10. THE BUSY BEAVER PROBLEM

§ 10.1. The Problem

Consider the set, T, consisting of all Standard Turing Machines.

We define the Beaver function, B, as a function B: $T \to \mathbb{N}$ to the set of natural numbers by defining B(M) as follows in terms of the behaviour of M when started with a blank tape.

$$B(M) = \begin{cases} \text{\# steps before halting} & \text{– if it does halt} \\ 0 & \text{if it does not halt} \end{cases}$$

Example 1: If M, N and R are the Turing Machines:

M	0	1
0	1 L 1	1 R 2
1	0 R 0	1 L 1

N	0	1
0	1 R 1	1 R 2
1	0 R 0	1 L 1

R	0	1
0	0 L 1	1 L 2
1	1 R 0	1 L 1

then B(M) = 3 since M will halt after 3 steps when started with a blank tape.

B(N) = 0 since N will never halt and
B(R) = 6.

We extend the Beaver function to a finite set of Turing Machines, S, by defining **B(S)**, to be the largest value of B(M) for all those M $\in$ S.

Example 2: If M, N and R are the binary machines above then B({M, N, R}) = 6 since this is the maximum of B(M), B(N) and B(R).

We often describe a set of Turing Machines by using a 'wildcard' in a table. The symbol '*' can stand for any character, direction or state, according to its position in the instruction.

Example 3: The table

S	**0**	**1**
0	1 * 1	* R 2
1	0 R *	1 L 1

represents a set, S, of 12 Turing Machines. The first '*' can be either L or R. The second can be either 0 or 1. The third represents a state and has three possibilities: 0, 1 or 2.
Of these, the four machines represented by:

	0	**1**
0	1 * 1	* R 2
1	0 R 2	1 L 1

will halt after 2 steps.

The two machines of the form:

	0	**1**
0	1 L 1	* R 2
1	0 R 0	1 L 1

halt after 3 steps. The remaining six machines don't halt. Thus B(S) = 3. The reader is invited to examine these six non-halters to confirm that indeed they don't halt.

The table:

$\mathbf{T_2}$	**0**	**1**
0	* * *	* * *
1	* * *	* * *

represents the set, T_2, of all 2-state Standard Turing Machines. There are a lot of them, $2^8.3^4 = 20736$ in fact. $B(T_2) = 6$, and the machine R in example 1 is one of the record holders. However it takes a considerable amount of analysis to verify that no 2-state machine can run for more than 6 steps and still halt.

An *n*-state Standard Turing machine has 2*n* instructions. These instructions have the form *c*D*s* where *c* = 0, 1, D = L, R and *s* = 0, 1, 2, … , *n*, so in all there are

$4n + 4$ possible instructions and hence there are in all, $(4n + 4)^{2n}$ n-state Standard Turing machines.

We define, for each $n \geq 1$, $\mathbf{T}_n$ to be the set of all n-state Standard Turing Machines. The size of these sets grows rapidly with n, but each is finite. We define the **Busy Beaver Function** on the set of positive integers by $\beta(n) = B(T_n)$.

In other words $\beta(n)$ is the largest number of steps that an n-state Standard Turing Machine can run for, starting with a blank tape, and still halt. As mentioned above it can be shown that $\beta(2) = 6$. It is easy to see that $\beta(1) = 1$.

Note the six aspects to this definition:

* maximum
* number of steps
* n-state
* standard (one track, one head, 2 characters)
* start with blank tape
* only halting TMs considered

The phrase 'Busy Beaver' is an analogy with the North American animal that fells large trees with its enormous teeth to build dams and

whose tireless behaviour has given rise to the phrase "as busy as a beaver". Also note that there are variants to the Busy Beaver Function. Some references consider the largest number of 1's printed instead of the largest number of steps, and this gives rise to different values. However the actual values of are not very important. What makes the Busy Beaver function so interesting is that, no matter which definition one uses, it is *non-computable*!

Example 4: $\beta(1) = 1$.
Proof: A 1-state Standard Turing machine has the form:

	0	**1**
0	* * s	* * *

where s = 0 or 1. We consider what happens when such a machine is started with a blank tape. If s = 0 the machine will never halt. The machine will continue moving left or right for ever, leaving behind a string of 1's or 0's). In all four cases the machine will not halt.
But if s = 1 the machine will halt after 1 step and so

$$\beta(1) = 1.$$

Only three other values of $\beta(n)$ are known:

$$\beta(2) = 6$$
$$\beta(3) = 21$$
$$\beta(4) = 107.$$

It is known that $\beta(5) \geq 47{,}176{,}870$ since there is a 5-state machine that runs for this number of steps, and then halts. But to date the analysis of all 5-state machines is not complete so there could be one that runs longer than this. For $n = 6$ we have even less precise information: $\beta(6) > 3 \times 10^{1730}$. It's unlikely that $\beta(6)$ will ever be known, but we cannot prove that.

What we can show is that there will never be a computer program that can compute $\beta(n)$ for any given n, even if we had the most powerful computer we could imagine, with more memory than would be possible in this universe and even if we were prepared to wait till the death of the universe for the answer. Such a program is a logical impossibility, as we shall show.

