

**York University**  
**CSE 2001 – Unit 2 Models**  
**Instructor: Jeff Edmonds**

Don't cheat by looking at these answers prematurely.

1. Loops: Do some computational problems require two loops or can all of them be accomplished by a single loop?
  - Answer: A single loop works just fine. A multi-loop program can be converted into one with goto's and this can be converted into a single loop program. Also a TM can be simulated with a single loop.
2. Suppose we want to design a TM to know whether a graph  $G$  is connected. What is the difference between  $G$  and  $\langle G \rangle$ ? When do we use  $\langle G \rangle$  and why is this done?
  - Answer:  $\langle G \rangle$  is the binary string representation of the graph  $G$ . TM take only strings as inputs. This allows them to have a standard input and allows the bits of the string to be placed one per memory square.
3. What is the difference between a computational problem that is a function vs one that is a language? If  $L$  is a language, what (if anything) do the following mean:  $L(x)$ ,  $x \in L$ ,  $L = \emptyset$ ,  $L = \{0,1\}^*$ ,  $L$  is decidable.
  - Answer: A language is the set of input instances for which the answer is "yes". If  $x$  is a string then  $L(x)$  has an improper syntax. If  $x$  is a TM then  $L(x)$  is the language computed by it.  $x \in L$  states that the string is in the language and hence the TM should accept.  $L = \emptyset$  states that there are no strings in the language. The TM should always reject.  $L = \{0,1\}^*$  states that every strings is in the language. The TM should always accept.  $L$  being decidable means that there is a TM such that on every finite binary string it eventually stops and gives the correct answer, i.e. *accepts* if the input is in  $L$  and *rejects* if it is not.
4. (Answer in Slides) TM Transitions: In the slides we proved that a TM that is only able to write the characters  $\{0,1,b\}$  on the tape is as powerful as one that has a larger tape alphabet  $\Sigma$ . The simpler machine simulated the more complex one by grouping its cells into blocks of some fixed length so that each block contained a 0/1 code for the required character from  $\Sigma$ . Suppose this block size is four, input character 0 is encoded with 0000 and 1 with 0001. The simulation must start by inserting three zeros in front of every bit of the input. For example, on input  $b1011bb\dots$ , the output of this first phase should be  $\dots bb0001000000010001bb\dots$ . Note that we don't actually care if the the output is at the beginning of the tape as long as it is surrounded by blanks.
  - (a) Writing down TM descriptions is hard. Instead, start by writing pseudo code using variables and loops. This is Jeff's level 4 abstraction of a TM. The only allowed actions are to read and write to the tape where the head is and to move the head to the left and right. I am also a big believer in *loop invariants*. This is a clear picture of what the tape looks like at the beginning of each iteration.

Hint: Use the following Loop Invariant. For some  $i \in [0, n]$ , the first  $i$  bits of the input have been blanked out. These bits have been copied into a block past the input separated by a single blank with three zeros inserted before each. The head is on the  $b$  before

    - Answer:  
**algorithm** AddZerosBetweenEachDigit()  
***<pre-cond>***: Assume head on the  $b$  before the input.  
***<post-cond>***: Required output on tape after where the input had been

```

begin
  loop      Loop Invariant: See above.
1:      Move head right one
2:       $a = \text{bit at head (i.e. the } i+1^{\text{st}} \text{ input bit), Write a blank, Move head right}$ 
3:      Move head right until it is at a blank (i.e. after remaining input)
4:      Move head right one (i.e. into the output)
5:      Move head right until it is at a blank (i.e. after produced output)
6-9     Write 0, head right; Write 0, head right; Write 0, head right; write  $a$ 
10:     Move head left until it is at a blank (i.e. after produced output)
11:     Move head left one (i.e. into the remaining input)
12:     if head is on a blank then exit loop (i.e. no more input)
13:     Move head left until it is at a blank (i.e. after remaining input)
  end loop
end algorithm

```

(b) (Sorry: see the answer to the previous question before continuing.)

Your next task is to prove that you know how to translate this code into a TM. You do not need to give the full translation. That would be far too tedious. Suppose a program had 13 lines of code and two variables  $x, y \in \{1..5\}$ , then what Pooh bear should write on the wall is the current line number that is being executed and the value of  $x$  and of  $y$ . Hence for each  $k \in \{1..13\}$ ,  $x', y' \in \{1..5\}$ , let  $q_{\langle \text{line}=k, x=x', y=y' \rangle}$  denote the state in which the TM is on line  $k$ ,  $x$  has the value  $x'$ , and  $y$  has the value  $y'$ . What are the states for the above program and how many of them are there? Give the general idea for how the transition function  $\delta(q, c) = \langle q', c', \text{direction} \rangle$  is defined.

