The First Assignment

3190102721 Xu Shengze

Note: Some topics in this homework were completed after discussing with Zhou Yuxin and Zhou Ziyue.

Problem 1.1 Construct a Turing machine that reverses its input (e.g., produces "0010111" from "1110100").

Answer 1.1

We consider a Turing machine M.

$A = \{0, 1\}$ is the external alphabet.

$S = \{\sqcup, 0, 1, *, 0', 1'\}$ is the alphabet, which consists of the symbols of the external alphabet, the empty symbol $\sqcup$, and three auxiliary marks.

$Q = \{q_0, q_f, r_0, r_1, l_0, l_1, l_{0'}, l_{1'}\}$ is the set of states.

And the transition function is $\delta : Q \times S \to Q \times S \times \{1, 0, 1\}$.

Table 1: Operation

δ	0	1	$\sqcup$	$*$	$0'$	$1'$
q_0	$(r_0, *, +1)$	$(r_1, *, +1)$	$(q_0, \sqcup, -1)$		$(q_f, 0, -1)$	$(q_f, 1, -1)$
r_0	$(r_0, 0, +1)$	$(r_0, 1, +1)$	$(l_{0'}, \sqcup, -1)$		$(l_{0'}, 0, -1)$	$(l_{0'}, 1, -1)$
r_1	$(r_1, 0, +1)$	$(r_1, 1, +1)$	$(l_{1'}, \sqcup, -1)$		$(l_1, 0, -1)$	$(l_{1'}, 1, -1)$
l_0	$(l_0, 0, -1)$	$(l_0, 1, -1)$		$(q_0, 0, +1)$		
l_1	$(l_1, 0, -1)$	$(l_1, 1, -1)$		$(q_0, 1, +1)$		
$l_{0'}$	$(l_0, 0', -1)$	$(l_1, 0', -1)$		$(q_f, 0, -1)$		
$l_{1'}$	$(l_0, 1', -1)$	$(l_1, 1', -1)$		$(q_f, 1, -1)$		
q_f						

The transition function is described as follows.

(1) Beginning of work:

$$(q_0, 0) \mapsto (r_0, *, +1), \quad (q_0, 1) \mapsto (r_1, *, +1), \quad (q_0, \sqcup) \mapsto (q_0, \sqcup, -1).$$

The third formula indicates that the entire sequence is empty and should be stopped immediately.

(2) Transfer to the right:

$$(r_0, 0) \mapsto (r_0, 0, +1), \quad (r_1, 0) \mapsto (r_1, 0, +1),$$

$$(r_0, 1) \mapsto (r_0, 1, +1), \quad (r_1, 1) \mapsto (r_1, 1, +1).$$

The machine moves to the right until it encounters the ned of the input string or a mark.

(3) A change in the direction of motion from right to left:

$$(r_0, 0') \mapsto (l_{0'}, 0, -1), \quad (r_1, 0') \mapsto (l_1, 0, -1),$$

$$(r_0, 1') \mapsto (l_{0'}, 1, -1), \quad (r_1, 1') \mapsto (l_{1'}, 1, -1),$$

$$(r_0, \sqcup) \mapsto (l_{0'}, \sqcup, -1), \quad (r_1, \sqcup) \mapsto (l_{1'}, \sqcup, -1),$$

$$(l_{0'}, 0) \mapsto (l_0, 0', -1), \quad (l_{1'}, 0) \mapsto (l_0, 1', -1),$$

$$(l_{0'}, 1) \mapsto (l_1, 0', -1), \quad (l_{1'}, 1) \mapsto (l_1, 1', -1).$$

It consists of two actions: remove the mark and place it in the left adjacent position.

(4) Transfer to the left:

$$(l_0, 0) \mapsto (l_0, 0, -1), \quad (l_1, 0) \mapsto (l_1, 0, -1),$$

$$(l_0, 1) \mapsto (l_0, 1, -1), \quad (l_1, 1) \mapsto (l_1, 1, -1).$$

(5) Change of direction from left to right:

$$(l_0, *) \mapsto (q_0, 0, +1), \quad (l_1, *) \mapsto (q_0, 1, +1).$$

(6) Completion of work:

$$(q_0, 0') \mapsto (q_f, 0, -1), \quad (q_0, 1') \mapsto (q_f, 1, -1),$$

$$(l_{0'}, *) \mapsto (q_f, 0, -1), \quad (l_{1'}, *) \mapsto (q_f, 1, -1).$$

The transition function is undefined for the state q_f, therefore the machine stops after switching to this state.