Yet we have a perfectly well-defined function and in principle there's no reason why $\beta(n)$ couldn't be computed for any given n. But we'll prove that no program can ever be written to compute this function.

Note carefully what is being claimed here. We're not simply claiming that so far a program has never been written to compute the Busy Beaver function. We're not even claiming that it is such a difficult problem that it will probably never be solved. We're claiming that it's **logically impossible** for such a program to exist!

Mankind has found in the past that it's very risky to make claims about future possibilities. Man will never be able to fly, reach the moon or create life, it has been claimed over the years. But the first two have been proved

to be possible and there's much controversy over the third.

However the non-computability of the Busy Beaver Function is not like these. There is no controversy about whether or not it will ever be possible. It's not a case that one day someone might be clever enough, or that technology will advance so that we have the computational resources to solve the problem.

It is simply impossible – as impossible as it is for someone to discover a ruler-and-compass construction to exactly trisects a given angle (something that was proved impossible long ago) or to come up with a valid proof that $2 + 2 = 5$.

§ 10.2. Why It Is Unsolvable

The Busy-Beaver Problem is unsolvable. That is, there is no Turing Machine that computes $\beta(n)$ for all n. What's more, the fact that no such Turing Machine can exist means that no such program can ever be written in any computer language on any computer – not now, not ever. For if anyone is ever clever enough to do so then it follows that a Turing machine could be created to compute $\beta(n)$. But this would bring about a logical impossibility and our whole world of logical reasoning would collapse!

Theorem 1: If $m < n$ then $\beta(m) < \beta(n)$. In other words, β is a strictly increasing function.

Proof: If M is an *m*-state Standard TM that runs for $\beta(m)$ steps before halting then we can create an $(n + 1)$-state machine that runs for one more step by adding the instructions:

n	0 R (*n*+1)	0 R (*n*+1)

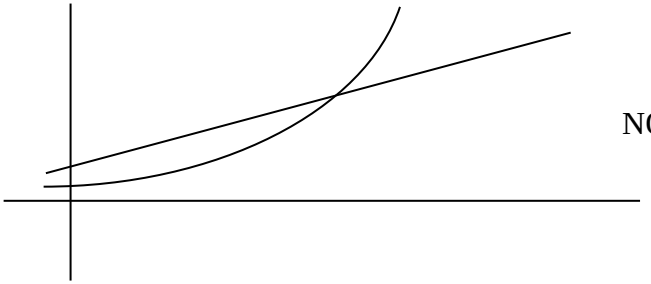
to the bottom of the table. Hence $\beta(m)$ is a strictly increasing function of *m*.

Theorem 2: There is no Turing Machine that computes the Busy Beaver function.
Proof: The proof is a proof by contradiction. Suppose that there exists an M-state Standard Turing machine called BEAVER such that BEAVER(m) = $\beta(m)$ for all *m*.

We now choose *n* large enough so that:

$$2^n > 10n + M + 2.$$

This is possible because the right hand side is a linear function of *n* while the left hand side is exponential.



But why $10n + M + 2$? The answer is that this is the number of states of a Turing Machine, Ω, that we're about to manufacture.

We have seen that there exists a 2 state Turing Machines INC such that

$$\text{INC}(n) = n + 1$$

and a 10 state machine DOUBLE such that

$$\text{DOUBLE}(n) = 2n.$$

INC	0	1
0	1 L 1	1 L 0
1	0 R 2	

DOUBLE	**0**	**1**	
0	0L5	0R1	**REPEAT**
1	0R2	1R1	
2	1L3	1R2	
3	0L4	1L3	
4	1R0	1L4	
5	0R6	1L5	
6	1R7	1R6	**ADD**
7	0L8	1R7	
8		0L9	
9	0R10	1L9	

We follow one copy of INC by n copies of DOUBLE. Given a blank tape this will output 2^n. (It differs from the machine POWER that we constructed in chapter 8 to compute any power of 2. This one only computes a specific 2^n, the power of 2 that corresponds to the particular n that we've chosen.) We now follow all this with BEAVER.

$$\Omega = \text{INC} + \underbrace{\text{DOUBLE} + \ldots + \text{DOUBLE}}_{n \text{ copies}} + \text{BEAVER}$$

Clearly Ω has $10n + M + 2$ states. Now when start with a blank tape we get 1, after INC has finished, then doubling n times we get 2^n as BEAVER takes over. The final output will be $\beta(2^n)$. In other words $\Omega(0) = \beta(2^n)$.

The machine Ω has $10n + M + 2$ states and, starting with a blank tape it will halt. After how many steps? Well, it has to print out $\beta(2^n)$ 1's, and each of these will take one step, so Ω must run for at least $\beta(2^n)$ steps before halting.

Hence $\boldsymbol{\beta(10n + M + 2) \geq \beta(2^n)}$.
But β is a strictly increasing function and since

$$2^n > 10n + M + 2$$

we conclude that $\boldsymbol{\beta(2^n) > \beta(10n + M + 2)}$.
Thus we have a contradiction, and so BEAVER cannot exist.

§ 10.3. The Halting Problem

Alan designed the perfect computer

It would be nice if someone could write a very clever computer program into which I can feed my own programs, and by analysing their structure, tell me which ones will fail to halt on certain input data. Maybe it will not happen soon. But sooner or later the ingenuity of man or woman will be able to solve that problem. Or so we might think.

