruction $1:L$ forces the machine to scan to the left until it finds another 0, which it then changes to a 1. By doing so the machine *changes the permutation* that is being read at state Q_2. The effect of the red-green cycle is fully described by a permutation of four symbols; the period of the red-green cycle is 4. The greatest length of the tape that needs to be considered is found by supposing that the machine in state Q_3 steps to the left to find a 0 leftmost in a new block of four symbols, also to the left of the old block. In this, we also consider the fact that in the red-green cycle there are only moves to the right.

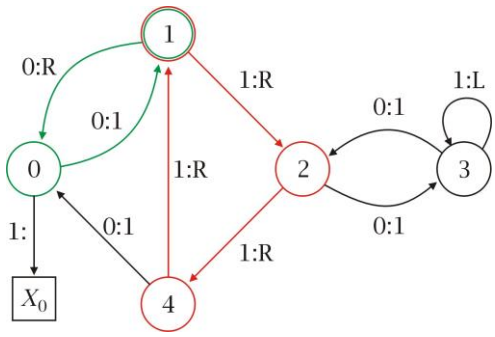


Figure 10. B5 Champion. Red-green 2-cycle.

Loops of the red-green cycle

shift	tape configuration	
0	... 0011 $\boxed{0_1011}$ 0011 ...	
1	... 0011 00_011 0011 ...	
2	... 0011 01_111 0011 ...	
3	... 0011 011_21 0011 ...	
4	... 0011 0111_4 0011 ...	
5	... 0011 0111 $\boxed{0_2011}$ ...	$0_1\overline{0110} \rightarrow \overline{0110}0_1$
	LOOP	
0	... 1111 $\boxed{1_111}1$ 1111 ...	
1	... 1111 11_211 1111 ...	
2	... 1111 111_41 1111 ...	does not enter
3	... 1111 $111\boxed{1_1\ 11}11$ ...	the green cycle
	LOOP	$1_1\overline{111} \rightarrow \overline{1111}_1$
1	... 1100 1_1100 1100 ...	
2	... 1100 11_200 1100 ...	
3	... 1100 110_40 1100 ...	exit at X_4
	HALT	

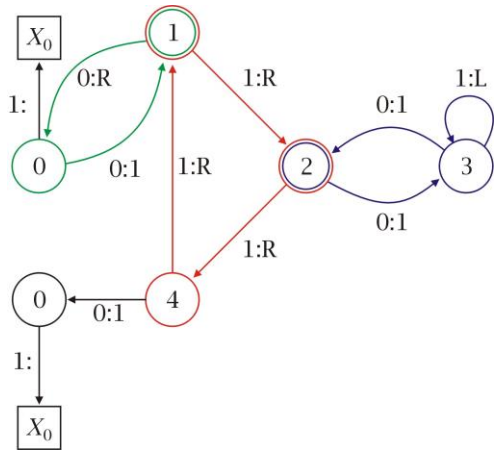
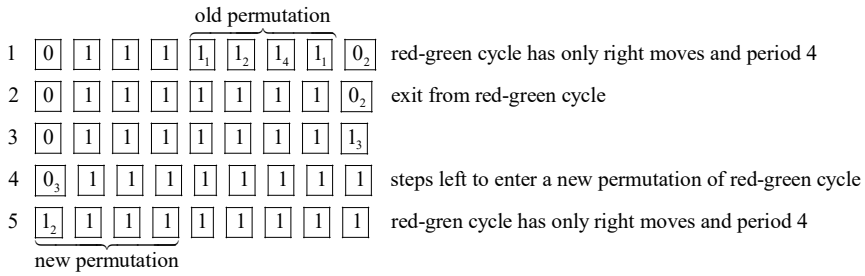


Figure 11. Ghost loop and the red-green-blue cycle

Permutators

The investigation of the red-green-blue cycle of the B5 Champion tells us a lot about the configuration of the tape required to make these changes take place.



We can only obtain a maximal period if there are two 0s on the tape separated by seven 1s. Any insertion of the repeat unit $\overline{111}$ into this block does not change the permutation at the end of the process. Although it would seem that by line 5 there has been no change in the tape configuration there has: the state in which the first 1 in a block of four 1s is being scanned has changed from Q_1 to Q_2.

The $1:L$ structure is a PERMUTATOR and its effect here is to create a blue-red-green cycle of period 9 and shift-number 13. The maximum number of steps in the execution of the program is 21.

The program steps through repeated instances of a loop as if it stepped through just one of them. Hence, we use the over-bar notation to remove the extraneous information and identify the tape-configuration. This is the essential insight upon which the *finite* nature of Turing machines is recognised. Without it, Turing machines *appear* to enter incomprehensible infinite configurations from which their halting or non-halting behaviour cannot be identified. However, this is an illusion.

The maximum period introduced by a permutation is $2\rho+1$ where ρ is the period of the cycle to which the permutator is joined. A permutator changes the permutation that is currently being tested. But as all permutations are tested by the method of inputs systematically, this cannot cause the branch of a tree not to terminate.

The question is – what is the length of the permutation that we must now test in order to ensure that we have found every halting configuration, and thereby also every non-halting configuration?

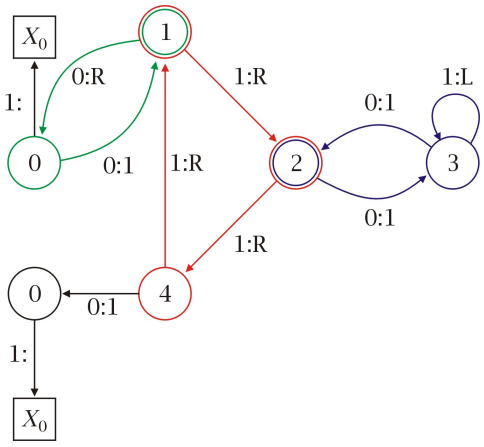


Figure 11. B5 Champion. The red-green-blue cycle.

The effect of the permutator

On input of test data 010101_1010 the program halts on $0101\boxed{11110_4}$ after 9 steps; only the data marked in the box has been used – the 0101 to the right of those entries was redundant; the cycle was in fact tested by a period of just 5, less than the maximum. Suppose we input something more than the maximum.

011111_111100	two 0s separated by eight 1s
01111111102_0	
01111111113_0	
$0_31111111110$	two 0s separated by nine 1s

Apparently we have reached a permutation of eleven symbols – larger than the maximal period of nine symbols. But this is an illusion. Within this configuration there is a repeat unit of $\overline{111}$ through which the red cycle merely loops. It is essential to eliminate redundant loops.

$011\overline{1_111}100$	eliminate redundant loops
$011\overline{111}10_20$	
$011\overline{111}1_30$	
$0_3\overline{111}0$	eliminate redundant loops

In this example we have in fact reached a permutation of 5 symbols, less than the maximum.

To test the blue-red-green cycle, by the above argument, the maximum length is 9. To start in standard configuration at Q_0 and assuming an exit at Q_4 we require two shifts $Q_0 \to Q_4 \to Q_2$, but as one of these is $0:1$ its period is increased by just 1. To exit at $Q_2 \to Q_4 \to X_0$ is also two shifts and adds 1 to the period. The maximum length of the permutation to be tested is 11. The maximum step number is 23. This is consistent with the known result for the input of standard configuration – eleven 0s are changed to eleven 1s and the machine halts scanning the leftmost of these in state Q_0 having left Q_4. The machine could exit via state Q_1, an addition of three shifts, adding 2 to the period, making a maximum length of the permutation to be tested as 12. The step number is 24. So, to write the complete criterion of this machine, we must input all tape configurations comprising 2^{12} permutations of twelve symbols (1s or 0s); to these we must add the state symbol of the initially scanned state; one state symbol may be inserted at any one of 12 places; and there are 5 such states. Thus we must test $5\times12\times2^{12} = 245760$ inputs in all.

Now the question becomes – for how many steps do we need to run each input through the machine? Since the blue loop is a permutator, it has the effect of starting with one input configuration at Q_2 and replacing it by another, and in a sense before we have finished with the first configuration – so, *we are starting the test all over again*. But if this process continues then it can do so only finitely. In the course of the algorithm, one of the $5\times12\times2^{12}$ configurations at Q_2 is reached immediately, and others subsequently. Therefore, if we run the program long enough we may be sure that either (a) the machine will halt for the given starting input, or (b) an exact replica of an input configuration will be reached, in which case that starting configuration and all others generated by that input are non-halting. Hence, to be sure that we find every halting and non-halting configuration for this Turing machine, we must run each of the $5\times12\times2^{12}$ tape configurations through $5\times12\times2^{12}$ steps. For the complete test a total of $\left(5\times12\times2^{12}\right)^2$ steps, approximately 60 billion. But this is the upper limit – since each configuration is tested repeatedly in this process, the number of steps required in practice is much less. The lower bound for the entire test is provided by the number of tape configurations we need to test, which is $5\times12\times2^{12}$. We observe, for instance, that the configuration 010101_1010 halts after 9 steps in the red-green-blue cycle, so much less than $5\times12\times2^{12}$ steps.

5. Proof of the Complete Criterion Theorem

Before we proceed to that proof, we shall consider a final objection to its possibility. The insight upon which the whole resolution of the Halting Problem is based is that in a Turing machine of finite states (it could not have actually infinite states) whatever happens is thoroughly determinate and finite. It is simply the plethora of combinatorial possibilities that has mesmerised us into not abiding by this intuition: *it is finite, therefore, the solution is also finite*.

The only way in which a machine could become indeterminate and non-finite in its behaviour would be if it were "irrational" – for example, it randomly generated numbers (in binary) in some wholly unpredictable way. Therefore, the objection runs: *what could be easier? We have algorithms that generate irrational numbers.* For example, we have many algorithms for the generation of π. One of these is $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots$. As is well-known π has a decimal expansion in which no discernible pattern in the sequence of digits can be found; it is effectively a random number generator. So, let us make a Turing machine that churns out the decimal expansion (in binary) of π, and that will clearly have entered into an infinite non-halting loop, for which there is no discerning its halting or non-halting behaviour if that machine should we not already know what it was designed for.

In reply: (1) the Turing machine that could generate any respectable stages of the algorithm indicated by $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots$ would be a *very large* Turing machine. We have stumbled in our analysis over the complexity of occasional instances of Turing machines with merely five states. According to the literature, there are approximately 760 million Turing machines with six states. (See Kellett [2005].) In other words, the halting problem for any machine that could implement the above algorithm to any interesting degree would be practically vast; *but not theoretically so*. It is one thing to *know* that the halting problem can be solved for a machine of arbitrary *n* states, and another thing to actually compute its complete criterion and thereby solve that problem. In summary the distinction between *knowing* and *computing* has constantly been overlooked in this matter.

(2) *It is not true that there is any machine that can compute* π. Surprising though this statement may seem, *every machine is finite*, so all the most powerful computer in the whole wide world can do in this matter is compute *a finite approximation* to π. Being finite, the computation is that of a finite (rational) number. As such it will be represented by a (very long) binary sequence that will constitute the output for a Turing machine started in standard configuration. Such a machine will halt. If there are *n* symbols in the binary representation of that number, then one can well boggle at the vast size of the number of steps needed to confirm that such a machine really does halt. We have just seen an example of a five-state machine that may require as many as 10 billion steps to achieve this.

Always in this topic the actually finite nature of any Turing machine is forgotten. It is true that from the theoretical point-of-view the Turing machine in general is *potentially* infinite. This means that given any Turing machine of *n* states it is possible to build another Turing machine of $n+1$ states. But even this is a theoretical possibility belonging more to the domain of (abstract) number theory than to practical life, for we cannot build a machine that will exhaust all the matter of the universe. Therefore, to acknowledge that Turing machines are in theory potentially infinite is not to say that there ever has been a Turing machine that was not actually finite; the actually infinite Turing machine is also a creation of pure fantasy on the one hand, or legitimate theoretical speculation on the other – but to confuse the two is surely wrong.

Equivalence of the method of exits and the method of inputs

Once the theorem is proven it follows as a corollary that the method of inputs and the method of exits are equivalent. As they both write the complete criterion, then they must be equivalent, for there is only one complete criterion, subject to equivalences of notation.

In the language of set theory, a complete criterion is a set of equivalence classes of symbols. This makes any complete criterion into a second-order object in either set theory or logic. Since it is not first-order it unlikely that it is computable. And yet, the human mind can write it. There is no reason to presume *a priori* that computability and human reason are equivalent.

Computing Pi

The day on which this monograph was completed we have the following announcement.

The value of the number pi has been calculated to a new world record length of 31 trillion digits, far past the previous record of 22 trillion. Emma Haruka Iwao, a Google employee from Japan, found the new digits with the help of the company's cloud computing service. … The calculation required 170TB of data (for comparison, 200,000 music tracks take up 1TB) and took 25 virtual machines 121 days to complete.

www.bbc.com/news/technology-47524760

The 31 million digits calculated makes up a finite and rational number. Infinity cannot be beaten.

The Halting Problem for the machine Ms. Iwao employed in this calculation would be vast beyond imagination, yet still finite. In theory, its complete criterion could be written; in practice, it could require more resources than the universe could provide.

Proof by mathematical induction

Recall that a *complete criterion* for an *n*-state machine is a specification of those tape configurations for any inputs to the machine that cause that machine to halt, specifying the exit at which it halts, and giving completely the tape configuration at any given exit. The **Complete Criterion Theorem** states that for any finite Turing machine of *n* states, it is possible to write a complete criterion.