- Answer: Which of the 13 lines of code the algorithm is on needs to be remember as well as the 0/1 value of  $a$ . This requires only  $26 = 13 \times 2$  different states. Let  $q_{\langle \text{line}=k, a=a' \rangle}$  denote the state in which the TM is on line  $k$  and  $a$  has the value  $a'$ . For each such state and each value of  $c$ , when the TM is in this state and has its head on  $c$ , it should be clear from the code which state the TM should be in next, whether to write, and how to move the head.

(c) Consider line 2 of the code,

i.e. “ $a = \text{bit at head (i.e. the } i+1^{\text{st}} \text{ input bit), Write a blank, Move head right}$ ”.

i. This line of code is a level 3 abstraction of a TM, i.e. we interpret these blocks of transitions as lines of Java code. Explain what this code gets the TM to do.

- Answer: If the algorithm is on code line 2, independent of the current value of  $a$ , if a 0 is being read, set  $a = 0$  and if a 1 is being read, set  $a = 1$ . (If a  $b$  is being read, panic.) Then write a blank and move the head right.

ii. Give a level 1 abstraction of this line, namely “Meaningful State Names”. This involves giving each relevant entry of the table defining  $\delta(q, c) = \langle q', c', \text{direction} \rangle$ . The change from the level 0 abstraction is that we have given states meaningful names  $q_{\langle \text{line}=39, x=3, y=0 \rangle}$ .

- Answer: The level 1 transitions in the table are:

$$\delta(q_{\langle \text{line}=2, a=0 \rangle}, 0) = \langle q_{\langle \text{line}=3, a=0 \rangle}, b, \text{right} \rangle$$

$$\delta(q_{\langle \text{line}=2, a=0 \rangle}, 1) = \langle q_{\langle \text{line}=3, a=1 \rangle}, b, \text{right} \rangle$$

$$\delta(q_{\langle \text{line}=2, a=1 \rangle}, 0) = \langle q_{\langle \text{line}=3, a=0 \rangle}, b, \text{right} \rangle$$

$$\delta(q_{\langle \text{line}=2, a=1 \rangle}, 1) = \langle q_{\langle \text{line}=3, a=1 \rangle}, b, \text{right} \rangle$$

$$\delta(q_{\langle \text{line}=2, a=0 \rangle}, b) = \langle q_{\text{halt error}}, b, \text{right} \rangle$$

$$\delta(q_{\langle \text{line}=2, a=1 \rangle}, b) = \langle q_{\text{halt error}}, b, \text{right} \rangle$$

iii. Give level 2 abstraction, namely “Meta Names and Transition Function”. We talk about a block of states  $q_{\langle \text{line}=39, x=x, y=y \rangle}$  for  $x \in \{0, 1, 2, 3, 4\}$ , and  $y \in \{0, 1, 2, 3, 4\}$ . We fill in blocks of the transition function table  $\delta(q_{\langle \text{line}=39, x=x, y=y \rangle}, c) = \langle q_{\langle \text{line}=40, x=x, y=c \rangle}, c, \text{right} \rangle$ .

- Answer: The level 2 generalization of this is

$$\forall a', c \in \{0, 1\}, \delta(q_{\langle \text{line}=2, a=a' \rangle}, c) = \langle q_{\langle \text{line}=3, a=c \rangle}, b, \text{right} \rangle$$

- (d) Do the same for lines 3 and 4, i.e. “Move head right until it is at a blank” and “Move head right one (i.e. into the output)”.