Since the Turing Machines model the computing process, and everything that can ever be done on any computer we can ever imagine, let alone build, can be done on a Turing Machine, the existence of such a program would be equivalent to a Turing Machine, that we might call HALT which, when given the description of some Turing machine M, and some input data D, will decide whether or not M will halt if given as input that data.

We'll prove that no such machine HALT can exist, a consequence of which is that the above dream will always remain a fantasy. We may be able to devise such a program that works in many cases, but none that *always* works.

Theorem 3: If we can solve the Halting Problem we can solve the Busy Beaver Problem.

Proof: Suppose we had a Turing Machine HALT with the following specifications. The input to HALT is to be a pair (M, D) where M is a description of an arbitrary TM and where D is some input data. It is required that HALT itself always halts (so it would have to be a bit cleverer than just simulating M). The output is to be 1 or 0 according as M would halt, or not, when started with input D.

Then here is how we could use it to compute the Busy Beaver function. (But since we have shown that this is impossible we get our desired contradiction.)

Take a natural number *n*. There are only finitely many binary TM's with *n* states. It's a routine task to systematically generate all their tables. A Turing Machine can do that. Having generated the table of one of these *n*-state Turing machines we could use HALT to check whether it will halt when started with a blank tape. If HALT tells us that this TM will not halt we discard it, but if it tells us that M *will* halt then we simulate it, keeping a count of the number of steps. Sooner or later it will halt and we record the number of steps it took.

In this way we process all the possible *n*-state binary TM's. At the end we'll be left with a finite number of natural numbers and it's an easy thing to compute their maximum. It might take some ingenuity to do all this on a Turing Machine but since we can conceive the possibility of carrying out this task by hand we know that a Turing Machine can do it.

So the existence of a Turing Machine that solves the Halting Problem implies the existence of a Turing machine that computes the Busy Beaver function.

But that, we have shown, is impossible. The only possible flaw in our argument is our assumption that HALT exists. Therefore it cannot exist.

The Busy Beaver Problem and the Halting Problem are in fact equivalent. If one could be solved then so could the other – but in fact neither s soluble.

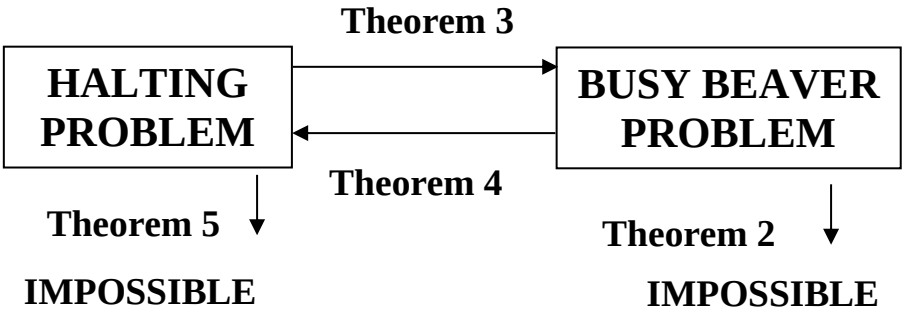
Theorem 4: If we can solve the Busy Beaver Problem we can solve the Halting Problem.

Proof: Suppose that the Busy Beaver Function is computable. We could then decide whether an arbitrary Turing Machine, M, would halt when started with a blank tape, as follows. If M has n states we use the Busy Beaver TM to compute $\beta(n)$.

Next we run M for $\beta(n)$ steps. If it hasn't halted by then it clearly will never halt, by the definition of $\beta(n)$. So in finite time we could decide whether or not M halts.

We have thus shown that the unsolvability of the Halting Problem and the non-computability of the Busy Beaver Function are equivalent. We've also given a stand-alone proof of the non-computability of the Busy Beaver function. We now provide an independent proof of the unsolvability of the Halting Problem. In effect this

provides two separate proofs of the unsolvability of both problems.



Theorem 5: There is no Turing Machine that can predict whether any given Turing Machine will halt, when started with certain data.

Proof: Let us use the notation M[D] to represent the output, on the tape, that results from running the Turing Machine M on the data D. This output is only defined if and when the machine halts, so let us use the symbol '∞' to represent the pseudo-output "machine does not halt".

Suppose there exists a machine, HALT, which solves the Halting Problem by taking the combination of the description of a Turing Machine M, and some data, D and halts with output '0' if $M[D] \neq \infty$ (i.e. M halts when started with data D) and output '1' if $M(D) = \infty$ (i.e. M does not halt with data D).

The description of M and the data D would have to be presented to HALT in a certain format which we can represent symbolically by M:D. So

$$\text{HALT(M:D)} = \begin{cases} 0 \text{ if } M(D) \neq \infty \\ 1 \text{ if } M(D) = \infty \end{cases}$$

It's possible to construct a machine Δ that duplicates some data so that $\Delta(D) = D:D$. We just need to modify COPY.