We now proceed to the proof of the Complete Criterion Theorem. In the proof, we refer at times to the method of exits, but the proof proceeds by mathematical induction on the method of inputs. Clearly, a complete criterion can be written for all 1-state Turing machines. In the induction step we assume that we have a complete criterion for a machine of *k* states. The task is to show that granted this assumption a complete criterion may be written for a machine of $k+1$ states. We start with a Turing machine T_k of *k* states and assume that it has a complete criterion. We add a further state, designated Q_{k+1}, and argue that a complete criterion for the resultant T_{k+1} machine can be written. The inductive step takes the form represented by *figure 12*.

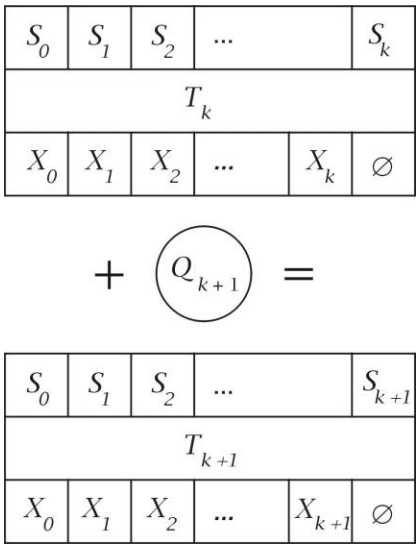


Figure 12. The Inductive Step

To prove the induction step, we exhaust all the cases, demonstrating in each case that if the complete criterion for T_k is given, then the number of permutations needed for the method of inputs is finite. This also entails that in the method of exits the tree for T_{k+1} is finite.

It may happen that a given machine *T* comprises two sub-machines between which there is a *ghost connection*. Suppose we have two machines *M* and *N*, and an edge in the flow-diagram from an exit in *M* to an input at *N*. Now suppose that in fact *M* never reaches this exit for any input configuration, so that there never is an input to *N* from *M*. Then *T* comprises two separate machines between which there is only a ghost or redundant connection that may be deleted. Hence, without loss of generality we assume in the induction hypothesis that T_k is a *k*-state machine in which there are no ghost connections. Since there are no ghost connections, each state of the machine can be reached from every other state by input of at least one tape configuration. This entails that there are no isolated loops within the machine, and that T_k has a maximal cycle with period ρ and shift number σ.

(1) There may be any number of inputs from the exits of T_k to state Q_{k+1}. These inputs to Q_{k+1} are represented in the graph of T_{k+1} by at most a finite number of edges (*figure 14*). In the method of exits a tree is generated: every new edge in the graph joining T_k to Q_{k+1} produces in that tree the possibility of a new branch. As some of these branches may result in tape contradictions, the number produced will be less than or equal to the number of inputs (edges) from T_k to Q_{k+1}. If the maximal cycle T_k has period ρ, then in the method of inputs, the maximal permutation that needs be tested is $\rho+1$. In the method of exits, the tree for T_{k+1} is finite.

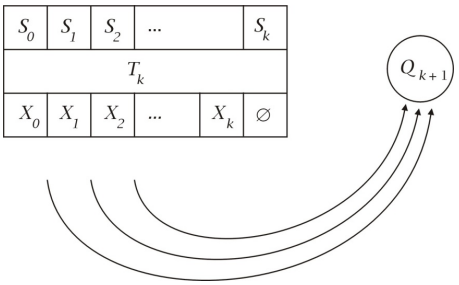


Figure 14. Case 1. Exits from T_k to Q_{k+1}.

(2) There are at most two outputs from the state Q_{k+1} corresponding to the instructions for Q_{k+1} for the symbols 0 and 1 (*figure 15*). Any symbol for which the instruction at Q_{k+1} is not given will define a halting exit from the machine T_{k+1}. Such an exit will introduce no loop. As per the preceding paragraph, since there will be an input to Q_{k+1} from T_k it will increase the shift number by 1, the period by at most 1 and the number of steps of the machine to reach it by 1. If both 0 and 1 lead to the exit X_{k+1} the same conclusion follows. The tree for T_{k+1} is finite.

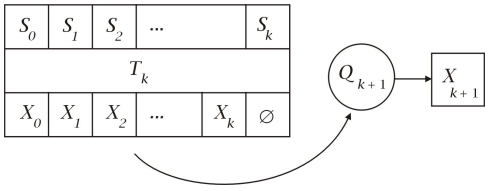


Figure 15. Case 2. Exit from Q_{k+1}.

(3) There may be at most two inputs (edges) leading from Q_{k+1} to the machine T_k (*figure 16*). Each introduces a loop to the machine. We begin with the addition of one loop. By the induction hypothesis T_k is a single machine with no ghost connections and a maximal cycle of period ρ and shift number σ. An input from Q_{k+1} to T_k must join this cycle. It may join and exit at the same state, or it may join and exit at different states. If it joins at the same state (*figure 17*), then the new maximal cycle formed by the addition of Q_{k+1} has maximum period $\rho+2$ and maximal shift number $\sigma+2$. If it joins at a different state (*figure 18*) then the new cycle starts at the exit *X* but follows the old cycle in T_k once (red in the above diagram), before leaving that cycle at *X* then passing through Q_{k+1} it returns to the old cycle at least one shift down from *X* and follows the cycle (blue) until it reaches *X* again. So the new cycle has maximum period $2\rho+1$ and maximum shift number $2\sigma+1$. At most a cycle of maximum period $2\rho+1$ is added. In the method of inputs the length of the permutation that must be considered is also increased by a maximum of $2\rho+1$, a finite number. The tree for T_{k+1} is finite. The addition of a second loop is just an iteration of the preceding case.

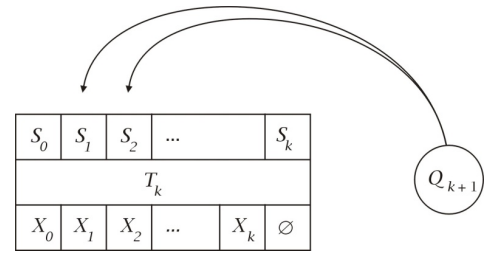


Figure 16. Case 3. Inputs from Q_{k+1} to T_k.

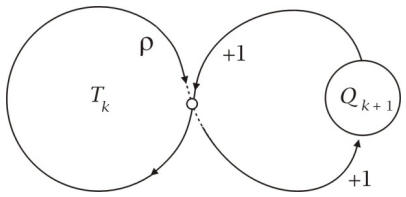


Figure 17. Loop from Q_{k+1} exits and joins from the same state in T_k.

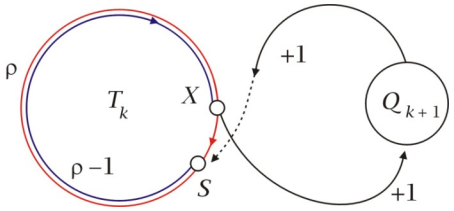


Figure 18. Loop from Q_{k+1} exits and joins from different states in T_k.

(4) Suppose that a loop is introduced into state Q_{k+1}. Furthermore, suppose that this loop constitutes a symbol change (*figure 19*). The only possibilities are $0:1$ and $1:0$; the analysis for each is the same. The symbol change introduces the possibility of a tape contradiction, so that in the method of exits the tree for machine T_{k+1} may terminate at Q_{k+1}. However, assuming that the instruction is consistent for some tape configurations, then the instruction reduces the potential number of loops introduced by Q_{k+1} to just one; this does not affect the analysis of the preceding paragraph, save to make the increase in shift-number 3 and the increase in period at most 2. So the new cycle has maximum period $2\rho+1$ and maximum shift number $2\sigma+2$. In the method of inputs the length of the permutation that must be considered is increased by a maximum of $2\rho+1$, a finite number. The tree for T_{k+1} is finite.

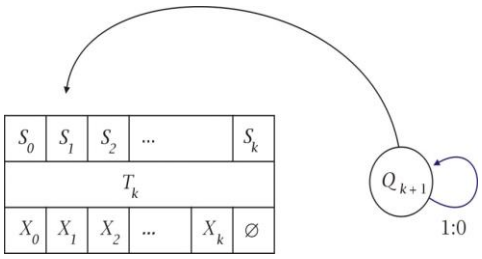


Figure 19.

Loop at Q_{k+1} with a symbol change.

(5) Now suppose that the loop is a $0:L, 1:L, 0:R$ or $1:R$ instruction (*figure 19*). If two such loops are added at Q_{k+1} then it is immediate that the machine enters an infinite loop if, and only if, it reaches Q_{k+1}. Therefore, we consider the case where there is one such loop structure, and the other exit from Q_{k+1} is an input at T_k. Such a loop is a **permutator**. That is, it has the effect of changing the loop configuration at the input S_i from one permutation to another. Let the maximal cycle in T_k have period ρ and shift number σ. To deal with the case of the permutator we argue in two steps.

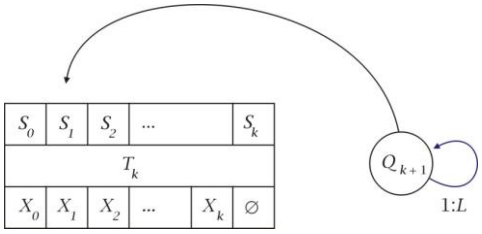


Figure 19.

Loop at Q_{k+1} with a permutator.

Step 1. Convert the permutator temporarily to an exit (*figure 20*). This means that the machine exits at Q_{k+1} scanning a symbol $\tau \in \{0,1\}$ and returns to the cycle on the complement, τ'. This creates a temporary machine T'_{k+1} of $k+1$ states. By the induction hypothesis we have the complete criterion for T_k. By the argument of section (3) we have a new maximal cycle. Then by the preceding analysis the new maximal period arising from the adjoining of the loop to the cycle T'_{k+1} is $\rho' = 2\rho+1$ and the new maximal shift number is $\sigma' = 2\sigma+1$. That is the maximal cycle in T'_{k+1} now has parameters ρ', σ'. By the method of inputs or the method of exits write the complete criterion for T'_{k+1}. This identifies both the halting and non-halting configurations of T'_{k+1}.

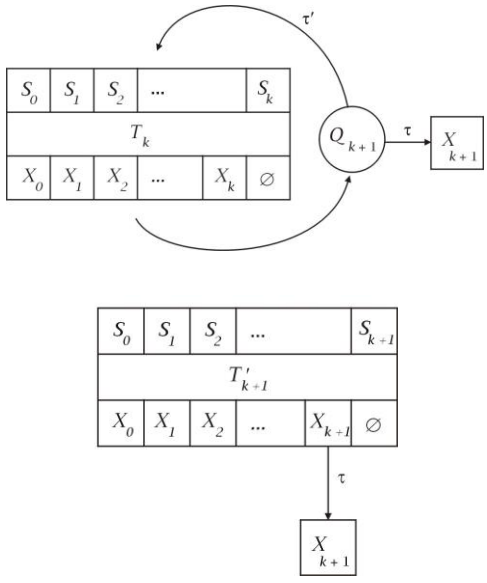


Figure 20.

Temporarily replacing the permutator by an exit.

Step 2. Replace the temporary exit by a permutator (*figures 21, 22*). This generates the full T_{k+1} machine. In this analysis we are treating T'_{k+1} as a cycle that includes Q_{k+1} with parameters ρ', σ'. But in order to enter the permutator loop we must regard the program as having left that cycle because it is scanning the alternate symbol at Q_{k+1}; the other symbol would cause it to remain within the cycle. We can show this move by adding a dummy state to the diagram (*figure 23*). The effect of the permutator is to switch from one tape configuration to another. The permutator may progressively introduce redundant blocks into the tape configuration (*figure 24* below).

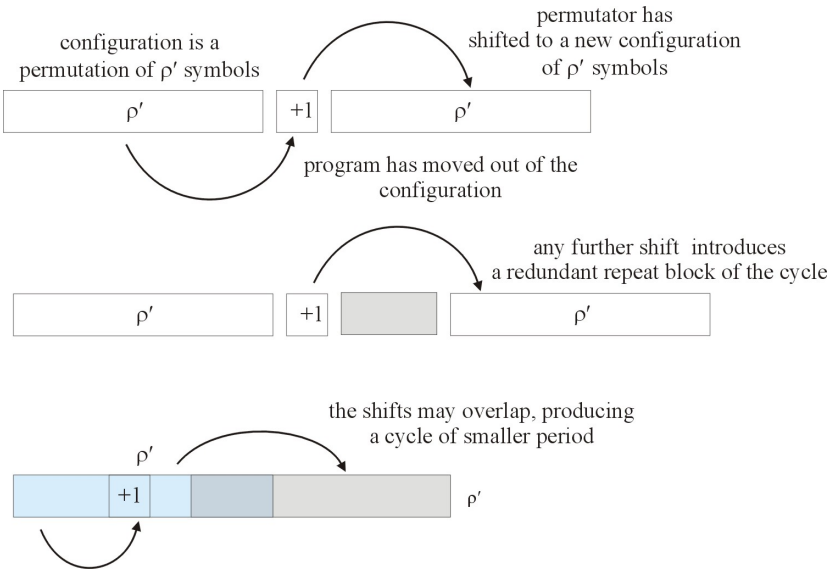


Figure 24. The permutator may introduce redundant blocks into the tape.

From the above the cycle with the permutator has maximal period $2\rho' + 1 = 2(2\rho + 1) + 1 = 4\rho + 3$, and similarly with the shift number. To prove that the method of inputs is finite introduce a test configuration of length $2\rho' + 1$ on an otherwise blank tape. Run the machine forward; either:

A. The machine does not exit the cycle (does not reach the permutator), or
B. It does exit the cycle at the permutator.

In the case of *A*, then since we have the complete criterion for T'_{k+1} this configuration belongs to those that do not exit at X'_{k+1} and this is added to the complete criterion for the full machine T_{k+1}. It will either exit and halt elsewhere or represent a non-halting configuration, one that has already been identified.