• Answer:

i. Suppose the algorithm is on code line 3 and is remembering that  $a = 0$ . Then the TM is in state  $q_{\langle \text{line}=3, a=0 \rangle}$ . If it is reading the bit 0/1, then because this is not a blank, it should not break this one line loop, leave the tape alone, and move the head right.

ii. The level 1 table entries are

$$\delta(q_{\langle \text{line}=3, a=0 \rangle}, 0) = \langle q_{\langle \text{line}=3, a=0 \rangle}, 0, \text{right} \rangle.$$

$$\delta(q_{\langle \text{line}=3, a=0 \rangle}, 1) = \langle q_{\langle \text{line}=3, a=0 \rangle}, 1, \text{right} \rangle.$$

$$\delta(q_{\langle \text{line}=3, a=1 \rangle}, 0) = \langle q_{\langle \text{line}=3, a=1 \rangle}, 0, \text{right} \rangle.$$

$$\delta(q_{\langle \text{line}=3, a=1 \rangle}, 1) = \langle q_{\langle \text{line}=3, a=1 \rangle}, 1, \text{right} \rangle.$$

iii. The level 2 meta table entries are

$$\forall a', c \in \{0, 1\}, \delta(q_{\langle \text{line}=3, a=a' \rangle}, c) = \langle q_{\langle \text{line}=3, a=a' \rangle}, c, \text{right} \rangle.$$

Another case:

i. On the other hand, if it is reading a  $b$ , then it should leave the head and the tape alone, but break out of this loop moving to line 4 of the code. Actually, line 4 only moves the head right, so we can simply do that and move to line 5.

ii. The level 1 table entries are

$$\delta(q_{\langle \text{line}=3, a=0 \rangle}, b) = \langle q_{\langle \text{line}=5, a=0 \rangle}, b, \text{right} \rangle$$

$$\delta(q_{\langle \text{line}=3, a=1 \rangle}, b) = \langle q_{\langle \text{line}=5, a=1 \rangle}, b, \text{right} \rangle$$

iii. The level 2 meta table entries are

$$\forall a' \in \{0, 1\}, \delta(q_{\langle \text{line}=3, a=a' \rangle}, b) = \langle q_{\langle \text{line}=5, a=a' \rangle}, b, \text{right} \rangle$$

- (e) Do the same for line 9, i.e. “Write  $a$ ”.

• Answer:

i. Suppose the algorithm is on code line 9 and is remembering that  $a = 0$ , then it should over write the blank with a 0 and leave the head put. Actually, it could start the line 10 loop by moving the head left.

ii. The level 1 table entries are

$$\delta(q_{\langle \text{line}=9, a=0 \rangle}, b) = \langle q_{\langle \text{line}=10, a=0 \rangle}, 0, \text{left} \rangle.$$

$$\delta(q_{\langle \text{line}=9, a=1 \rangle}, b) = \langle q_{\langle \text{line}=10, a=1 \rangle}, 1, \text{left} \rangle.$$

iii. The level 2 meta table entries are

$$\forall a', c \in \{0, 1\}, \delta(q_{\langle \text{line}=9, a=a' \rangle}, b) = \langle q_{\langle \text{line}=10, a=a' \rangle}, a', \text{left} \rangle.$$

Another case:

i. If the head is not reading a blank, then there is an error.

ii. The level 1 table entries are getting tedious.

iii. The level 2 meta table entries are

$$\forall a', c \in \{0, 1\}, \delta(q_{\langle \text{line}=9, a=a' \rangle}, c) = \langle q_{\text{halt error}}, ?, ? \rangle.$$

Sometimes we dont bother writing giving the error detections code.

- (f) Bonus 10 marks if correct: The problem is to redo the previous problem, except the TM can only write 0 or 1. The tape starts with a two blanks before the input and an infinite number after it, but once a blank gets a 0/1 written in it, it will never be blank again. Hint: At first I was thinking that this could not be done, because the TM could not differentiate between whether what is written on the tape is the initial input or some markings to know that progress had been made. Then I saw how to do it. See if you can do it too.

• Answer: The TM in the start state writes a 1 on the second blank and never goes back to the start state. The loop invariant is the same as before, except instead of the first  $i$  bits of the input having been blanked out, the first  $i + 2$  characters of the tape have been replaced with the first blank still there, followed by  $i$  zeros, followed by a single one. When the head is moving left looking for the  $i + 1^{\text{st}}$  character to read, before it simply moved left to the blank

separating the input and the output and then left past what remained of the input until it found a blank, then moved one cell right to what would be the  $i + 1^{st}$  character to read. Now when moving left past what remained of the input, it does not come to a blank until it finds the single blank at the beginning of the tape. At this point, it moves right past this first blank and past the  $i$  zeros, until it gets to the one. Then it writes a zero on top of this one and moves right onto what would be the  $i + 1^{st}$  character to read. After remembering this character by changing the state to have  $a$  have its value, the TM overwrites this  $i + 1^{st}$  with a one and continues the algorithm as before. Note how the loop invariant is maintained.