And it's not difficult to create a machine, Γ, such that $\Gamma(0) = \infty$ and $\Gamma(1) = 0$. Starting with a blank tape Γ has to go on forever, but starting with the input 1 it halts leaving a blank tape. For example the following machine would do this:

Γ	**0**	**1**
0	0 L 0	0 L 1

So now let's create the machine:

$$\Omega = \Delta + \text{HALT} + \Lambda.$$

What would happen if the machine Ω is started with its own description on the tape? Remember that the description of a Turing Machine could be regarded as just data.

Clearly Ω will either halt or not halt when fed its own description:
That is, either $\Omega(\Omega) = \infty$ or $\Omega(\Omega) \neq \infty$.

If $\Omega(\Omega) = \infty$ then $\Omega(\Omega) = \Gamma(\text{HALT}(\Delta(\Omega)))$
$= \Gamma(\text{HALT}(\Omega : \Omega))$
$= \Gamma(1)$
$= 0.$

This is a contradiction because it says that Ω *doesn't* halt, when started with its own description on the tape, then it *does* halt!

But if $\Omega(\Omega) \neq \infty$ we also get a contradiction since:
$\Omega(\Omega) = \Gamma(\text{HALT}(\Delta(\Omega)))$
$= \Gamma(\text{HALT}(\Omega:\Omega))$
$= \Gamma(0)$
$= \infty.$

If Ω *does* halt when started with its own description on the tape then it *doesn't* halt! Both possibilities lead to a contradiction so our assumption that such a machine as HALT exists must be false.

Generally students have more trouble understanding the above proof, which appears to depend on a logical sleight-of-hand, than the Busy Beaver proof, which rests on the fact that exponentials eventually overtake any linear function. For example a common response to the above proof is to recognise the "rather sneaky twist" provided by Γ and to observe that the contradiction would be avoided if it was removed.

But in a proof by contradiction one's aim is not to avoid a contradiction but rather to try to arrange a head-on collision. Fortunately we can do without Theorem 5

and prove the unsolvability of the Halting Problem via the uncomputability of the Busy Beaver function.

§ 10.4. Deciding Whether a Given Turing Machine Will Halt

Suppose S is a finite set of Turing Machines. To find the value of $\beta(S)$ we simply go through each machine in S and run it with a blank tape. If the machine halts we record the number of steps. $\beta(S)$ is the maximum of these numbers.

The problem, of course, is to know whether a given machine will halt. We can't run it for ever – at some point we must abort. And just because a machine is still running after a large number of steps doesn't guarantee that it will never halt. We may give up after 100 steps, but we won't know whether it might have halted after 103 steps, or 1003. Simply running the machine will never answer the question "will it halt?" conclusively. And the fact that the Halting Problem is unsolvable means that there's no automatic process for doing this. But in certain cases, with a little ingenuity, we can decide.

The simplest test for non-halting is to see if there's a halting instruction. Clearly a TM with no halting instruction can't possibly halt!

Example 5: Consider the Turing Machine:

	0	**1**
0	1 L 1	0 R 0
1	0 R 0	1 L 1

This machine never halts because it has no instruction of the form **2.

A Turing Machine may have a halting instruction but never be able to reach it. Now of course proving that it can never reach a halting instruction is no different to proving that it doesn't halt, but there are some cases where one can show that irrespective of what is on the tape there is no way of reaching a state that contains a halting instruction.

Example 6: Consider the Turing Machine:

	0	**1**
0	1 L 2	0 R 0
1	0 R 1	1 L 3
2	0 R 0	1 L 2

There's a halting instruction in state 1 but state 1 is inaccessible (using the same techniques as for Finite State Acceptors, ignoring the symbol and direction parts of the instructions). So the machine can't halt.

If an instantaneous description is ever repeated exactly then of course we're in a loop and the machine must continue forever.

Example 7: Consider the Turing Machine:

	0	**1**
0	1 L 2	0 R 1
1	0 R 0	1 L 3
2	0 R 0	1 L 2

This machine has a halting instruction and it is potentially accessible (considering only the transitions from state to state).

However the trace of the first 4 steps is:

0⟩ [0] 0.0 0 0
1⟩ [2] 0.0 1 0
2⟩ [0] 0.1 0 0
3⟩ [1] 0 0.0 0
4⟩ [0] 0 0 0.0

The instantaneous descriptions at steps 0 and 4 are identical. (They may not look identical and the head has moved to the right two squares. But this doesn't matter on an infinitely long blank tape! Since the subsequent behaviour depends only on the current ID this machine will repeat these 4 steps forever and so will never halt.

All of these simple tests for non-halting TMs are routine and could be built into a program. However there are many more subtle ways in which a machine can fail to halt. To deal with the general case we must detect a

pattern that we are reasonably sure will continue for ever. But, of course, being reasonably sure is not enough. We need to *prove* that our pattern will continue. What prevents this from becoming an algorithm is that there is an infinite variety of possible patterns. Discovering the patterns for all machines would require infinite creativity.

Having found what appears to be the pattern, we can prove that it must continue by mathematical induction. Induction is a very useful tool in computer science and this application illustrates that fact.