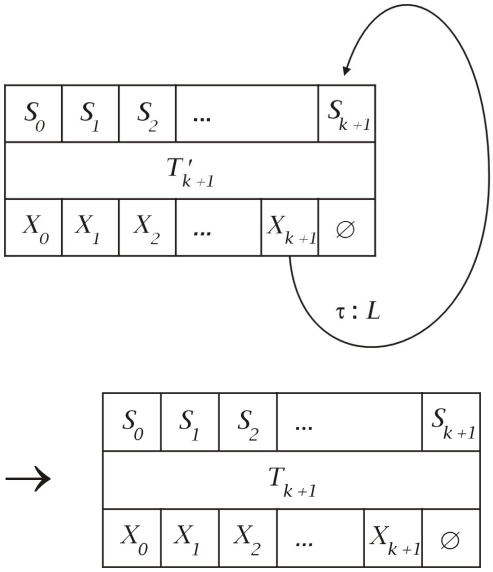


Figure 21.
Replacing the temporary exit by a permutator.

This diagram in this figure represents a program that in the flow diagram would appear as in figure 22 below.

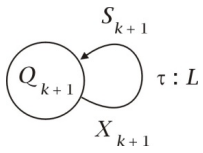


Figure 22. A permutator in a flow graph.

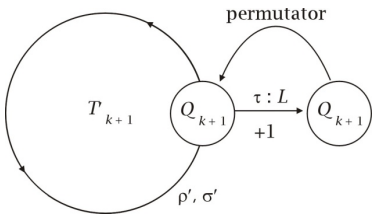


Figure 23.
Representation of a permutator by means of a dummy state.

In the case of *B*. The induction hypothesis is that we have the complete criterion for T_k from which we obtain that for T'_{k+1}. Therefore, we know everything about the halting and non-halting behaviour of the blocks represented in the above diagram with period ρ'. Examine the tape and identify if the permutator has immediately entered the permutator LOOP. This will occur, for instance if the permutator is $0:L$ and the scanned symbol is a 0 to the left of which is a completely blank tape. If the program does not enter this LOOP then it is because the alternative symbol (here a 1) is somewhere to the left of the scanned square of the tape. (Since the input date was placed on an otherwise blank tape, the whole tape is determined, and it is known whether this symbol is on the tape or not.) Run the program forward to where it leaves the permutator; this occurs when the symbol changes. (In our example from a 1 to a 0.) Examine the tape. Replace all instances of a LOOP configuration from the criterion for the T'_{k+1} machine with a single instance, and mark this by an over-bar. The over-bar is needed to identify any LOOP. This step is essential in order to identify the LOOP structures. By means of the over-bar we reduce the information in a potentially infinite repetition of the same configuration to a finite expression. The aim is to ensure that at each stage a new configuration of $2\rho'+1$ symbols is examined in the cycle. This will create a new permutation of $2\rho'+1$ symbols without redundant LOOP configurations from the maximal cycle of the T'_{k+1} machine.

The effect of the permutator on the cycle T'_{k+1} is completely described by the procedure of testing two instances of tape configurations run in the cycle side-by-side. Denote any instance of the configuration of the cycle (of length ρ' by π. Let two of these instances be π_1 and π_2, and let there be between them the symbol that causes the program to engage the permutator, τ, at Q_{k+1}. For brevity we will denote this also by τ, though in practice it might be its complement τ' if the program enters the permutator on the change of symbol instruction, $\tau':\tau$. So the entire test is of the concatenation $\pi_1 \oplus \tau \oplus \pi_2$, where $\oplus$ denotes concatenation (joining two strings of symbols at one end). First we test π_1 to see whether it leaves the cycle at the permutator. If it does not, then the complete criterion for all configurations $\pi_1 \oplus \tau \oplus \pi_2$ that begin with that π_1 configuration are governed by the complete criterion for π_1 for the machine T'_{k+1}. For those configurations that exit at the permutator, we test the second instance π_2 from the state, Q_{k+1}, where it re-enters the cycle.

Effect of the permuatator

Instead of testing a single instance of the tape configuration of the maximal cycle, π, we test two instances jointly. Concatenating these with the permutator τ we test the string

$$\pi_1 \oplus \tau \oplus \pi_2$$

This gives us a sequence of inputs tested.

$$\pi_1 ... \rightarrow \pi_2 ... \rightarrow \pi_3 ...$$

Eventually every permutation of the tape configuration of the maximal cycle is tested and classified as either halting or non-halting.

If the program exits at the permutator when π_1 is being tested then we have not completed the test of π_1 before identifying its halting or non-halting behaviour. The same will apply to π_2 if it too exits. The process is itself an iterative process. As we move around the cycle, we are testing equivalence classes of instances of the cycle configuration so each shift within the cycle eventually classifies one configuration and multiple instances of π are being tested together. When the test re-enters the cycle, all we have done is shift to another instance of π which we then proceed to test. The change of permutation makes no difference. We get an iterative sequence of tests: $\pi_1 ... \rightarrow \pi_2 ... \rightarrow \pi_3 ...$ Provided we guarantee that each instance of π is included in the test of inputs, this test must terminate. We must ensure: (1) that every instance of the form $\pi_1 \oplus \tau \oplus \pi_2$ is an input. (2) Redundant configurations which are repeat instances of tape configurations belonging to the cycle (its non-halting configurations) are removed. In the test of $\pi_1 \oplus \tau \oplus \pi_2$ we guarantee that neither π_1 nor π_2 are instances of the LOOP of the cycle. The instances of the LOOP in the cycle are non-halting configurations recognised and classified from the already constructed complete criterion for the cycle of T'_{k+1}. (3) The final thing we need to do is require that any given test is run through a sufficient number of steps to ensure that all instances of $\pi_1 \oplus \tau \oplus \pi_2$ encountered in that equivalence class are classified as either halting or non-halting. (4) We can only eliminate redundant configurations when inside the cycle of T'_{k+1}. Only then are they redundant. If we eliminate them during the permutator stage, some permutation may be omitted and the test becomes unsound.

The length of the permutation that must be investigated in the method of inputs is increased from ρ' to a maximum of $2\rho'+1$. Furthermore, since the permutator switches the input from one permutation to another, it is as if we had abandoned the test of one permutation before it was finished and started to test another one afresh. This is not a situation that can continue indefinitely (ad infinitum) because there are only a finite number of permutations of tape configurations that need to be tested. Furthermore, what we are now testing is not the configuration of period (length) ρ', but the configuration of length $2\rho'+1$. So, as the test proceeds we eventually reach either an exit (halt) or there will be a replication of a tape configuration by means of which we will identify a LOOP. As there are $k+1$ states in the T_{k+1} there are $(k+1)(2\rho'+1)2^{2\rho'+1}$ configurations to be tested.

There are $2^{(2\rho'+1)}$ permutations of a tape of length $2\rho'+1$; the starting state symbol may be inserted at $2\rho'+1$ places to make $(2\rho'+1)2^{(2\rho'+1)}$ tape configurations of one state; there are $k+1$ states; hence the number of tape configurations in all is $(k+1)(2\rho'+1)2^{(2\rho'+1)}$. As the permutation can be switched by the permutator by no more than this number of times, we can be sure to reach either an EXIT and halt or a LOOP within $(k+1)(2\rho'+1)2^{(2\rho'+1)}$ steps. The number of permutations in the method of inputs is finite. Though this may be a large number, the tree generated by the method of exits for T_{k+1} is also finite. (In most cases it will be a much more efficient method of identifying the complete criterion.)

The great practical difficulty of the method of inputs is the vast size of the sample space, which may mean that in practice a complete test cannot be conducted, simply owning not to any theoretical constraint but to practical limitations – time, patience, man-hours, human error, computing power and so forth. But mathematical induction is a theoretical tool that establishes knowledge not necessarily practical tests. Therefore, it is no limitation upon mathematical induction that in the case of Turing machines we hit the buffers of practical limitations relatively early on; we encounter exactly the same situation with every instance of a result established by mathematical induction.

While the method of exits is much more efficient than the method of inputs, it too has a practical (not theoretical) limitation that can make it very difficult to implement – this practical limitation occurs when the branching of the tree generated by the method of inputs becomes very large – the tree becomes very wide, though it remains finite.

Conclusion

Hence the induction step holds: in all cases, either by the method of inputs or by the method of exits the complete criterion for T_{k+1} may be written. Hence, by mathematical induction, for all Turing machines of *n* states, the complete criterion may be written.

Corollary

The Halting Problem may be solved for any Turing machine.

The B5 Champion again

The B5 Champion is an example of a machine for which the problem of writing the complete criterion by either the method of inputs or the method of exits is very large. By the method of exits the tree generated becomes very wide. The method of inputs is avowedly very expensive in resources and we have already indicated that $5\times12\times2^{12}=245760$ inputs must be tested, and a upper bound of 60 billion steps calculated. The underlying problem is that the length of the tape configuration being tested (the period of the whole cycle) is already 12; secondly, that almost all tape configurations are non-halting – which accounts for the extensive branching of the tree. However, we illustrate the process of the method of inputs described for the permutator. This is given in Appendix 5.

1. $\overline{=}0010{0_0}1110010100\overline{=}$

   Halts in 29 steps, 23 tape configurations are classified

2. $\overline{=}0100_0110000000\overline{=}0$

   Halts in 103 steps, 66 tape configurations are classified

3. $\overline{=}00000_0000000000\overline{=}$

   Halts in 47 steps, 47 tape configurations are classified.

4. $\overline{=}0\ \overline{1_211}00\overline{=}$

   A non-halting LOOP of the B5 champion.

The last example is an instance of a non-halting configuration that does enter the permutator. Note in this last example the presence of the final 0; the configuration $\overline{\overline{=}11}$ is an instance of the non-halting loop of the cycle without the permutator. We see also the use of the over-bar notation. This notation is essential to (a) eliminating the redundant configurations, (b) recognising the essential replica nature of non-halting configurations. At each pass through the permutator the program increases the number of instances of the repeat unit by 1, but the notation eliminates this information under the category of "finite but indeterminate" and the non-halting behaviour of $\overline{=}0\ \overline{1_211}00\overline{=}$ is recognised.

A non-halting loop of the B5 champion

test input: $S_0=\overline{=}0\ \overline{1_211}00\overline{=}$	
$\overline{=}0\ \overline{1_211}00\overline{=}$	new permutation
$\overline{=}0\ 11_41_10_2\overline{=}0$	exit cycle
$\overline{=}0\ \overline{111}1_3\overline{=}0$	
$\overline{=}00_3\overline{11110}\overline{=}$	
$\overline{=}01_2\overline{11110}\overline{=}$	new permutation
$\overline{=}011_41_11_21_40_10_0\overline{=}0$	
$\overline{=}01\overline{1111}01_1\overline{=}0$	
$\overline{=}01\overline{1111}010_2\overline{=}0$	exit cycle
$\overline{=}01\overline{1111}011_3\overline{=}0$	
$\overline{=}01\overline{111}10_3110\overline{=}$	
$\overline{=}01\overline{111}11_3110\overline{=}$	
$\overline{=}00_3\overline{11111111}0\overline{=}$	
$\overline{=}01_2\overline{11111111}0\overline{=}$	
$\overline{=}01_2\overline{1100}\overline{=}$	eliminate redundant cycles
LOOP	

It is a conjecture that this is the only additional non-halting LOOP of the B5 Champion – a loop that is not already a LOOP of the previous cycle. If valid this conjecture would solve the complete criterion for the B5 Champion.

6. Philosophical and cultural considerations

There is a philosophical and cultural background to the Halting problem which is the mainspring of its interest. Thesis: all mathematical reasoning (human or machine) is a species of computation. Antithesis: it is not.

See also Appendix 3 – the Church-Turing thesis.

The apparent impossibility of a solution to the Halting Problem

The proof that it is not possible to solve the Halting Problem states merely that if one machine *A* solves the halting problem for another machine *B*, then machine *A* has many more states than machine *B*. From this we may infer that it is impossible to build a universal machine that would solve the halting problems of all machines, including its own. This proof does not refer to human reasoning, so says nothing directly about it. The possibility of *knowing* that a halting problem is solvable, yet not being practically able to solve that halting problem is not denied by this argument. Computers may exist that compute the problem for machines much smaller than themselves. It is a moot question whether the procedures described in this paper constitute an algorithm. Certainly, the steps described here look like an iterative procedure in which also algorithms are involved. But then, so do many other computations in mathematics – iterative procedures that take the output of one instance of an algorithm and use it as the input of another – computing the Fibonacci series for instance, or its sum. But all such computations stumble on the apparently forgotten fact that any computer implementing them must be finite, however large. Hence, there will always be a Fibonacci number so big that no machine has ever computed it. What is really expressed in an algorithm and an associated iteration is the *knowledge* that the algorithm and iteration will always work. It is not that the actually infinite sequence of all Fibonacci numbers has ever been computed – something which is impossible – but that we *know* that a potentially infinite number of them could be computed, in the sense that given two last Fibonacci numbers, we could compute one more. In this the Halting Problem is no exception.

Is the Halting Problem really so different?

What gives the impression that the Halting Problem is different is merely the fact that we hit the buffers of our ability to compute/solve the halting problem for a given machine. Already for five-state machines there are said to be 98 hold-outs – machines that cannot be solved. However, it is a claim here that they have now all been shown to be halting. Already by the five- and six-state machines we are reaching problems too large for practical solution – subject to the caveat that even these difficulties may have been exaggerated. But in principle there is no distinction between the situation for Turing Machines and the situation for the application of algorithms in general. All machines are finite, even if it is true that it is always possible in theory, given one machine, to build a larger one.