5. (Answer in Slides) TM computing  $J(I)$

- (a) The input consists two blocks each consisting of exactly a million 0/1 characters. The first is the binary description of a JAVA program  $J$ , the second of an input  $I$ . Describe, if possible, a simple TM for computing whether  $J(I) = 1$ , meaning that program  $J$  on this input halts and accepts or not,  $J(I) = 0$ . Focus on giving a meaningful name to each of the states of your machine. Describe the complete  $\delta(q, c) = \langle q', c', direction \rangle$  function in all of its details.

- Answer: The algorithm reads in the two million bits and remembers them all. This forms a binary tree of states. After it has read the binary string  $\alpha$ , the TM is in state  $q_\alpha$ . It reads next character and moves the head to the right. This is done with  $\delta(q_\alpha, c) = \langle q_{\alpha c}, b, right \rangle$ . Note that it is blanking the input as it goes. When it has scanned the entire input  $\langle P, I, \rangle$ , it remembers this input by being in state  $q_{\langle P, I, \rangle}$ . Note this tree of states contains  $2 \times 2^{2,000,000} - 1$  states. The TM then moves the head to the first cell. Once the TM is in this state, the transition function  $\delta$  simply has encoded in it what the output should  $M(I)$  be. The TM can write the answer and halt.  $\delta(q_{\langle J, I, \rangle}, b) = \langle q_{halt}, J(I), stay \rangle$ .

- (b) Redo the question when  $J$  is arbitrarily long, but  $I$  is still of this fixed length. Remember that lots of problems can be solved that have arbitrarily long inputs so if you want to say that this one is uncomputable, don't give this as your reason.

- Answer: This one cannot be solved because knowing whether an arbitrary program  $J$  halts even on the input 0 is undecidable.

6. (Answer in Slides) Recall that a Universal TM is a TM  $M_{universal}$ , which when given input  $\langle "M", I \rangle$ , simulates TM  $M$  on input  $I$ . You are to give a sketch of how such a  $M_{universal}$  would be built.

To make it easier, lets assume that  $M$  only has one tape and one head.

Hint: Because  $M_{universal}$  is one fixed TM, it must have one fixed set of states  $Q_{universal}$  and one fixed tape alphabet  $\Sigma_{universal}$  (i.e. the set of characters  $c$  that it is allowed to write in each tape call.)

Hint: Fix  $\Sigma_{universal} = \{ '0', '1', ' \langle', ' ', ' \rangle' \}$ .

Hint: Let  $M_{universal}$  have three tapes each with its own head.

When  $M_{universal}$ 's computation begins, the description of  $M$  will appear on  $M_{universal}$ 's first tape and  $I$  on its second.

- (a) We must assume that the TM  $M$  described in  $M_{universal}$ 's input has an arbitrarily large set of states and an arbitrarily large tape alphabet  $\Sigma_M$ .

Explain how you would describe  $M$  on  $M_{universal}$ 's first tape. Remember this description must be encoded using characters from  $\Sigma_{universal} = \{ '0', '1', ' \langle', ' ', ' \rangle' \}$ .

- Answer: Each state  $q$  in  $M$ 's set of states  $Q_M$  is specified by a unique (finite) 0/1 string " $q$ " and each character  $c \in \Sigma_M$  with a string " $c$ ".

Each of these strings will be of some fixed length  $r_M$  depending on  $M$ .

( $M$ 's input  $I$  is assumed to be specified using characters from this tape alphabet  $\Sigma_M$ .)

The description of  $M$  will consist of a complete list of tuples  $\langle "q", "c", "q'", "c'", "direction" \rangle$ , where  $\delta_M(q, c) = \langle q', c', direction \rangle$ .

- (b) At the beginning of  $M_{universal}$ 's simulation of the  $t_M^{th}$  time step of  $M$ 's computation,  $M_{universal}$ 's first tape will still contain the description of  $M$ . What would be best to put on  $M_{universal}$ 's second and third tape? Remember what you have written there must be encoded using characters from

$\Sigma_{universal} = \{ '0', '1', '\langle', '\rangle', '\langle', '\rangle' \}$ .

Also where will  $M_{universal}$  three tape heads be.