Example 8: Consider the Turing Machine:

	0	**1**
0	1 L 1	1 R 2
1	0 R 2	1 L 1
2	1 L 1	1 R 2

The trace of the first 11 steps of this machine is:

0⟩ [0] 0.00
1⟩ [2] 0.010
2⟩ [2] 0.10
3⟩ [2] 01.00
4⟩ [1] 0.110
5⟩ [1] 0.0110
6⟩ [2] 0.110
7⟩ [2] 01.10

8⟩ [2] 011.00
9⟩ [1] 01.110
10⟩ [1] 0.1110
11⟩ [1] 0.01110

So what's the pattern? It would appear that the head moves right, past all the 1's and deposits an extra 1 at the right-hand end. Then it moves left, bounces back from the 0 and repeats the whole cycle again. Because the string of 1's continues to grow, the cycles get longer and longer.

Now the above example is sufficiently simple that you may feel that you don't need a formal proof to convince you of the fact that this pattern must continue forever, and therefore that this machine doesn't halt (when started with a blank tape). And you'd be right. However the pattern only needs to get just a little more complicated before you would be unable to see it as a whole. We need to use formal methods. And to give ourselves practice with these methods we'll insist on their use, even in simple cases where it's obvious what is happening.

We pick a convenient step. Sometimes it may take a number of steps before the pattern is established. Certainly step 0 never recurs, but we could use step 1. Something like it occurs again in step 5 and again in step 11. In each case we're in state 1, scanning a 0. To the right

of that 0 is a block of 1's and the number of 1's grows by one each time.

What's the pattern of these numbers 1, 5, 11, ...? You might guess that the successive differences, 4, 6, ... continue ... , 8, 10. That would mean that the next time we have a similar instantaneous description would be in step 19. This proves to be the case.

So we could make our hypothesis:

For all natural numbers $n > 0$ the ID after a certain number of steps is [1] ... 0.01^n0 ...

We can now prove this by induction by firstly checking that it holds for $n = 1$, which it does, and then supposing that it's true for n and proving that it's true for $n + 1$.

Suppose that it's true for n. That is, after a certain number of steps (the number of steps is not important) the ID is:
[1] 0.01^n0

We operate this machine for some further steps:

[1] 0.01^n0
[2] $0.11^{n-1}0$
.....
[2] $01^n.00$
[1] $01^{n-1}.110$
[1] 0.11^n0
[1] $0.01^{n+1}0$

Thus the result holds for $n + 1$ and so, by induction, it holds for all n. This establishes beyond all doubt that this machine will never halt.

Whenever you are required to show that a given Turing Machine doesn't halt you can demonstrate it formally by one of the following:

(1) showing that it has no halting instruction;

(2) showing that none of the halting instructions are in a state that is accessible;

(3) showing from your trace that an instantaneous description is repeated exactly

(4) by observing a pattern and proving it by induction.

It's not good enough to say "clearly it is ...." and describing in words what appears to be happening.

§ 10.5. The Flying Dutchman Problem

In the legend of this name the Dutchman is condemned for blasphemy and is doomed to sail the seven seas forever, without making landfall. There have been claimed sightings in the 19th and even in the 20th century which, some say, add weight to the claim that the story is based on truth.

You can read about the legend at the website:

www.occultopedia.com/f/flying_dutchman.htm

In Wagner's operatic version the Dutchman is allowed to come ashore once every seven years, which gives the opportunity for a young girl, Senta, to fall in love with him.

In a way the Flying Dutchman (in its original version) is like a Turing Machine that never halts. Yet, because the earth is finite, he never travels more than a certain distance (halfway round the world) from his port of departure.

If, when the Turing Machine M is run with a blank tape, the head only ever moves a finite distance from its starting position, we define D(M) to be the largest such distance. If the head moves arbitrarily far we define D(M) = 0.

Example 9: If M, N, R are the following machines:

M	**0**	**1**
0	1 L 1	1 L 1
1	0 R 0	1 L 1

N	**0**	**1**
0	1 R 1	1 R 2
1	0 R 0	1 L 1

R	**0**	**1**
0	0 L 1	1 L 2
1	1 R 0	1 L 1

then D(M) = 1, D(N) = 0 and D(R) = 2.

Theorem 6: The Flying Dutchman function, D, is non-computable.

Proof: Suppose there's a Turing Machine that computes the D function and suppose M is an arbitrary TM. If D(M) = 0, M does not halt.

Suppose D(M) > 0. Then the head can only ever reach K = 2D(M) + 1 squares and so there are only 2^K possible tapes that can occur. With K possible positions of the head and *n* states the number of distinct instantaneous descriptions that can occur is at most $nK2^K$. So to determine whether or not M halts we simulate it until either it halts or until it has run for

$$nK2^K + 1 \text{ steps.}$$

If it's still running after this number of steps it must have repeated an instantaneous description and so will never halt.

So, if D was computable, we could solve the Halting Problem. Therefore D cannot be computable.

EXERCISES FOR CHAPTER 10

EXERCISES 10A (Halting)

Ex 10A1: The following table represents a family of 4 Turing Machines M_n for n = 0, 1, 2, 3:

	0	**1**
0	0L*n*	1L2
1	1R0	1L1
2	1L3	0L2

For each of the 4 values of n give the output of the machine if, and when, the machine halts. If one of these machines doesn't halt, when started with a blank tape, give brief reasons why you're convinced it will never halt.

Ex 10A2: Show that none of the following Turing Machines will halt when started with a blank tape.