The existing impossibility proof is *not really* an impossibility proof. *What would an impossibility proof for the Halting Problem look like?* Such a proof would have to show that for some particular number *N* mathematical induction breaks down. We require an argument that for at least one machine of finite *N* states there is no possibility of determining whether or not it halts. There must be a specific counter-example produced.

Where mathematical induction breaks down

Such a situation in general can obtain in mathematics. For example, there are algorithms providing solutions to quadratic, cubic and quartic equations; but it has been shown that the quintic is insoluble. Hence any attempt to provide a general solution to equations of all degree by mathematical induction would break down for degree 5.

Empirical versus non-empirical methods

Boolos and Jeffrey [1980] proposed an empirical approach to the Halting Problem. (*See quotation.*) It is an error to apply an empirical procedure alone to this problem. What happens to a Turing machine is wholly determined by the states of the machines, its program and the configuration of the tape. It is not an empirical question but a question of logic and combinatorics.

In practice, a mixture of empirical and non-empirical – number theoretic – approaches is appropriate. As it happens, as soon as one devises an algorithm to screen for certain types of machine, then the application of that algorithm is a process that combines both the empirical and the non-empirical. The screening process is empirical – the *knowledge* that the screening process classifies some machines as halters and non-halters is not. When we make a trace by running the machine from standard configuration, then that is an empirical method. But, perhaps, it is a starting point for an investigation. Human reason gets involved, and starts to say to itself, *now what really is going on here? Can I make sense of this?* It is to be observed that hitherto, when computers were run, only one tape configuration, the standard input of a blank tape starting in the first state, had been considered. The true nature of a Turning machine to compute a function upon a finite configuration of the tape given any starting state would appear not to have been recognised until now.

The synthetic character of mathematical induction

If all mathematical reasoning is a species of algorithm, it follows that mathematical induction is also a species of algorithm. Here we have not assumed at the outset that mathematical induction is or is not one or the other, but we have employed it in a proof that the Halting Problem can be solved. If the conclusion is valid, it therefore follows that mathematical induction cannot be a species of computation, and that the thesis that it is a species of computation is refuted. It seems that computability is a sub-branch of number theory, not the other way around.

Accepting that the Halting Problem is solved in the way given in this paper, we may ask what it is about mathematical induction that makes it non-computational. The early twentieth century saw the rise of mathematical logic and set theory together with the claim that these were the foundation of all mathematics.

The empirical approach to the Halting Problem

The question is, 'Why isn't p computable in some intuitive sense?' After all, there are only finitely many different graphs of *n*-state machines if we don't trouble to number any of the nodes except for node 1, the starting node. Then for each *n* we can (in imagination), anyway) set all the n-state machines going, starting in state 1 on a blank tape, and await developments. As time passes, one or another of the machines may halt, at which point we can see whether it is scanning the leftmost of an unbroken string of 1s on an otherwise blank tape. If it does halt in that standard position, we find its productivity by counting the number of 1s in the string, but if it halts in a non-standard position we know that its productivity is 0. But there is a catch: some of the *n*-state machines may never halt, so that no matter how long we wait, it may be that the productivity of one or more of the n-state machines cannot be determined in the way we have just sketched. Those machines will have productivity 0 because they never halt.

Boolos and Jeffrey [1980] p. 40

Claims made for first-order logic and set-theory

Most logicians (though perhaps not most mathematicians) are convinced that all correct proofs in mathematics could, with enough effort, be translated into formal proofs of first-order logic.

Wolf [2005] p.29

ZFC is a remarkable first-order theory. All of the results of contemporary mathematics can be expressed and proved within *ZFC*, with at most a handful of esoteric exceptions. Thus it provides the main support for the formalist position regarding the formalizability of mathematics. In fact, logicians tend to think of *ZFC* and mathematics as practically synonymous.

Wolf [2005] p.36

Set theory is the foundation of mathematics. All mathematical concepts are defined in terms of the primitive notions of set and membership. In axiomatic set theory we formulate a few simple axioms about these primitive notions in an attempt to capture the basic "obviously true" set-theoretic principles. From such axioms, all known mathematics may be derived. However, there are some questions which the axioms fail to settle, and that failure is the subject of this book.

Kunen [1980] p.xi.

The possibility also that set theory is not a form of computation would seem to have been laid aside. Already in 1914 Henri Poincaré expressed his critical opposition to the rising movement in favour of symbolic logic. In his essay *Mathematics and Logic*, he stated that the principle of complete induction "appeared to me at once necessary to the mathematician and irreducible to logic." (Poincaré [1996] p. 148.) This claim made by Poincaré is upheld by the conclusions reached here. Therefore, it is important to examine afresh the character of mathematical induction.

In the conclusion of any argument from mathematical induction the expression "all numbers" refers to a potentially infinite collection – the possibility of adding one to a given number never reaching an end. This idea of the *unendingness* of the operation of adding 1 is not found directly in the act of adding 1. Taking a number and adding 1 to it does not *mean* a collection; the meaning of the potential infinite is not expressed by it. Nor is it expressed by the idea of a starting place in the sequence of natural numbers. Mathematical induction simply replicates in an argument the movement contained in the idea of starting somewhere, 0, and adding one more each time, and so on and so on ... *endlessly*. Induction ties by algebra or a species of argument a result to that 0, and a result to that act of adding one, but it rests on the underlying notion of a potential infinity. But that notion is not found in the meaning of the operation of adding one, which means only adding one; it is implied by that operation but not meant by it.

Therefore, mathematical induction is a species of synthetic reasoning. By its means the human mind steps from two premises that have no connection in *meaning* with a conclusion that expresses a *new meaning*, one that encompasses the concept of (potential) infinity. This inference is also necessary. Hence we have a pattern of inference that is both necessary and synthetic. It is synthetic because the conclusion expresses more information than is contained in the premises. How it can also be necessary is a metaphysical problem not undertaken here.

Poincaré's objection

... syllogistic reasoning remains incapable of adding anything to the data given in it; these data reduce themselves to a few given axioms, and we should find nothing else in the conclusions.

... mathematical reasoning has of itself a sort of creative virtue and consequently differs from the syllogism.

The difference must even be profound. We shall not, for example, find the key to the mystery in the frequent use of that rule according to which one and the same uniform operation applied to two equal numbers will give identical results.

All these modes of reasoning, whether or not they be reducible to the syllogism properly so called, retain the analytic character, and just because of that are powerless."

Quoted in Detlefsen [1990], and p.291 of Jacquette [2002]. The original is in Poincaré, *The Value of Science* (1905) in *The Foundations of Science*, ed., and trans. G. Halsted, The Science Press, 1946.

Mathematical Induction

Particular Result: the result is true for $k = 0$ (or for some other starting value).

Induction Step: if the result is true for the number *k* then the result is true for $1 : L$.

Conclusion: the result is true for all numbers *n* (or for all *n* greater than the starting value).

Knowing and computing

From the phenomenological point-of-view human beings are aware that they think, hold notions, use concepts and understand meanings. We may hold it as a possibility that all such self-conscious activities have their ground in some underlying mechanism, say, in the mechanical operations of the brain, and that such a mechanism runs as a species of algorithm would upon a computer. But such metaphysics is not given directly to phenomenological intuition. A person seeks to establish truth and knowledge, and mathematical induction is one means by which such a person seeks these things when considering number. Therefore, prima facie, there is no reason at all to suppose that *knowing* and *computing* are one and the same, and that arguments used by human reason to establish to reason that such-and-such a statement is *known to be true* is a species of algorithm. The cultural viewpoint that has led to the belief that mathematical induction is a species of computation no doubt runs throughout the history of mathematical logic, of set theory and of computing science in the twentieth century and into our own, but if mathematical induction does establish results known to be true, and also known not to be algorithms, then indeed we have no reason to suppose that knowing in general is also a species of computation.

Appendix 1: The halting problem is not effectively computable

The **halting problem** is the problem of designing an effective procedure for identifying Turing machines which never halt, once started in their lowest-numbered states on blank tapes.

From Boolos and Jeffrey

We confine our attention to Turing machines which read and write only two symbols *B* and 1. Any such machine *M* can be thought of as computing some total or partial function *f* from positive integers to positive integers as follows: To discover the value (if any) which *f* assigns to the argument *n*, start *M* in its lowest-numbered state, scanning the leftmost of a block of *n* 1s on an otherwise blank tape. If *M* eventually halts in a *standard configuration*, i.e. scanning the leftmost of a block of 1s on an otherwise blank tape, $f(n)$ is the number of 1s in that block. But if *M* never halts, or halts in some nonstandard configuration, $f(n)$ is undefined.

Boolos and Jeffrey [1980] p. 34.

Boolos and Jeffrey use the symbol *B* where we in this text use the symbol 0.

Productivity

Let *T* be a Turing machine of *n* states using only the symbols 0 and 1. Initially *T* scans only a blank tape. The machine *T* either halts in "standard configuration" scanning the leftmost of an unbroken string of 1s on it otherwise blank tape, or it does not. If it does not it may either not halt at all, or halt scanning some other configuration. The **productivity** of *T* is defined to be:

$$p(T) = \begin{cases} \text{the length of the string that } T \text{ scans if it halts in standard configuration} \\ 0 \text{ otherwise} \end{cases}$$

This is a function defined for each Turing machine. From this we may derive another function, $p(n)$ which is defined to be the productivity of the most productive *n*-state Turing machine.

The **Busy Beaver** is a Turing machine which computes the function *p*. The machine must read and write only the symbols 0 and 1. The **Busy Beaver Problem** is the problem of designing such a machine.

Properties of productivity (Boolos and Jeffrey [1980] p. 35.)

(1) $p(1) = 1$. There are 25 different 1 state machines corresponding to 25 flow graphs of one node. By examination of cases we can show that some of these do not halt and of those that do the maximum productivity is 1.

(2) $p(47) \geq 100$. It is possible to construct a machine of 47 states with productivity 100. Machines with 47 states do exist with greater productivity but this is not what is asked for in this result.

(3) $p(n+1) > p(n)$. We can always add one more state to a given machine of *n* states that adds another 1 and has productivity $p(n)+1$.

(4) $p(n+11) \geq 2n$. This is another instance of a concrete *n* state machine that has productivity $p(n+11) = 2n$.

Proposition

The busy beaver problem is not effectively computable: there is no machine that can compute $p(n)$.

Proof

Assume that such a machine exists, and let it have *k* states. It would start scanning the leftmost of *n* 1s and finish scanning $p(n)$. Call this a "Busy Beaver Machine" (BB). Then there exists a machine that comprises a machine that writes *n* 1s followed by two copies of the BB machine.

Write *n* 1s $\longrightarrow$ BB $\longrightarrow$ BB

This machine has $n+2k$ states and its productivity is $p(p(n))$. Then, assuming that BB exists we have $p(n+2k) \geq p(p(n))$. From the result $p(n+1) > p(n)$ it follows $p(i) > p(j)$ if $i > j$, from which it follows by contraposition that if $j \geq i$ then $p(j) \geq p(i)$. Let $j = n+2k$ and $i = p(n)$, then this gives with $p(n+2k) \geq p(p(n))$ the conclusion $n+2k \geq p(n)$. This remains true if *n* is increased by 11 to give $n+11+2k \geq p(n+11)$. But then $p(n+11) \geq 2n$ entails $n+11+2k \geq 2n, 11+2k \geq n$, which is true for all *n*. But putting $n = 12+2k$ we obtain a contradiction. Therefore, BB does not exist.

Proposition (Boolos and Jeffrey [1980] p. 30). If the halting problem is solvable, then the function *p* is computable.

Proof

Suppose we have "a systematic procedure for identifying non-halters ... If we had such a procedure we could apply it to the graphs of all the *n*-state machines while those machines are working (having been started in their lowest-numbered states, on blank tapes). After some finite period of time, each machine will either have halted or have been identified as a non-halter, so that for each *n*, there would be some period of time after which we would know the productivity of every *n*-state machine. For each *n*, we would then be able to compute $p(n)$: the function would be computable in an intuitive sense if there were a systematic procedure for identifying non-halters."

Boolos and Jeffrey [1980] p. 30.

Appendix 2: Turing machines

A Turing machine is a device for performing a computation. It may be visualised as a car moving along a *track* or *tape* that is divided into segments and is potentially infinite in length. The segments contain *symbols*. A result shows that only two different symbols are required. These are designated 0 and 1. The car scans one segment of the track at a time. The car has a *program* that instructs it what to do when it scans a symbol on the track. The instruction depends on (a) the *state* the car is in when it scans the symbol and (b) on the symbol. These two pieces of information are sufficient to instruct the car to perform an *action* and to tell it which state to go to next. This information is encoded in a *quadruple*, so the mechanical description of the machine is an implementation only of an abstract structure.

Definition, quadruple

A *quadruple* is an ordered 4-tuple of the form $\left(q_i\ \sigma_i\ \sigma_i'\ q_i'\right)$ where q_i, q_i' are states, $\sigma_i \in \{0,1\}$ and σ_i' is an action, $\sigma_i' \in \{0,1,L,R\}$. A quadruple encodes an *instruction* to the Turing machine.

Proposition (Boolos and Jeffrey [1980] p. 30)
Any function from positive integers to positive integers which is Turing computable is Turing computable in monadic notation by a Turing machine which uses only the symbols 0 and 1.

Definition, action

An action is one of the following.

$0:1$ Change the scanned symbol on the tape to 1. Similarly, for $1:0$.

$0:L$ On scanning the symbol 0 move left. Likewise, $1:L$.

$0:R$ On scanning the symbol 0 move right. Likewise, $1:R$.