- Answer: The second tape of  $M_{universal}$  will encode the current contents of  $M$ 's tape. It will be encoded as the sequence of tuples  $\langle "c_1" \rangle, \langle "c_2" \rangle, \dots$  where  $c_i \in \Sigma_M$  is the character in  $M$ 's  $i^{th}$  cell.  
 $M_{universal}$ 's second head will be pointing at the beginning of the tuple  $\langle "c" \rangle$  corresponding to character  $c$  in the cell that  $M$ 's head is pointing at.  
 The third tape of  $M_{universal}$  will contain " $q$ ", where  $q \in Q_M$  is the current state of  $M$ .

(c) We say that  $M$  can compute in one time step any function of "what it knows", where "what it knows" is its current state and the character that its head currently on. More formally, if  $M$  is in state  $q$  and its head is seeing the character  $c$ , then  $\delta_M(q, c) = \langle q', c', direction \rangle$  specifies  $M$ 's next state and next actions.

Is it possible that  $M_{universal}$  "knows"  $q$ , or  $c$ , or for that matter  $\delta_M$ ?

- Answer: No,  $M$  may have many more states than  $M_{universal}$ , hence with  $M_{universal}$ 's states it can't know  $q$ . Similarly,  $\Sigma_M$  may be very large and hence  $M_{universal}$  can't know  $c$  either.  $M_{universal}$  definitely can't store in its states all of  $\delta_M$ , which is part of  $M_{universal}$ 's input.

(d) Quickly sketch the actions taken by  $M_{universal}$ 's simulation of the  $t_M^{th}$  time step of  $M$ 's computation.

- Answer: As said  $M_{universal}$  can't "know" either the state  $q$  that  $M$  is in or the character  $c$  that it is looking at. However, this information is stored on  $M_{universal}$ 's tape so  $M_{universal}$  can access it by moving its head back and forth.

$M_{universal}$ 's second head will be pointing at the beginning of the tuple  $\langle "c" \rangle$  corresponding to character  $c$  in the cell that  $M$ 's head is pointing at.

$M_{universal}$ 's third tape will contain " $q$ ", where  $q \in Q_M$  is the current state of  $M$ .

$M_{universal}$ 's first head will scan its first tape looking for the matching tuple  $\langle "q", "c", "q", "c", "direction" \rangle$ .

As said,  $M_{universal}$  will have to move its third head back and forth to compare what is on its first tape with " $q$ " on its third tape. Similarly, for " $c$ " on its second tape.

$M_{universal}$  will then copy the tuple  $\langle "c" \rangle$  of length  $r_M$  into the cells of its second tape previously occupied by  $\langle "c" \rangle$ , copies the encoding " $q$ " onto its third tape and finally moves its head on its second tape in the required direction by one tuple.

(e) Let  $T(|I|)$  denote the running time of  $M$  on  $I$  and let  $|"M"|$  denote length of the description of  $M$  stored on  $M_{universal}$ 's first tape.

What is the running time needed by  $M_{universal}$  to simulate  $M(I)$ ?

- Answer: With care, each of  $M$ 's time steps can be simulated in  $\mathcal{O}(|"M"|)$  time, because the main work is scanning the description " $M$ " on the  $M_{universal}$ 's first tape. Hence the total time will be  $\mathcal{O}(|"M"| \times T(|I|))$ .

(f) It is not fair that  $M_{universal}$  is allowed three tapes but  $M$  only one. Suppose instead,  $M_{universal}$  was only allowed one tape.

Remember that a one tape TM  $M_{universal}$  is able to simulate our three tape  $M_{universal}$  by interweaving the cells of the three tapes on one tape or by having a larger tape alphabet  $\Sigma_{universal}$  to include characters like  $'0 \langle '1'$  to signify that  $'0'$  is on its first tape  $' \langle '$  on its second and  $'1'$  on its third.

The challenge is that the one tape version of  $M_{universal}$  has one not three heads.

Show how the straight forward implementation requires  $[\mathcal{O}(|"M"|) + \mathcal{O}(T(|I|))] \times T(|I|)$  time.

Show how this can be improved to only  $\mathcal{O}(|"M"|) \times T(|I|)$  time with the smart trick of moving the " $M$ " and the " $q$ ".

- Answer: If  $M$  runs for  $T(|I|)$  time steps, its tape may have  $T(|I|)$  characters on it and hence its description of length  $|"c"| \times T(|I|)$ . In the straight forward implementation, for each step

of  $M$ , the simulator must move its head from where it is in this long description to where “ $M$ ” is described and then search through “ $M$ ”. This takes  $[O(|“M”|) + O(T(|I|))]$  time per step of  $M$ .