(i)

	0	**1**
0	1L1	1R1
1	0R0	0L0

(ii)

	0	**1**
0	0R2	1L2
1	0L0	1R3
2	1L0	0R2

(iii)

	0	**1**
0	0R0	1L2
1	0L0	1R3
2	1L0	0R1

(iv)

	0	**1**
0	1R1	0R0
1	0L0	1L2

Ex 10A3: Show that if the following Turing Machine is started with a blank tape, the instantaneous description after $n^2 + 3n$ steps is: [0] 0.1^n.

	0	**1**
0	1R1	1R0
1	0L2	1L3
2	0R0	1L2

Ex 10A4: Give reasons why none of the following Turing machines will halt, when started with a blank tape.

(a)

	0	**1**
0	1L2	1R0
1	0R2	1L2
2	1R3	0L3
3	1R2	0L1

(b)

	0	**1**
0	1L1	1R0
1	0R3	1L0
2	1R2	0L5
3	1R1	0L1
4	0L5	1R4

(c)

	0	**1**
0	1L3	1R4
1	0L3	1R1
2	1L4	0L0
3	1R1	0L1

(d)

	0	**1**
0	1L2	1L3
1	1R3	0R3
2	1R0	0L1
3	0L4	1R0

Ex 10A5: The Turing Machine below is run, starting with a blank tape. Prove by induction that for all $n \geq 0$, after $3n + 1$ steps, the machine is in state 1 with the tape reading $\ldots 01^{n+1}\mathbf{.}0\ldots$

	0	1
0	1R1	0R3
1	1L2	1R1
2	0L3	1R1

Ex 10A6: The Turing Machine below is run starting with a blank tape. Prove by induction that for all $n \geq 2$, after a certain number of steps (in this question you don't have to keep track of how many steps it takes), the machine is in state 2 with the tape reading:

$$\ldots 011\mathbf{.}(10)^n 0\ldots$$

	0	1
0	1R1	1L3
1	1R2	0R10
2	1R3	1R3
3	0R4	0R10
4	1L5	1R8
5	0L6	1L6
6	0L10	1L7
7	0L9	1R2
8	0R4	1R9
9	1L10	1L7

Ex 10A7: Prove by induction on n that if the following TM is started with a blank tape, the ID after $10n + 3$ steps will be [3] $(1110)^n 1\mathbf{.}11$

	0	1
0	1R2	0L5
1	0R0	1R1
2	1R4	1R5
3	0R1	1L3
4	1L3	1L5

Ex 10A8: Prove by induction on *n* that if the Turing Machine

	0	1
0	1R3	1R1
1	1R4	0L2
2	0L3	1R0
3	1L0	0L4

starts with a blank tape then after 5*n* steps the Instantaneous Description is [0] 1^n**.**0.

EXERCISES 10B (Busy Beaver)

Ex 10B1: A *Right-Wing* Turing Machine is defined to be one in which every instruction moves the head to the right. Let ρ(n) denote the largest number of steps that an n-state binary Right-Wing Turing Machine can run, starting with a blank tape, and still halt.

(a) Compute ρ(3).

(b) Prove that ρ(*n*) is computable by constructing a suitable TM.

Ex 10B2: (a) Let β be the binary Busy Beaver Function. The value of β(92) is not known and probably never will be. However it's possible to obtain a lower bound.
Show that $\beta(92) \geq 1024$ by constructing a 92 state binary Turing Machine that, when started on a blank tape, will run for over 1024 steps before halting.

(b) Can you improve on this lower bound? (i.e. how big a *K* can you find for which
$\beta(92) \geq K$?)
HINT: Of course it's too much to build such a big machine from scratch. Instead you should build one out of the smaller machines in this chapter. Reading the proof of the non-computability of the Busy Beaver function may give you some ideas.

Ex 10B3: Calculate the Busy Beaver number of the set of Turing machines given by the following table in which ∗s are wild cards. Notice that the machine in question Ex 10A5 is one of the machines in this set.

	0	**1**
0	1∗∗	0R3
1	1L2	1R1
2	0L3	1R1

Ex 10B4: Calculate the Busy Beaver number of the set of Turing Machines in which ∗s are wild cards.

	0	1
0	1R1	1R0
1	0L2	1L3
2	0R*	1L2

Ex 10B5: Find the Busy Beaver number of the set of Turing Machines represented by

	0	1
0	1R3	1R1
1	1R4	0L2
2	0L3	1**
3	1L0	0L4

EXERCISES 10C (Discussion of Computability)

Ex 10C1: The Busy Beaver Problem is the problem of devising an algorithm to calculate the function $\beta(n) = M$, where M is the largest number of steps a n-state binary Turing Machine can run for and still halt. What is the status of this problem? That is, has it been solved, or is a solution expected in the near future?

Ex 10C2: In about half a page discuss the following quotation:
"*The Turing Machine was the first computer ever built. It is now obsolete and has purely historical interest.*".

SOLUTIONS FOR CHAPTER 10

Ex10A1: M_0 doesn't halt given a blank tape. It will continue moving left, obeying the instruction 0L0.
M_1 halts after 8 steps; M_2 halts after 2 steps; M_3 halts after 1 step.