Example: $(2\,1\,L\,5)$ is read, "When in state 2 scanning 1 move left and enter state 5".

Definition, program, Turing machine

A *program* is a set of quadruples. A program is also called a *Turing machine*.

This summary is as brief as it may be. However, using this and the examples of the text, a reader can be fully acquainted with the whole topic in very little time. For a more extensive introduction see Boolos and Jeffrey [1980].

Flow graphs

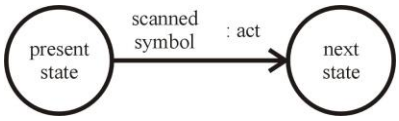


Figure 24. Flow graph template

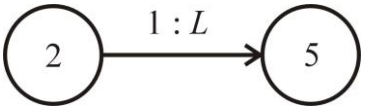


Figure 25. Example of a flow graph

When in state 2 scanning 1 move left and enter state 5. This corresponds to the action: $(2\,1\,L\,5)$.

Appendix 3: The Church-Turing thesis

There are several distinct mathematical descriptions of effective computation.

1. Turing's analysis based on Turing machines.
2. The Gödel-Herbrand analysis based on recursive functions.
3. An analysis in terms of Abacus machines.
4. Church's analysis based on his lambda calculus.
5. Markov algorithms.
6. Post systems.

Definition, The Church-Turing Thesis

The *Church-Turing thesis* is the claim: -

1. The six analyses of what is effectively computable given above are all formally equivalent. Call this equivalence class the *class of all Turing computable functions*.

2. All effectively computable functions belong to the class of all Turing computable functions and conversely.

This definition equates a mathematically precise notion of *Turing computable function* to the intuitive notion of *effectively computable function*. (This point has been frequently expressed in the literature.) This gives the thesis a very peculiar status. Part (1) is a mathematical theorem that has been formally proven, and is thereby not contentious. Part (2) is said to be an empirical thesis and to be supported by "evidence" of an observational nature rather than mathematical.

It is a consequence of this empirical thesis that it is expected that should anyone come up with a new analysis of what an effectively computable function is, then it will be proven to belong to the class of all Turing computable functions. In the Church-Turing thesis an extension (the class of all Turing computable functions) is equated with an intension (all effectively computable functions). Notwithstanding the empirical character of part 2 of the definition above, we take it as a *definition* of what *effectively computable* means. To demonstrate the pertinence of this, consider two quotations from the seminal paper *Computing Machinery and Intelligence* by Turing (Turing [1950]).

The Church-Turing Thesis in the literature

1. "Church's thesis: all computable functions are Turing computable."

Boolos and Jeffrey [1980] p. 54.

2. "In around 1936, several mathematicians (Alonzo Church, Stephen Kleene, Emil Post, and Alan Turing) independently proposed precise definitions of the notion of effective procedure. Even though their definitions were very different from each other conceptually, it was proved that these definitions are all equivalent, in the sense that every function that can be computed under one definition can be computed using the others as well."

Wolf [2005] p.96

3. CTT: "... no human computer, or machine that mimics a human computer, can out-compute a universal Turing machine"

p.10 of Abramson in Olsweski et al. [2006] who is quoting from Jack Copeland, 2002a, p.67

The equivalence of these definitions, as well as our substantial experience with computer languages and programs, provide strong empirical evidence that these definitions do in fact correctly represent the intuitive notion of an effective procedure, or mechanical computation. There is no way to prove this, but it is a standard view as a sort of informal axiom, called *Church's thesis.*.

Wolf [2005] p.96.

... we only permit digital computers to take part in our game. [p.7]

The idea behind digital computers may be explained by saying that these machines are intended to carry out any operations which could be done by a human computer. The human computer is supposed to be following fixed rules; he has no authority to deviate from them in any detail. We may suppose that these rules are supplied in a book, which is altered whenever he is put on to a new job. He has also an unlimited supply of paper on which he does his calculations. He may also do his multiplications and additions on a "desk machine", but this is not important. [p.8]

Computing Machinery and Intelligence by Turing (Turing [1950]).

These quotations provide a description of what Turing means by "effective computation" and *following fixed rules without deviation* lies at the core of this. But even this is ambiguous, and the implicit notion of what we shall take for a rule serves to convert the discussion into a species of definition.

Turing also places emphasis on digital computers. If we are restricting our attention to digital computers, we have certainly very good evidence of the mathematical variety for accepting the Church-Turing thesis. This concerns the structure of a digital machine that is made solely of binary switches. The binary switches restrict the digital computer theoretically to a very specific domain, whose topology can be precisely delimitated, at least in the sense of being given an upper bound as a Boolean algebra. Hence, we adopt the Church-Turing thesis as a *definition* of what is effectively computable, and as a theorem about digital computers and reject the claim that it is an empirical thesis.

Church –Turing Theorem

The six analyses (1) Turing's analysis based on Turing machines, (2) The Gödel-Herbrand analysis based on recursive functions, (3) Abacus machines, (4) Lambda calculus, (5) Markov algorithms, (6) Post systems are all formally equivalent. Call this equivalence class the *class of all Turing computable functions*.

Definition, Church-Turing thesis, V2

The *Church-Turing thesis V2* is the statement that any *effectively computable function* is defined to be a Turing computable function.

By defining effectively computable in terms of the equivalence class of Turing computable functions, it makes no direct claim as to whether *all mathematical proofs are Turing computable*: simply accepting this version Church-Turing thesis does not force one to accept that mathematical proofs are computable.

Appendix 4: Hold-out machines used as examples

Hold-out machine no. 60. Kellett [2005] B.9

Non-halting behavior. It is straightforward to demonstrate that no. 60 does not halt for standard configuration. It has only one exit at X4, where it halts on a 0. A backward trace by the method of exits constructs a tree with no branches and minimal depth.

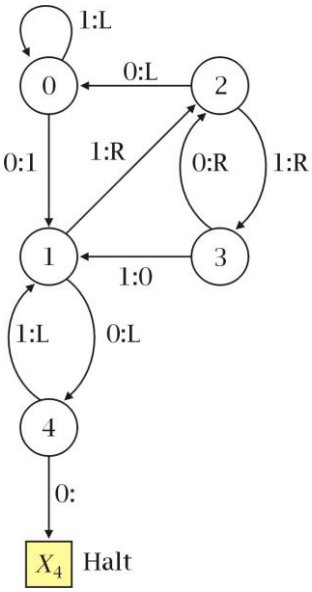


Figure 26. Hold-out no. 60.

Exit at X_4

0_4

00_1

01_3 0 : 1 001_4

1 : *R* 0010_1

0011_3 0 : 1 00101_4

001_21 1 : *R* $00\overline{1}010_1$

00_311 $00\overline{1}01_40$ $00\overline{1}011_3$

1 : *R* LOOP LOOP

There is no path backwards from this exit to the standard configuration. In fact, this machine loops—is non-halting—for all almost all inputs whatsoever.

Hold-out no. 54. Kellett [2005] A. 79.

We approach the solution to no. 54 (*figure 27*) by solving for a 4-state sub-machine first, obtained by removing state $Q2$. We denote this machine by $T4$ (*figure 28*).

T4 halts in state $Q1$ on both a 0 and a 1. This machine has a sub-routine: when we solve by the method of exits we find the program enters into an infinite loop defined by the sub-routine. This is represented as finite information using the over-bar notation. The loops bring the tree generated by the method of exits to termini; the tree is thus shown to be a finite tree.

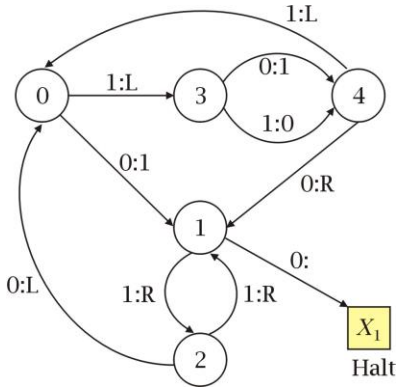


Figure 27. Hold-out no. 54

SUB-ROUTINE

1_0
11_4
10_3
101_0

$1_0 \rightarrow \overline{10}\{1_0, 11_4, 10_3$

Exit at X_1

0_1	1_1	
0_40	0_0	0_41
1_30	01_4	1_31
$1:L$	00_3	11_0
	001_0	LOOP
	LOOP	

$00\}1 \overline{10}\{1_0, 11_4, 10_3 \rightarrow H = 01_1\}1_1 \overline{11}\{1, 11, 10$

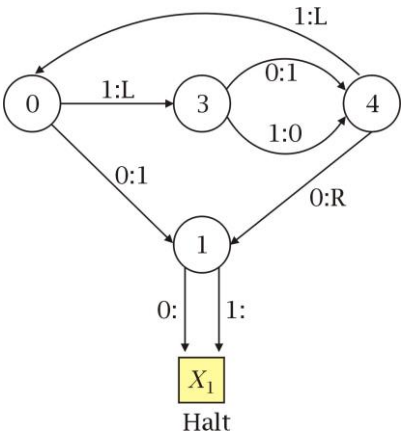


Figure 28. T4 for no. 54

Complete Criterion for *T4*.

S_0		S_3		S_4	
0_0	$H = 1_1$	1_30	$H = 00_1$	0_40	$H = 00_1$
001_0	$H = 1_111$	1_31	$H = 01_1$	01_4	$H = 1_11$
$00\overline{10}1_0$	$H = 1_1\overline{111}\,110_3$	$H = 01_11$	0_41	$H = 01_1$	
S_1		$00\}1 \overline{10}10_3$	$H = 01_1\}1_1 \overline{1111}$	$00\}1 \overline{10}11_4$	$H = 01_1\}1_1 \overline{1111}$
0_1	$H = 0_1$				

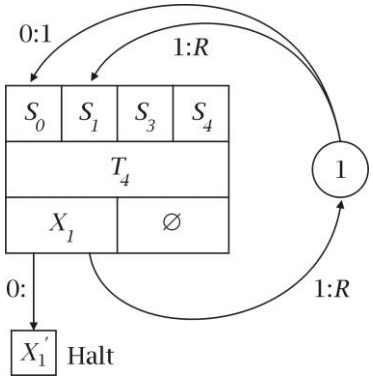


Figure 29. T5

We also solve for *T5* (*figure 29*) by the method of exits. To solve the halting problem for *T5* it is not necessary to write out the complete criterion, but as we wish to illustrate the iterative process that lies behind the inductive proof that the Halting Problem is soluble, we do so.

Method of exits for *T5*

Exit at X_1

$X_1 = 0$

$S_1 = 0$

1_20

1_110

0_010	11_010	01_110 R	1_2110		
$0 : L$	$11_0\overline{1}110$	11_010	1_11110		
		$0 : L$	0_01110	001_010	11_01110
			$0 : L$	$1 : R$	1_211110
					$1_1\overline{1}110$
					LOOP

Halting configurations

0_1	1_20
1_110	11_010
0_010	1_2110
1_1	$\overline{1}110$
11_0	
0_010	
001_01	

Complete Criterion for *T5*

S_0		S_1	
0_010	$H = 110_1$	0_1	$H = 0_1$
11_010	$H = 0110_1$	1_110	$H = 110_1$
$11_0\overline{1}110$	$H = 01\overline{1}110_1$		
S_2		S_3	
1_20	$H = 10_1$	1_30	$H = 00_1$
1_2110	$H = 1110_1$		
$1_21\overline{1}110$	$H = 1\overline{1}110_1$		
S_4			
00_4	$H = 00_1$		

From the complete criterion we see that the starting standard configuration is not among the halting configurations. Hence, *T5* = no. 54 does not halt for standard configuration.

Hold-out machine no. 20. Kellett [2005] B.4

This machine (*figure 27*) has one exit at X4. There are four inputs to state Q1, so direct solution of this machine, though possible, would lead to a tree with very many branches. In addition, there are cycles of periods of 1, 2, 3 and 4 states; the double cycles between states Q0 and Q1 and between Q1 and Q2 are particularly tricky (see below). The writing of the complete criterion for this machine is a difficult problem.

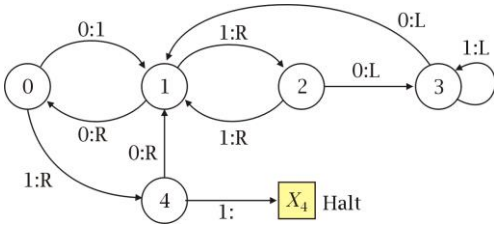


Figure 27. Hold-out no. 20

Direct proof of non-halting behaviour

We can directly prove that this machine is non-halting. Firstly, consider the machine T4* (T4 star *figure 27*) obtained by deleting state Q3 from no. 20. Then we adjoin state Q3 to T4* to make no.20. The point of this construction is that in T4* there are only moves to the right on the tape; thus in no. 20 any moves to the left are effected in the loop involving Q3. Supposing no. 20 halts (exits) at X4. Then by tracing back to state Q1 it must have a tape configuration at exit thus:

011_4 01_01 0_111

But this last configuration is not a candidate for a loop backwards through Q3, because of the tape contradiction with the move 0:*L*. Any other candidate can only have other symbols to the left of the 011, because T4* has only right moves. To loop backwards through Q3 the machine must scan a 0 and then move left into state Q1, so for the machine to reach the final state having passed through Q3 and exit it must have a tape configuration of the form:

$....\boxed{1}011$ $\boxed{\ }0_311$

Here the box indicates an unspecified symbol. Then working backwards through the loop with Q3 we obtain:

0_311 011_3 $0:L$

We obtain another tape contradiction: we cannot have exited from X2 scanning a 0. There can be no path from the standard configuration at S0 to X2, Q3, S1 and finally to X4. The 011 combination needed to make machine no. 20 halt must be on the tape at the outset; it cannot have been written to the tape by iteration of the instruction 0:1 at state Q0. Therefore no. 20 does not halt for standard configuration. Another observation is that the only write instruction in no. 20 is the 0:1 at state Q0. Therefore, the machine could never erase a 1 once it has been placed onto the tape. Hence, the machine can never place a 0 next to the final 1 at the supposed exit at X4. Again, this proves that the 1 on which X4 halts must exist on the tape at the outset.