When “ $M$ ” and “ $q$ ” are considered to be of constant length wrt the length of the input  $I$ , it is best to have shifted them along with the head. Hence, when  $M$  has its head on a given cell containing  $c$ ,  $M_{universal}$ ’s head will be on the corresponding tuple containing “ $c$ ” and the descriptions “ $M$ ” and “ $q$ ” are in the same basic place. Note that as  $M$ ’s head moves one cell to the left or right, “ $M$ ” and “ $q$ ” can be shifted by one tuple of cells. This takes the same time  $|“M”|$  needed to search in “ $M$ ”.

## 7. (Solution in Slides)

Think carefully about the definitions of Turing Machines. For every state  $q$  that the machine might currently be in and character  $c$  that might be written in the cell of the tape pointed to by the head, we define  $\delta(q, c) = \langle q', c', direction \rangle$  where  $q'$  is the next state,  $c'$  is the character to write, and  $direction$  is the direction in which to move the head one cell.

For each of these three specified changes to the TM model, explain the effect it would have on the reasonableness and on the computational power of the model. Do one of:

- A: Either argue that the set of problems computable does not change because the new and the old versions of TM can simulate each other
- B: or show that the new model is weaker by giving a computable problem that no longer can be computed
- C: or show that the new model is stronger by giving an easy algorithm for an arbitrary uncomputable problem (for example the halting problem).

- (a) Instead of requiring the characters  $c$  that are written in each cell of the tape to be from a set of a fixed size (eg binary or ascii), allow the  $c$  and the  $c'$  in the above definition  $\delta(q, c) = \langle q', c', direction \rangle$  to be arbitrarily large integers.

- Answer: The model would no longer be reasonable because then  $\delta$  would need to be defined for an infinite number of pairs  $\langle q, c \rangle$  as such the machine would require an infinite amount of space to specify. Such a machine could compute any function, even the halting program. It would express the entire arbitrarily long input  $I$  as a single integer  $c$  in a single cell and then have  $\delta$  specify in one time step the answer.

- (b) Change the model so that the TM “knows” the location of the head. More formally,  $\delta(q, i, c) = \langle q', c', direction \rangle$  also depends on the index  $i$  of the cell that head is currently at.

- Answer: Again, the model would no longer be reasonable because then  $\delta$  would need to be defined for an infinite number of tuples  $\langle q, i, c \rangle$ . Such a machine could compute any function, even the halting program. It would express the entire arbitrarily long input  $I$  as the location  $i$  of the head and then have  $\delta$  specify in one time step the answer.

- (c) Change the model so that the TM no longer has a head but can specify the index of a cell of the tape to read and to write to. The TM still has its input stored in the first  $n$  cells of the input and is only allowed a fixed/constant/finite number of states and alphabet size. Formally,  $\delta(q, c) = \langle q', c', i, i' \rangle$  where  $c$  is the last value read,  $c'$  will be written to cell indexed by  $i$  and the next  $c$  will be read from that indexed by  $i'$ .

- Answer: Such a machine could not compute very much. Because there is only a constant number of tuples  $\langle q, c \rangle$ , there is only a constant number of indexes  $i$  and  $i'$  that could ever get addressed. The characters of the input  $I$  that are stored in other cells could never be read and hence the actions of the TM could not depend on them.

8. Eric Ruppert had a contest to make as a TM with only 6 states that starting with string 11111100000 runs for as long as possible. I gave him a solution with only 5 states that ran for 3,983,884 time steps.

My idea is think of the contents of the tape in binary and to count down and to extend the length of the used tape by one each decrement. However, we flip the role of zeros and ones. And we flip the order of the bits from left to right.

Our initial tape is to contain #11111100000. (Here # indicates the start of the tape)

Changing zeros to ones and ones to zeros and flipping the order of the bits from left to right gives 11111000000#, which is 1984 in binary.

Each meta step will decrement this number by one and add a leading zero.

The loop invariant is that after the  $i^{\text{th}}$  meta step, the number in binary will be  $1984-i$  with  $i$  leading zeros.