Ex 10A2: (i) has no halting instruction;
(ii) has a halting instruction in state 1 but can never reach that state;
(iii) never leaves state 0;
(iv) repeats an instantaneous description;

Ex 10A3: Proof by induction: For $n = 0$ it is clearly true since for $n = 0$, 0.1^n represents a blank tape. Suppose it holds for n. Then after $n^2 + 3n$ steps the ID is:

$n^2 + 3n\rangle \quad [0] \quad 0.1^n$.

Continuing, we get the following sequence of IDs:

$n^2 + 3n + 1\rangle \quad [0] \quad 1.1^{n-1}$
$n^2 + 4n\rangle \quad [0] \quad 1^n.0$
$n^2 + 4n + 1\rangle \quad [1] \quad 1^{n+1}.0$
$n^2 + 4n + 2\rangle \quad [1] \quad 1^n.1$
……..……
$n^2 + 5n + 2\rangle \quad [2] \quad 0.1^{n+1}$
$n^2 + 5n + 3\rangle \quad [2] \quad 0.01^{n+1}$
$n^2 + 5n + 4\rangle \quad [0] \quad 0.1^{n+1}$

So after $(n + 1)^2 + 3(n + 1) = n^2 + 5n + 4$ steps the ID is

$$[0] \quad 0.1^{n+1}$$

and so the statement is true for $n + 1$. Therefore, by induction, it's true for all n. It follows that this Turing Machine won't halt, when started with a blank tape.

Ex 10A4: (a) has no halting instruction.
(b) This TM starts in state 0 and from there it can only ever reach states 0, 1, 3 so it can never reach the halting instructions in states 2 and 4.
(c) This TM moves to state 3 and from there it never leaves either state 1 or 3. None of these have a halting instruction.
(d) the ID [0] 1.1 occurs after 2 steps and again after 4 steps.

Ex 10A5:
$0\rangle$ [0] 0.0
$1\rangle$ [1] 1.0
Hence the result holds for $n = 0$.

Suppose that it holds for n.
$3n + 1\rangle$ [1] $1^{n+1}.0$
$3n + 2\rangle$ [2] $1^{n}.11$
$3n + 3\rangle$ [1] $1^{n}1.1$
$3n + 4]$ [1] $1^{n}11.0$
So after $3(n + 1) + 1 = 3n + 4$ steps the ID is [1] $1^{n}11.0$, that is [1] $1^{n+2}.0$
Hence the result holds for $n + 1$.
Therefore, by induction, it holds for all n.

Ex 10A6: The trace for this TM begins as follows:

[0] 0**.**0
[1] 1**.**0
[2] 11**.**0
[3] 111**.**0
[4] 1110**.**0
[5] 111**.**01
[6] 11**.**101
[7] 1**.**1101
[2] 11**.**1010

So the result holds for $n = 2$.

Suppose it holds for $n \geq 2$. The ID at some stage is:

[2] 11**.** $(10)^n$
[3] 111**.**0$(10)^{n-1}$
[4] 1110**.** $(10)^{n-1}$
[8] 11101**.**0$(10)^{n-2}$
[4] 1110(10)**.**$(10)^{n-2}$
……
[4] 11$(10)^n$**.**0
[5] 11$(10)^{n-1}$1**.**01
[6] 11$(10)^{n-1}$**.**1010
[7] 11$(10)^{n-2}$1**.**01010
[9] 11$(10)^{n-2}$**.**101010
……
[9] 11**.** $(10)^{n+1}$
[7] 1**.**1$(10)^{n+1}$
[2] 11**.**$(10)^{n+1}$

Hence the result holds for $n + 1$.
Therefore, by induction, it holds for all n.

Ex 10A7:
Check for $n = 0$:

$0\rangle$ [0] 0.0
$1\rangle$ [2] 1.0
$2\rangle$ [4] 11.0
$3\rangle$ [3] 1.11

Suppose that the result is true for n. Then
$10n + 3\rangle$ [3] $(1110)^n$1.11
$10n + 4\rangle$ [3] $(1110)^n$.111 = $(1110)^{n-1}$1110.111
$10n + 5\rangle$ [3] $(1110)^{n-1}$111.0111
$10n + 6\rangle$ [1] $(1110)^{n-1}$1110.111 = $(1110)^n$.111
$10n + 7\rangle$ [1] $(1110)^n$1.11
$10n + 8\rangle$ [1] $(1110)^n$11.1
$10n + 9\rangle$ [1] $(1110)^n$111.0
$10n + 10\rangle$ [0] $(1110)^n$1110.0 = $(1110)^{n+1}$.0
$10n + 11\rangle$ [2] $(1110)^{n+1}$1.0
$10n + 12\rangle$ [4] $(1110)^{n+1}$11.0
$10n + 13\rangle$ [3] $(1110)^{n+1}$1.11

So after $10(n + 1) + 3$ steps the ID is [3] $(1110)^{n+1}$1.11
Hence the result is true for $n + 1$.
Therefore, by induction, it is true for all n.

Ex 10A8: The statement is clearly true for $n = 0$.