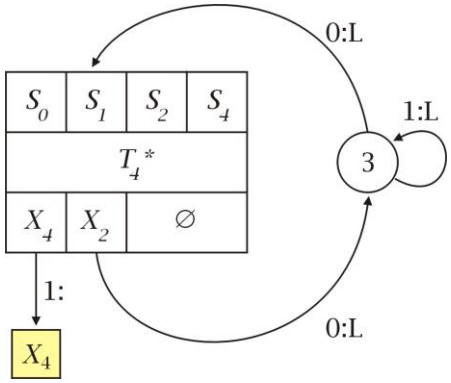


Figure 28. T4 star

A tape contradiction

Exit at $X4$	Exit from $Q3$
$0_1\boxed{1_3}1$	$\boxed{1}0_311$
required	required

Tape contradiction

Halting behaviour is a function of the initial tape configuration. A complete criterion for an *n*-state machine *Tn* is a specification of those tape configurations for any inputs to the machine that cause that machine to halt, specifying the exit at which it halts, and giving completely the tape configuration at any given exit.

If our purpose was only to demonstrate that no. 20 did not halt, we would be done. But we also seek to illustrate the inductive method whereby we demonstrate by mathematical induction that the halting problem for all Turing machines is soluble. The essence of the inductive argument is that, given a complete criterion for an *n* state machine, T*n* then a complete criterion for the *n* + 1 state machine made by adding one more state to T*n* is also determined. Since complete criteria can be written for all 1-state machines, this inductive step proves that a complete criterion can be written for all Turing machines, and thereby that the halting problem is solvable for all Turing machines.

It is intuitively clear that the inductive argument is valid, because ultimately what happens inside a Turing machine is wholly determined by the initial tape configuration and how the machine responds to it. It is a finite function. If a machine involved no loops whatsoever, this result would be obvious. However, while the loops themselves present difficulties, these do not prevent the writing of a complete criterion. This is because each loop involves a finite sequence of instructions, and hence any tape configuration that leads to non-halting behaviour can be characterised by a finite description, in the same way numbers with recurring decimal expansions may be characterised. If a machine does not halt it is because of some infinite repetition of a portion of the tape configuration.

Writing a complete criterion for a given Turing machine could be a problem not worth the effort to solve. As the number of states of a Turing machine increases, the permutations of repeat symbols can grow so large as to exceed human patience. But this applies to all inductive proofs whatsoever. To prove inductively that a result holds for all numbers, is not to say that the result has been actually applied to all numbers whatsoever – that is impossible.

We begin by examining a three-state sub-machine of no. 20, designated T3.

T3 has exits at X0 and X2 (*figure 29*). It involves two 2-cycles: one between states Q0 and Q1; the other between states Q1 and Q2. Whether this machine enters an infinite cycle depends upon the tape configuration—one configuration causes the machine to enter the Q0—Q1 loop, the other the Q1—Q2 loop. Any permutation of such configurations will cause the machine to halt. The over bar represents a tape configuration repeated a finite but unspecified number of times, possibly zero times.

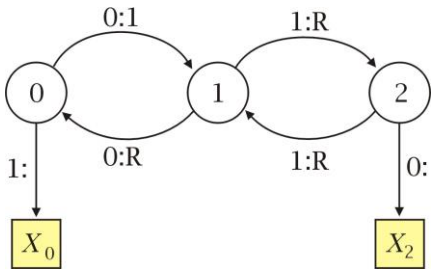


Figure 29. T3 for no. 20

initial		repeat		terminal	
$0_1 01$	$E = 011\boxed{1}$	$\overline{1_1 1}$	$E = \overline{11}\boxed{1}$	1_0	$H = \boxed{1_0}$
$1_1 1$	$E = 11\boxed{1}$	$\overline{0_1 01}$	$E = \overline{011}\boxed{1}$	$0_1 1$	$H = 0\boxed{1_0}$
1_2	$E = 11\boxed{1}$			0_2	$H = \boxed{0_2}$
$0_0 1$	$E = 11\boxed{1}$			$1_1 0$	$H = 1\boxed{0_2}$
				$1_2 10$	$H = 11\boxed{0_2}$
				$0_0 0$	$H = 1\boxed{0_2}$
				$0_1 00$	$H = 01\boxed{0_2}$

Steps through the repeat cycles of T3

$\overline{0_1 01}\square$	$\overline{1_1 1}\square$
$\overline{00_0 1}\square$	$\overline{11_2}\square$
$\overline{01_1 1}\square$	$\overline{11}\boxed{1}$
$\overline{011_2}\square$	
$\overline{011}\boxed{1}$	

When the machine moves through a configuration represented by the over bar, it moves through the symbol in the same way in which it would move through a single instance. Whether this machine halts at exits X0 or at X2 depends on a terminal sequence. Whenever this machine moves along the tape, it moves to the right, so the terminal sequence is the final end block to the right of any repeat sequences. Depending on what state the machine is in when it starts there is an initial configuration after which the machine enters the repeat sequences, then the terminal sequence and halts. For any other initial sequence, repeat sequence or terminal sequence, the machine does not halt.

Complete Criterion for T3

S_0		S_1		S_2	
1_0	$H = \boxed{1_0}$	$0_1 00$	$H = 01\boxed{0_2}$	0_2	$H = \boxed{0_2}$
$0_0 0$	$H = 0\boxed{1_0}$	$1_1 1\left\{\frac{\overline{11}}{001}\right\}01$	$H = 11\left\{\frac{\overline{11}}{011}\right\}0\boxed{1_0}$	$1_2 10$	$H = 11\boxed{0_2}$
$0_0 1\left\{\frac{\overline{11}}{001}\right\}01$	$H = 11\left\{\frac{\overline{11}}{011}\right\}0\boxed{1_0}$	$0_1 01\left\{\frac{\overline{11}}{001}\right\}01$	$H = 001\left\{\frac{\overline{11}}{011}\right\}0\boxed{1_0}$	$1_2\left\{\frac{\overline{11}}{001}\right\}01$	$H = 1\left\{\frac{\overline{11}}{011}\right\}0\boxed{1_0}$
$0_0 1\left\{\frac{\overline{11}}{001}\right\}10$	$H = 11\left\{\frac{\overline{11}}{011}\right\}1\boxed{0_2}$	$1_1 1\left\{\frac{\overline{11}}{001}\right\}10$	$H = 11\left\{\frac{\overline{11}}{011}\right\}1\boxed{0_2}$	$1_2\left\{\frac{\overline{11}}{001}\right\}10$	$H = 1\left\{\frac{\overline{11}}{011}\right\}1\boxed{0_2}$
		$0_1 01\left\{\frac{\overline{11}}{001}\right\}10$	$H = 001\left\{\frac{\overline{11}}{011}\right\}1\boxed{0_2}$		

Here $\left\{\frac{\overline{11}}{001}\right\}$ indicates any finite permutation of blocks of 11 and 001L: one is 11 11 001 11 001.

Because of the complexity of the problem it is also useful to reverse this table and correlate the outputs at the two exits X0 and X2 with their corresponding inputs at the source S1. Only S1 need be considered because when we construct the next machine "up", the T4 machine, we only have a feedback loop running through S1.

X_0		X_2	
Exit / output	Source / input	Exit / output	Source / input
$\boxed{1_0}$	1_0	$H = 01\boxed{0_2}$	0_100
			00_00
			01_10*
		$H = 1\boxed{0_2}$	1_10
$H = 11\left\{\frac{\overline{11}}{011}\right\}0\boxed{1_0}$	$1_11\left\{\frac{\overline{11}}{001}\right\}01$	$H = 11\left\{\frac{\overline{11}}{011}\right\}1\boxed{0_2}$	$1_11\left\{\frac{\overline{11}}{001}\right\}10$
$H = 001\left\{\frac{\overline{11}}{011}\right\}0\boxed{1_0}$	$1_2\left\{\frac{\overline{11}}{001}\right\}01$	$H = 001\left\{\frac{\overline{11}}{011}\right\}1\boxed{0_2}$	$0_101\left\{\frac{\overline{11}}{001}\right\}10$

* Sequence of outputs for $H = 01\boxed{0_2}$

0_100 00_00 01_10

In this table we include the potential input of 00_00 as this is the only way we can trace back an output to a standard configuration at S_0.

To make the T4 machine we may add either state Q3 or Q4 to T3 (*figure 30*). Here we add Q3 first. We can in fact show that T4 is halting for the standard configuration—also demonstrated by the straight printout of the tape configurations of No. 20 provided in Kellett's [2005] paper, where the machine exits on a 1 from state Q0 to state Q4 on line 39.

We solve by the method of exits. The trace is provided separately, together with notes. In the solution an impossible configuration is shown in red. These denote termini to the branches of the tree generated by the method of exits. Configurations representing termini to branches of the tree are marked in red by END. The standard configuration is almost at the deepest level of the tree, which is to be expected. The tree always terminates as the information expressed by the response of the Turing machine to any given tape configuration is always finite, although it may be large.

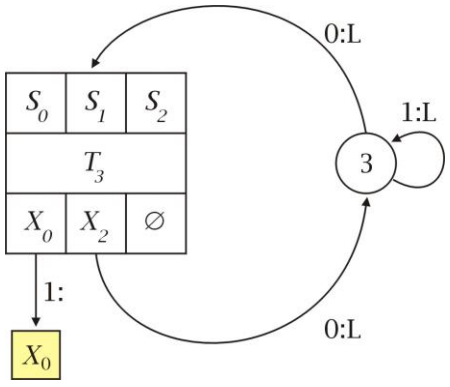


Figure 30. T4 for no. 20

From the tree generated by the method of exits, we could write the complete criterion for T4. To do so, we would need to fill in the missing intervening steps omitted by the "black box" method of viewing T3 as a machine computing outputs for given tape configurations at its inputs, and ignoring its internal states; these having already been considered at the earlier stage when the complete criterion for T3 was constructed. We omit this construction here; the method has been firmly established.

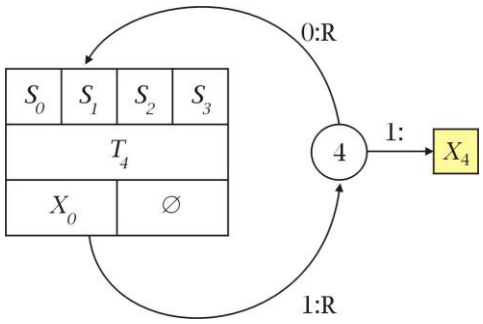


Figure 31. T5 for no. 20

Regarding the non-halting behaviour of T5 we can confirm the earlier result. By the method of exists we trace backwards.

Exit and halt at X_4
1_4
$1_0 1$
$0_1 11$

From the trace of T4 we see that the configuration $0_1 11$ (state Q1) never appears in it. Therefore, there is no path to this configuration from standard configuration, and T5 = no. 20 does not halt.

Holdout machine no. 20

T4 TRACE FOR THE METHOD OF EXITS.

Exit at X_0

1_0

$0_201\left\{\overline{\overline{11}\,\overline{001}}\right\}01$ $\quad$ $1_11\left\{\overline{\overline{11}\,\overline{001}}\right\}01$

$00_31\left\{\overline{\overline{11}\,\overline{001}}\right\}01$ $\quad$ $0:L$

$001_3\left\{\overline{\overline{11}\,\overline{001}}\right\}01$

$001\,\overline{11}0_21$ (1)

001110_21 (2)

00_201101 (3)

000_31101

0001_3101

00011_301

000110_21

00011_101

000110_31

0001101_3

00011010_2

0001101_20 $\quad$ 000110_200 (5)

00011010_3 $\quad$ 0001_21000 $\quad$ 000_201000 $\quad$ 0001100_30 (6)

000110100_2 $\quad$ $0:L$ $\quad$ 0000_31000 $\quad$ 00011000_2

END $\quad$ $[0000_21000]$ $\quad$ 00001_3000 $\quad$ END

$[000_201000]$ $\quad$ 000010_200 (7)

00000_0000 (8)

0000_20000

END

000_20101 (4)

0000_3101

00001_301

000010_21

00001_101 $\quad$ 0000_2001 (9)

000010_31 $\quad$ 00000_301

0000101_2 (10) $\quad$ 000000_21

END $\quad$ END

$001\overline{0_211}01$

$0010_211\overline{011}01$

$001_2011\overline{011}01$ $\quad$ $00_2001\overline{001}01$ (11)

$0010_311\overline{011}01$ $\quad$ $000_3011\overline{001}01$

$001011\overline{0_211}01$ $\quad$ $0000_211\overline{001}01$

LOOP $\quad$ END

$001\overline{011}\overline{0_211}01$

SIMILAR TO

$001\overline{0_211}01$ (12)

$001\overline{011}0_21$

$001\overline{011}01_3$

$001\overline{011}010_2$

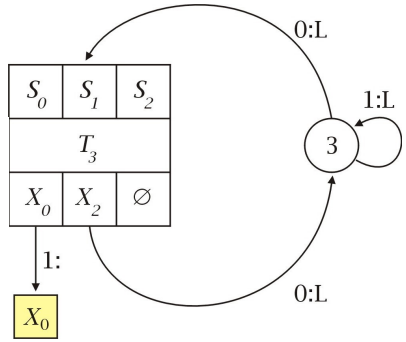
$001\overline{011}01_20$

$001\overline{011}010_3$

$001\overline{011}0100_2$

END

T4



(1) Only two possible configurations.