For example,

```
i=0: _____11111000000# = 1984 with 0 leading zeroes
i=1: _____011110111111# = 1983 with 1 leading zeroes
i=2: _____0011110111110# = 1982 with 2 leading zeroes
i=3: _____00011110111101# = 1981 with 3 leading zeroes
i=4: _____000011110111100# = 1980 with 4 leading zeroes
i=5: _____0000011110111011# = 1979 with 5 leading zeroes
```

Changing ones back to zeros and zeros back to ones and flipping the order of the bits back again from right to left then gives

the actual tape content to be

```
i=0: #11111100000_____
i=1: #000000100001_____
i=2: #1000001000011_____
i=3: #10000010000111_____
i=4: #110000100001111_____
i=5: #0010001000011111_____
```

Let  $n = 1984$  be our initial value.

Let  $m = 11$  be the initial number of bits.

The  $i^{\text{th}}$  meta step (going from  $i-1$  to  $i$ )

will change the integer from  $n-i+1$  to  $n-i$

and change the number of bits from  $m+i-1$  to  $m+i$ .

To do this, the algorithm must move the from left to the right and back to the left.

This takes  $2(m+i)$  time.

The last pass of the TM will just go once till it finds the blank.

Hence, the total time is

$$\begin{aligned} & [\sum_{i=1..n} 2(m+i)] + m+n \\ &= 2mn + [\sum_{i=1..n} 2i] + m+n \\ &= 2mn + n(n+1) + m+n \\ &= 2(11)(1984) + (1984)(1985) + 11 + 1984 \\ &= 3,983,883 \end{aligned}$$

To make it easier to understand,

I am first going to write the code with zeros and ones switched and left and right flipped.

Suppose the current state is

i=4: \_\_\_\_000011110111100# = 1980 with 4 leading zeroes  
with the head on the right most bit, i.e., the low order bit (here a 0)  
in state s0

and we must change it to

i=5: \_\_\_\_000011110111011# = 1979 with 5 leading zeroes  
with the head on the right most bit, i.e., the low order bit (here a 1)  
in state s0

Pseudo Code:

Move left changing zeros to ones until you reach the rightmost 1.

Change this one to zero and go one more step left.

The algorithm then moves the head all the way to the left to the first blank and changes it to a 0.

The algorithm then moves the head back all the way from left to right.

This reestablishes the loop invariant.

The code terminates when the integer on the tape is all zero.

Translate this into TM transitions.

- Answer:

Move left changing zeros to ones until you reach the rightmost 1.

Change this one to zero.

$\delta(s_0,0)$  = change to 1, go left, & stay in state s0

$\delta(s_0,1)$  = change to 0, go left, & change to state s1

The algorithm then moves the head all the way to the left to the first blank and changes it to a 0.

$\delta(s_1,0)$  = go left & stay in s1

$\delta(s_1,1)$  = go left & stay in s1

$\delta(s_1,blank)$  = change to 0, go right, & change to state s2

The algorithm then moves the head back all the way from left to right.

$\delta(s_2,0)$  = go right & stay in s2

$\delta(s_2,1)$  = go right & stay in s2

$\delta(s_2,\#)$  = go left & change to state s0

This reestablishes the loop invariant.

The code terminates when the integer on the tape is all zeros.

The following code will then move the head all the way to the left until it gets a blank.

$\delta(s_0,0)$  = change to 1, go left & stay in state s0

Hence the exit condition is

$\delta(s_0,blank)$  = change to the halting state.

Changing ones back to zeros and zeros back to ones

and flipping the order of the bits back again from right to left then gives

Code:

Our initial tape is to contain #11111100000 with the head on the left most 1.

$\delta(s_0,1)$  = change to 0, go right, & stay in state s0

delta(s0,0) = change to 1, go right, & change to state s1  
delta(s1,1) = go right & stay in s1  
delta(s1,0) = go right & stay in s1  
delta(s1,blank) = change to 1, go left, & change to state s2  
delta(s2,1) = go left & stay in s2  
delta(s2,0) = go left & stay in s2  
delta(s2,#) = go right & change to state s0  
delta(s0,blank) = change to the halting state.

If you can give me 2 more states, then I can increase the time by  
another  $2^{(1984+12)} \cdot (1984+12)$   
At the end of this algorithm the integer will be zero with m bits with  
n extra leading zeros  
for a total of m+n zeros.  
But really on the tape is m+n ones.  
As an integer, this is  $2^{(m+n)}-1$ .  
Use the two extra states to decrement this number to zero.