$5n\rangle$ [0] $1^n.0$

$5n + 1\rangle$ [3] $1^{n+1}.0$

$5n + 2\rangle$ [0] $1^n.11$

$5n + 3\rangle$ [1] $1^{n+1}.1$

$5n + 4\rangle$ [2] $1^n.1$

$5n + 5\rangle$ [0] $1\rangle^{n+1}.0$

Hence the statement holds for $n + 1$ and so, by induction, it holds for all n.

Ex 10B1: (a) Write the machine in the following form

	0	**1**
0	aRx	dRu
1	bRy	eRv
2	cRz	fRw

and consider the following cases which cover all possibilities:

(1) $x = 0$;
(2) $x = 1$, $y = 0$;
(3) $x = 1$, $y = 1$;
(4) $x = 1$, $y = 2$, z = 3;
(5) $x = 1$, $y = 2$, z < 3;
(6) $x = 1$, $y = 3$,
(7) $x = 2$, z = 0;
(8) $x = 2$, z = 1, $y = 3$;
(9) $x = 2$, z = 1, $y < 3$;
(10) $x = 2$, z = 2;
(11) $x = 2$, z = 3;
(12) $x = 3$.

Machines in cases (1), (2), (3), (5), (7), (9), (10) clearly never halt. Machines in case (12) halt after 1 step, those in cases (6) and (11) halt after 2 steps and those in cases (4) and (8) halt after 3 steps. Hence $\rho(3) = 3$.

Alternatively observe that since Right-Wing TMs always move right, they can never read what they've written. Hence the only ever read 0's. The right half of the table can be ignored and the subsequent behaviour of the machine at any stage depends solely on what state it's in. Hence if this machine ever repeats a state it must go forever.

With 3 states a halting machine of this type cannot run for more than 3 steps and still halt. Thus $\rho(3) \leq 3$. Since it's easy to construct a machine that goes from state 0 to state 1 to state 2 and then halt, $\rho(3) \geq 3$. Hence $\rho(3) = 3$

(b) Using the same argument in the above alternative solution to the general n-state case we see that $\rho(n) = n$. Thus ρ is the identity function and this is clearly computable. The following machine computes it by simply moving the head, first left then right.

	0	**1**
0	0R1	1L0

Ex 10B2: INC + 10 copies of DOUBLE has 92 states and produces output of $2^{10} = 1024$ when started with a blank tape. Each 1 printed takes at least 1 step and so this

machine must run for at least 1024 steps. Hence $\beta(92) \geq 1024$.
(b) The following machine produces six 1's when started with a blank tape.:
Follow this by 5 copies of POWER (17 states), making a 92 state machine which writes
$2 \wedge (2 \wedge (2 \wedge (2 \wedge 64)))$ on the tape when started with a blank tape. The number of steps that the machine runs for, before halting, must be at least this big so $\beta(92) \geq 2 \wedge (2 \wedge (2 \wedge (2^{64})))$.
This is a *very* large number yet it's still probably quite tiny when compared with the actual value of $\beta(92)$ itself!

Ex 10B3: Represent the values of the wildcards by D, *s*.

	0	**1**
0	1D*s*	0R3
1	1L2	1R1
2	0L3	1R1

If *s* = 3 the TM halts after 1 step.
If *s* = 2 the TM halts after 2 steps.
If *s* = 0 the TM never halts.
If *s* = 1 and D = R the TM is the one in question (2) and so never halts.
If *s* = 1 and D = L the TM halts after 3 steps.
Hence the Busy Beaver number of the set is 3.

Ex 10B4: If $*$ = 0 the TM does not halt (this can be proved by induction).
If $*$ = 1 the TM halts after 5 steps.

If $*$ = 2 the TM doesn't halt (it repeats an instantaneous description).
If $*$ = 3 the TM halts after 4 steps.
Hence the Busy Beaver number is 5.

Ex 10B5: In all cases we have

0⟩ [0] 0.0
1⟩ [3] 1.0
2⟩ [0] 0.11
3⟩ [1] 1.1
4⟩ [2] 0.1

1L0: Halts after 7 steps.
1L1: Halts after 6 steps.
1L2: Halts after 9 steps.
1L3: Halts after 8 steps.
1L4: Halts after 5 steps.
1R0: Does not halt by Ex 10A8.
1R1: Halts after 6 steps.
1R2: Halts after 7 steps.
1R3: Doesn't halt since the instantaneous description after 5 steps is the same as after 1 step.
1R4: Halts after 5 steps.
Hence the Busy Beaver number is 9.

Ex 10C1: The Busy Beaver Problem has been proved to be unsolvable. Therefore no solution can ever be obtained.

Ex 10C2: This statement is incorrect. There are problems that can be precisely defined for which it can be proved that no computer program can solve them. One is the computation of the Busy Beaver function, β. β(*n*) is the largest number of steps that an *n*-state Standard Turing Machine, starting with a blank tape, can run and still halt. This function is not computable by a Turing Machine, and since Turing Machines can model any computer running any programming language, it's impossible for such a computer program to be written.

The Halting Problem is to devise a Turing Machine that can take the description of a Turing Machine M, and some data D and decide whether or not M would halt if given the data D on its tape. This problem is in fact equivalent to the problem of computing the Busy Beaver problem and is therefore unsolvable.

Thus there are problems which can be posed precisely but where it's a logical impossibility to have a Turing Machine, or any other program for that matter, to solve them.