(2) To trace back further, there can be only one repeat block: $n = 1$.

(3) The configuration 001110_21 is the combination of the repeat block $\boxed{011}$ followed by the terminal block $\boxed{0_21}$. Hence traces back to 00_201101.

(4) The block 000110_21 has two possible backward traces: read either as $0001\boxed{10_2}1$ where $\boxed{10_2}$ is a terminal block or as $00\boxed{011}0_21$ where $\boxed{011}$ is a repeat block.

(5) $00011010_2 = 000110\boxed{10_2}$ or $00011\boxed{010_2}$.

(6) $000110_200 = 000\boxed{11}0_200$ or $00\boxed{011}0_200$; input from $0:L$ is also possible.

(7) $000010_200 = 000\boxed{010_2}00$ where $\boxed{010_2}$ is a terminal block.

(8) We have reached standard configuration. Hence, this machine halts for $S_0 = \overline{\overline{0}}$.

(9) $000010_21 = 0000\boxed{10_2}1$ or $000\boxed{010_2}1$, both terminal configurations.

(10) A tape contradiction, so the branch terminates.

(11) $0010_201\overline{001}01 = 00\boxed{10_2}01\overline{001}01$ or $0\boxed{010_2}01\overline{001}01$.

(12) The expression $\overline{0_211}$ may be taken to encode the information that the scanned symbol occurs at any one of the zeros in the finitely repeated sequence $\overline{0_211}$; thus $0_211011\ldots$ or $\ldots 0110_211011\ldots$ and so forth. Hence, we have entered a repeat cycle or loop.

Appendix 5. B5 Champion

The presence of the multiple cycles in the B5 Champion (*figure 32*) creates special difficulties. The cycle between Q2 and Q3 has the effect of interchanging a 1 for a 0 in any position in a block of 1s run through while in state Q3. For example, when in a backward trace (by the method of exits) we encounter the configuration $0_3 1110$. We are obliged to trace backwards as follows.

$$\overbrace{0_3 1110}$$

01_3110	011_310	0111_30
00_2110	010_210	0110_20
$1:R$	01_1010	011_100

Because the program then enters the other cycles the effect is that any number of permutations in a bloc of 1s may be halting. Hence the number of halting configurations for this machine is large, even though, in the final analysis, finite. The number of changes of the loop (the shift-number) defined by sub-machine Q0, Q1, Q2 and Q4 is at most 6. It is this fact that makes the problem finite.

One challenge is to show that by the method of exits the machine when starting in standard configuration does halt. To solve the problem we break the Champion into a sub-machine T4 plus one more state. Removing state Q0 leads to an excessive number of permutations. We remove state Q3. T4 (*figure 33*) has a sub-routine, and we apply the method of exits.

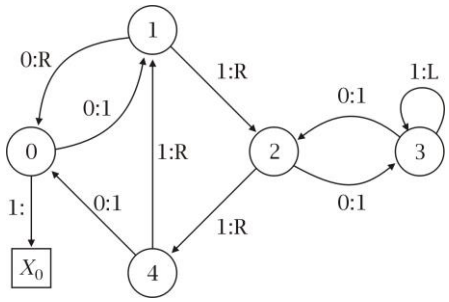


Figure 32. The B5 Champion

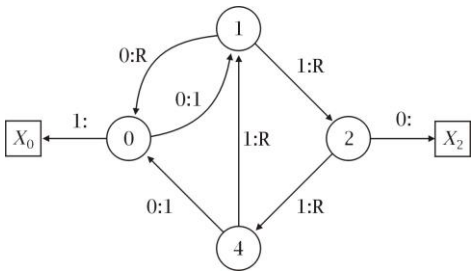


Figure 33. T4 for the B5 Champion

Sub-routine for T4

$$\begin{matrix} 1_1 \\ 1_41 \\ 1_211 \\ 1_1111 \end{matrix} \quad \begin{matrix} 0_0 \\ 0_10 \\ 1_400 \\ 1_2100 \\ 1_11100 \end{matrix} \quad 0:1 \quad 1_1 \rightarrow \begin{matrix} 1_1 \\ 1_41 \\ 1_211 \\ 0_0 \\ 0_10 \\ 1_400 \\ 1_2100 \end{matrix} \left\{ \begin{matrix} \overline{111} \\ \overline{1100} \end{matrix} \right\}$$

Exit at X_2

$$\begin{matrix} 0_2 \\ 1_10 \\ \text{LOOP} \end{matrix} \quad \left\{ \begin{matrix} 1_1 \\ 1_41 \\ 1_211 \\ 0_0 \\ 0_10 \\ 1_400 \\ 1_2100 \end{matrix} \right\} \left\{ \begin{matrix} \overline{111} \\ \overline{1100} \end{matrix} \right\} 0$$

Exit at X_0

$$\begin{matrix} 1_0 \\ 0_11 \\ 1_401 \\ 1_2101 \\ 1_11101 \\ \text{LOOP} \end{matrix} \quad \begin{matrix} 0_4 \\ 1_20 \\ 1_110 \\ \text{LOOP} \end{matrix} \quad \left\{ \begin{matrix} 1_1 \\ 1_41 \\ 1_211 \\ 0_0 \\ 0_10 \\ 1_400 \\ 1_2100 \end{matrix} \right\} \left\{ \begin{matrix} \overline{111} \\ \overline{1100} \end{matrix} \right\} \left\{ \begin{matrix} 1101 \\ 10 \end{matrix} \right.$$

T5 is obtained from T4 by addition of state Q3 In T5 there is only one exit at X0. As there is only one input to T4 at S2 only that part of the complete criterion for T4 is required. Our aim is only to show that T5 does halt in standard starting configuration; that is, the standard configuration is found as an element of the tree generated by the method of exits.

We observe that the permutations generated by these functions are considerable. Of special interest are the following instances of the schema from the method of exits.

S_2	
1_2100	$H = 1110_2$
1_21000	$H = 11010_2$
1_21100	$H = 1110_2$
$1_211\overline{11}10$	$H = 1\overline{11}110_2$

The interaction between these exits and the 1:L loop at state Q3 generates in the tree of exits for T5 many branches: the tree will be very wide. In the backward trace we see that the input to S2 must be a 1 because of the instruction 0:1 from the output of Q3, but the above schema indicate that within the machine T4 it is possible to convert a 0 at Q2 to a 1 at Q2. Hence, the complexity of this problem. It is helpful to reverse the information in the complete criterion and write out the transitions from S2 to X0 and X2 explicitly.

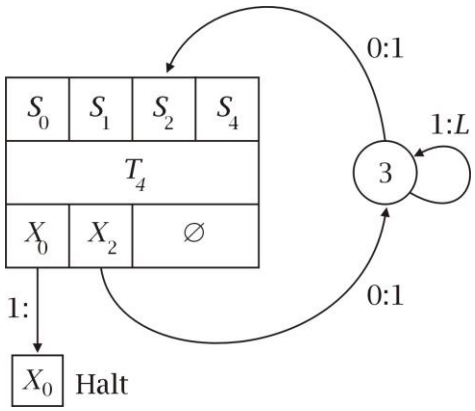


Figure 34. T5 for B5 Champion

Method of exits for T5

S_2	
1_2100	$H = 1110_2$
$1_211\left\{\frac{\overline{111}}{1100}\right\}0$	$H = 111\left\{\frac{\overline{111}}{1101}\right\}0_2$
1_2101	$H = 1101_0$
$1_211\left\{\frac{\overline{111}}{1100}\right\}\begin{cases}1101\\10\end{cases}$	$H = 111\left\{\frac{\overline{111}}{1101}\right\}\begin{cases}1101_0\\11_0\end{cases}$

$\underline{S_2 \to X_0}$

$1_20 \to 11_0$

$1_2101 \to 1101_0$

$$1_211\left\{\frac{\overline{111}}{1100}\right\}\begin{cases}1101\\10\end{cases} \to 111\left\{\frac{\overline{111}}{1101}\right\}\begin{cases}1101_0\\11_0\end{cases}$$

$\underline{1_0 \to 1_2}$

$11_0 \to 1_20$

$1101_0 \to 1_2101$

$$111\left\{\frac{\overline{111}}{1101}\right\}\begin{cases}1101_0\\11_0\end{cases} \to 1_211\left\{\frac{\overline{111}}{1100}\right\}\begin{cases}1101\\10\end{cases}$$

$\underline{S_2 \to X_2}$

$1_2000 \to 1110_2$

$$1_211\left\{\frac{\overline{111}}{1100}\right\}0 \to 111\left\{\frac{\overline{111}}{1101}\right\}0_2$$

$$1_2100\left\{\frac{\overline{111}}{1100}\right\}0 \to 1101\left\{\frac{\overline{111}}{1101}\right\}0_2$$

$\underline{0_2 \to 1_2}$

$1110_2 \to 1_2100$

$$111\left\{\frac{\overline{111}}{1101}\right\}0_2 \to 1_211\left\{\frac{\overline{111}}{1100}\right\}0$$

$$1101\left\{\frac{\overline{111}}{1101}\right\}0_2 \to 1_2100\left\{\frac{\overline{111}}{1100}\right\}0$$

Note: $11010_2 \to 1_21100$

We are now in a position to show how the method of exits would identify the branch in which the standard configuration appears. This is a very wide multiply branching tree, and we demonstrate only that (a) the standard configuration appears among the branches; (b) that the branch in which the standard configuration appears is finite. It ends on a CONTRADICTION.

Exit at X_0

1_0

11_0	$111\left\{\overline{\overline{111}\ \ 1101}\right\}1101_0$		$111\left\{\overline{\overline{111}\ \ 1101}\right\}11_0$		1101_0
1_20	$1_211\left\{\overline{\overline{111}\ \ 1100}\right\}1101$		$1_211\left\{\overline{\overline{111}\ \ 1100}\right\}10$		1_2101
ETC.	ETC.		$1_211\overline{111}110$	$1_211\overline{1100}110$	ETC.
			$0_311\overline{111}110$	ETC.	
$0_311\overline{1111}110$	$01_3\overline{1111}110$	$01_3\overline{111}110$	...	$011\overline{111}1_310$	$011\overline{111}1_30$
0 : 1	ETC.	ETC.		ETC.	$011\overline{111}10_20$
					$01\overline{1111}10_20$
					$01\overline{111}\boxed{1110_2}0$
					$01\overline{1111}_21100$
					$01\overline{111}0_31100$
			0 : 1	$01\overline{111}01_3100$	$01\overline{111}011_300$
				ETC.	$01\overline{111}1111011_300$
					$01\overline{111}\boxed{11010_2}00$
				$01_2\overline{111}100000$	$01\overline{111}_2100000$
				$00_3\overline{111}00000$	ETC.
0 : 1	$001_3\overline{111}00000$	$0011_3\overline{1111}00000$	...	$00\overline{1111}_300000$	
	ETC.	ETC.		$00\overline{1111}_300000$	001111_300000
				ETC.	001110_200000
					$001_211000000$
					$000_311000000$
			0 : 1	$00\,01_31000000$	$00011_3000000$
				ETC.	$00010_2000000$
					$0001_00000000$
					$\boxed{0000_00000000}$ SC
					CONTRADICTION
					1110_2 or 11010_2 required

We have demonstrated that the method of exits does lead to the standard configuration within a finite number of steps, and that the branch on which this configuration appears does end—it is finite.

In the preceding table, any branch that has not been investigated is shown in blue by the expression ETC. The contradictions are shown in red, and the branch leading to the standard starting configuration is boxed a marked SC in green. The difficulty created by the 1:L loop at state Q3 is demonstrated by this tree. Every time the machine is in state Q3 scanning a 0 to the right of which there is a block of 1s, it might have reached that state from any one of those 1s by an iterated move to the left (1:L). So this tree has many branches—that is, is very wide.

We have demonstrated that the method of exits does lead to the standard configuration within a finite number of steps, and that the branch on which this configuration appears does end—it is finite.

Permutator

Here we follow the method of the proof of the Complete Criterion Theorem given in the text. We add the permutator in two stages, firstly as an exit, and then as a loop back.

Following the method in the text, we add the permutator as a loop from the exit *X3* back to the input *S3* (*figure 34*) This time the part of the complete criterion for T5(i) that interests us concerns the inputs at *S3* as these only can be involved in loop back to *X3*.

S_3

0_3110	$H=1111_3$	0_30	$H=11_0$
$0_311\left\{\overline{\frac{111}{1100}}\right\}0$	$H=111\left\{\overline{\frac{111}{1101}}\right\}1_3$	0_3101	$H=1101_0$
$0_3100\left\{\overline{\frac{111}{1100}}\right\}0$	$H=1101\left\{\overline{\frac{111}{1101}}\right\}1_3$	$0_311\left\{\overline{\frac{111}{1100}}\right\}\begin{cases}1101\\10\end{cases}$	$H=111\left\{\overline{\frac{111}{1101}}\right\}\begin{cases}1101\\11_0\end{cases}$

If again we now apply the method of exits to T5(ii) (*figure 35*) we again obtain a tree which is very wide involving multiple configurations.

This machine has a cycle of period 9. To reach the exits from a starting configuration we must consider a configuration of 12 symbols. There are 49152 different configurations for the machine starting in state *Q0*. So we select some of these at random. To illustrate the reduction technique of the text when applied to the inputs of T5 (ii). We give four examples.

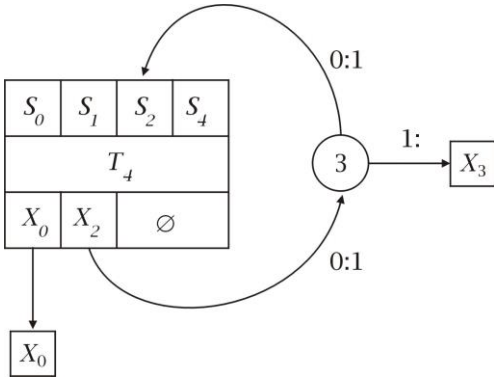


Figure 34. T5 (i)

Step 1 of the method in the proof of the Complete Criterion Theorem. The permutator is replaced by an exit.

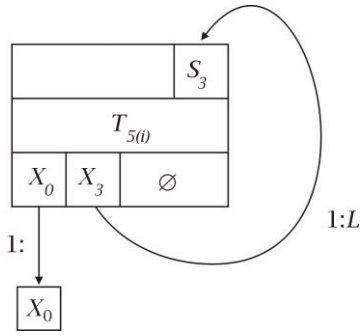


Figure 35 T5 (ii)

Step 2 of the method in the proof of the Complete Criterion Theorem. The permutator is added to T5 (i).

First Example

Test input: $S_0 = \overline{=}\,0\,0100_0111001010\,0\,\overline{=}$
random permutation of 12 symbols

1	$\overline{=}\,0\,0100_0111001010\,0\,\overline{=}$	permutation of 12 symbols
2	$\overline{=}\,0\,0101_1111001010\,0\,\overline{=}$	
3	$\overline{=}\,0\,01011_2\underline{1_4}\underline{1_1}0_201010\,0\,\overline{=}$	exit from cycle
4	$\overline{=}\,0\,01011111_301010\,0\,\overline{=}$	
5	$\overline{=}\,0\,010_311111010\,0\,\overline{=}$	
6	$\overline{=}\,0\,011_211111010\,0\,\overline{=}$	new permutation of 12 symbols
7	$\overline{=}\,0\,011_2\overline{11}01010\,0000\,\overline{=}$	remove redundant loop, new permutation
8	$\overline{=}\,0\,011_2\underline{1_4}\underline{1_1}0_21010\,0000\,\overline{=}$	exit from cycle
9	$\overline{=}\,0\,011\overline{111}_31010\,0000\,\overline{=}$	
10	$\overline{=}\,0\,0_3\overline{1111}1010\,0000\,\overline{=}$	
11	$\overline{=}\,0\,1_2\overline{111}1010\,0000\,\overline{=}$	new permutation of 12 symbols
12	$\overline{=}\,0\,1_2\overline{111}010\,0000000\,\overline{=}$	remove redundant loop, new permutation
13	$\overline{=}\,0\,1_2\underline{1_4}\underline{1_1}\,1_2010\,0000000\,\overline{=}$	
14	$\overline{=}\,0\,\overline{1111}010_4\,0000000\,\overline{=}$	
15	$\overline{=}\,0\,\overline{1111}011_0\,0000000\,\overline{=}$	exit and halt

During the execution of the method 23 permutations have been tested. Only two of these, the first and the last, start in state *Q0*. There are 245760 tape configurations in all to check.

Second example

Test input: $S_0 = \overline{\overline{0}100_0110000000\overline{0}}$
random permutation of 11 symbols

1	$\overline{\overline{0}}100_0110000000\overline{\overline{0}}$	permutation	23	$\overline{\overline{0}}1\overline{\overline{111}}1011111_3\overline{\overline{0}}$	
2	$\overline{\overline{0}}101_1110000000\overline{\overline{0}}$		24	$\overline{\overline{0}}1\overline{\overline{111}}10_3111110\overline{\overline{0}}$	
3	$\overline{\overline{0}}1011_21_40_1000000\overline{\overline{0}}$		25	$\overline{\overline{0}}1\overline{\overline{111}}111_2111110\overline{\overline{0}}$	new
4	$\overline{\overline{0}}1011100_000000\overline{\overline{0}}$		26	$\overline{\overline{0}}1\overline{\overline{111}}111_2\overline{11}0000\overline{\overline{0}}$	eliminate
5	$\overline{\overline{0}}1011101_100000\overline{\overline{0}}$		27	$\overline{\overline{0}}1\overline{\overline{111}}1111\ \overline{\overline{1}}_41_10_200\overline{\overline{0}}$	
6	$\overline{\overline{0}}10111010_20000\overline{\overline{0}}$	exit from cycle	28	$\overline{\overline{0}}1\overline{\overline{111}}1111\overline{\overline{11}}_300\overline{\overline{0}}$	exit from cycle
7	$\overline{\overline{0}}10111011_30000\overline{\overline{0}}$		29	$\overline{\overline{0}}0_3\ 1\overline{\overline{111}}1111\overline{\overline{11}}10\overline{\overline{0}}$	new
8	$\overline{\overline{0}}10111_3110000\overline{\overline{0}}$		30	$\overline{\overline{0}}1_2\ 1\overline{\overline{111}}0000000\overline{\overline{0}}$	eliminate
9	$\overline{\overline{0}}1011111_2110000\overline{\overline{0}}$	new	31	$\overline{\overline{0}}11_4\ \overline{1_11_21_4}\ 0_1000000\overline{\overline{0}}$	
10	$\overline{\overline{0}}10111111_41_10_200\overline{\overline{0}}$	exit from cycle	32	$\overline{\overline{0}}1\overline{\overline{111}}100_000000\overline{\overline{0}}$	
11	$\overline{\overline{0}}1011111113_00\overline{\overline{0}}$		33	$\overline{\overline{0}}1\overline{\overline{111}}01_100000\overline{\overline{0}}$	
12	$\overline{\overline{0}}10_31111113_00\overline{\overline{0}}$		34	$\overline{\overline{0}}1\overline{\overline{111}}010_20000\overline{\overline{0}}$	exit from cycle
13	$\overline{\overline{0}}11_21111111\overline{\overline{00}}\overline{\overline{0}}$	new	35	$\overline{\overline{0}}1\overline{\overline{111}}011_30000\overline{\overline{0}}$	
14	$\overline{\overline{0}}11_2\overline{\overline{111}}100000\overline{\overline{0}}$	eliminate	36	$\overline{\overline{0}}1\overline{\overline{111}}10_3110000\overline{\overline{0}}$	
15	$\overline{\overline{0}}11_21_41_1\ 1_21_40_100000\overline{\overline{0}}$		37	$\overline{\overline{0}}1\overline{\overline{111}}11_2110000\overline{\overline{0}}$	new
16	$\overline{\overline{0}}1\overline{\overline{111}}10100_000\overline{\overline{0}}$		38	$\overline{\overline{0}}1\overline{\overline{111}}111_41_10_2000\overline{\overline{0}}$	exit
17	$\overline{\overline{0}}1\overline{\overline{111}}10101_1000\overline{\overline{0}}$		39	$\overline{\overline{0}}1\overline{\overline{111}}1111_30000\overline{\overline{0}}$	
18	$\overline{\overline{0}}1\overline{\overline{111}}101010_200\overline{\overline{0}}$	exit from cycle	40	$\overline{\overline{0}}0_31\overline{\overline{111}}111111100\overline{\overline{0}}$	
19	$\overline{\overline{0}}1\overline{\overline{111}}101011_300\overline{\overline{0}}$		41	$\overline{\overline{0}}1_21\overline{\overline{111}}1111110000\overline{\overline{0}}$	
20	$\overline{\overline{0}}1\overline{\overline{111}}1010_31100\overline{\overline{0}}$		42	$\overline{\overline{0}}1_21\overline{\overline{111}}00000000\overline{\overline{0}}$	eliminate
21	$\overline{\overline{0}}1\overline{\overline{111}}1011_2110\overline{\overline{0}}$ new		43	$\overline{\overline{0}}11_41_11_2\ 0_40000000\overline{\overline{0}}$	
22	$\overline{\overline{0}}1\overline{\overline{111}}1011_21_41_1\ 0_2\overline{\overline{0}}$	exit	44	$\overline{\overline{0}}1\overline{\overline{1111}}_00000000\overline{\overline{0}}$	exit and halt

66 tape configurations tested. All halt.

Third example

Test input: $S_0 = \overline{=}\,0\,0000_0000000000$
standard configuration

1	$\overline{=}\,0\,0000_0000000000\,\overline{=}\,0$	standard configuration
2	$\overline{=}\,0\,0001_1000000000\,\overline{=}\,0$	
3	$\overline{=}\,0\,00010_2000000000\,\overline{=}\,0$	
4	$\overline{=}\,0\,00011_3000000000\,\overline{=}\,0$	exit cycle
5	$\overline{=}\,0\,000_311000000000\,\overline{=}\,0$	
6	$\overline{=}\,0\,001_2100000000\,\overline{=}\,0$	new permutation
7	$\overline{=}\,0\,0011_41_10_20000000\,\overline{=}\,0$	
8	$\overline{=}\,0\,001111_30000000\,\overline{=}\,0$	exit cycle
9	$\overline{=}\,0\,00_3111100000000\,\overline{=}\,0$	
10	$\overline{=}\,0\,01_2111100000000\,\overline{=}\,0$	new
11	$\overline{=}\,0\,011_41_11_21_40_10_000000\,\overline{=}\,0$	
12	$\overline{=}\,0\,01111101_100000\,\overline{=}\,0$	
13	$\overline{=}\,0\,011111010_20000\,\overline{=}\,0$	
14	$\overline{=}\,0\,011111011_30000\,\overline{=}\,0$	exit
15	$\overline{=}\,0\,0111110_3110000\,\overline{=}\,0$	
16	$\overline{=}\,0\,0111111_2110000\,\overline{=}\,0$	new
17	$\overline{=}\,0\,01111111_41_10_2000\,\overline{=}\,0$	
18	$\overline{=}\,0\,0111111111_3000\,\overline{=}\,0$	exit
19	$\overline{=}\,0\,0_31111111110000\,\overline{=}\,0$	
20	$\overline{=}\,0\,1_21111111110000\,\overline{=}\,0$	new
21	$\overline{=}\,0\,1_2\overline{11}1000\,\overline{=}\,0$	eliminate redundant
22	$\overline{=}\,0\,11_41_1\,1_20_4000\,\overline{=}\,0$	
23	$\overline{=}\,0\,\overline{111}11_1000\,\overline{=}\,0$	exit and halt

47 tape configurations tested. All halt.

Standard configuration halts.

Fourth example

test input: $S_0 = \overline{=}\,0\,\overline{1_211}0\,0\,\overline{=}\,0$

	$\overline{=}\,0\,\overline{1_211}0\,0\,\overline{=}\,0$	new permutation
	$\overline{=}\,0\,\overline{11_41_1}\,0_2\,\overline{=}\,0$	exit cycle
	$\overline{=}\,0\,\overline{111}\,1_3\,\overline{=}\,0$	
	$\overline{=}\,0\,0_3\,\overline{1111}\,\overline{=}\,0$	
	$\overline{=}\,0\,1_2\,\overline{1111}\,\overline{=}\,0$	new permutation
	$\overline{=}\,0\,11_41_11_2\,1_40_10_0\,\overline{=}\,0$	
	$\overline{=}\,0\,\overline{1111}101_1\,\overline{=}\,0$	
	$\overline{=}\,0\,\overline{1111}1010_2\,\overline{=}\,0$	exit cycle
	$\overline{=}\,0\,\overline{1111}1011_3\,\overline{=}\,0$	
	$\overline{=}\,0\,\overline{111}110_311\,\overline{=}\,0$	
	$\overline{=}\,0\,\overline{111}111_311\,\overline{=}\,0$	
	$\overline{=}\,0\,0_3\,\overline{1111}1111\,\overline{=}\,0$	
	$\overline{=}\,0\,1_2\,\overline{1111}1111\,\overline{=}\,0$	
	$\overline{=}\,0\,1_2\,\overline{11}\,0\,0\,\overline{=}\,0$	eliminate redundant cycles
	LOOP	

28 configurations (up to redundancy) tested.

LOOP = non-halting cycle identified.

All 28 configurations are non-halting.

References

Boolos, George and Jeffrey, Richard [1980]
Computability and Logic. Second edition. Cambridge University Press. Cambridge. (First edition, 1974)

Wolf, Robert S [2005]
A Tour through Mathematical Logic. The Mathematical Association of America.

Abramson, Darren [2006]
Church's Thesis and the Philosophy of Mind in Olszewski et al. [2006]

Poincaré, Henri [1946]
The Value of Science (1905) in The Foundations of Science, ed., and trans. G. Halsted, The Science Press, 1946.

Poincaré, Henri [1996]
Science and Method. Trans. Andrew Pyle. Routledge, London 1996.

Detlefsen, Michael [1990]
Brouwerian Intuitionsism, Mind 99, 396 (1990): 501 – 34. Reprinted in Jacquette [2002]

Jacquette, Dale [2002] Ed.
Philosophy of Mathematics, An Anthology. Blackwell.

Kellett, Owen [2005]
A multi-faceted attack on the Busy Beaver Problem. Rensselaer Polytechnic Institute, Troy, New York. 2005.

Machlin, R. & Stout, Q. [1990]. The complex behaviour of simple machines. *Physics D* 42, 85-98.

Ross, Kyle; Owen, Kellett et al. [2006]. A New-Millennium Attack on the Busy Beaver Problem. Rensselaer Polytechnic Institute, Troy, New York. 2006.

Turing, Alan [1950]
Computing Machinery and Intelligence. Mind LXI (236), p. 433 – 460. Also reprinted in Anderson [1964].] ed. Minds and Machines, Contemporary Perspectives in Philosophy, Prentice-Hall, New Jersey.