Problem 1.2 Construct a Turing machine that adds two numbers written in binary. (Assume that the numbers are seperated by a special symbol "+" that belongs to the external alphabet of the TM.)

Answer 1.2

We consider a Turing machine M.

$A = \{0, 1, +\}$ is the external alphabet.

$S = \{\sqcup, 0, 1, +, *, 0', 1'\}$ is the alphabet.

$Q = \{q_0, q_f, r_0, r_1, r_\sqcup, l_0, l_1, l_+, l_{0'}, l_{1'}, l_{+'}\}$ is the set of states.

And the transition function is $\delta : Q \times S \to Q \times S \times \{-1, 0, +1\}$.

Table 2: Operation

δ	0	1	$\sqcup$	$+$	$*$	$0'$	$1'$
q_0	$(q_0, 0, +1)$	$(q_0, 1, +1)$	$(q_f, \sqcup, 0)$	$(l_+, +, -1)$			
r_0	$(r_0, 0, +1)$	$(r_0, 1, +1)$	$(l_{0'}, \sqcup, -1)$	$(r_0, +, +1)$	$(r_0, *, +1)$	$(l_{0'}, 0, -1)$	$(l_{0'}, 1, -1)$
r_1	$(r_1, 0, +1)$	$(r_1, 1, +1)$	$(l_{1'}, \sqcup, -1)$	$(r_1, +, +1)$	$(r_1, *, +1)$	$(l_{1'}, 0, -1)$	$(l_{1'}, 1, -1)$
$r_\sqcup$	$(r_\sqcup, 0, +1)$	$(r_\sqcup, 1, +1)$		$(r_\sqcup, \sqcup, +1)$	$(r_\sqcup, \sqcup, +1)$	$(q_f, 0, 0)$	$(q_f, 1, 0)$
l_0	$(l_0, 0, -1)$	$(l_0, 1, -1)$		$(l_+, +, -1)$			
l_1	$(l_1, 1, -1)$	$(l_1, 0, -1)$		$(l_{+'}, 1, -1)$			
l_+	$(r_0, *, +1)$	$(r_1, *, +1)$	$(r_\sqcup, \sqcup, +1)$		$(l_+, *, -1)$		
$l_{0'}$	$(l_0, 0', -1)$	$(l_0, 1', -1)$		$(l_{+'}, 0', -1)$			
$l_{1'}$	$(l_0, 1', -1)$	$(l_1, 0', -1)$		$(l_{+'}, 1', -1)$			

The transition function δ is used to help implement the function of addition. Depending on the state and symbol, the result is different. The specific implementation process is similar to **Problem 1.1**.

Problem 1.3 ("The halting problem is undecidable"). Prove that there is no algorithm that determines, for given Turing machine and input string, whether the machine terminates at that input or not.

Answer 1.3

The proof is by contradiction.

Suppose that there is such an algorithm, i.e., that there exists a machine B which, for the input $([M], x)$, givesthe answer "yes'' if the machine M stops at input x and gives the answer "n'' otherwise.(Recall that $[M]$ denotes a description of the machine M.)

Algorithm 1 Turing machine B

Input: $([M], x)$
Output: answer
  if M stops at input x **then**
    answer←yes;
  else
    answer←no;
  end if

Let us define another machine B' that, given an input y, simulates thework of B for the input (y, y). If the answer of the machine B is "yes", thenB begins moving the head to the right and does not stop. If the answer of B is "no", then B' stops.

Algorithm 2 Turing machine B'

Input: $B(y, y)$
  if $B(y, y)$=yes **then**
    run forever;
  else
    stop;
  end if

Let us consider this proposition: does B' stop for the input $[B']$?

If $B([B', B']) = yes$, it implies that B' halts, but it contradicts with the assumption that B' runs forever.

If $B([B', B']) = no$, it implies that B' doesn't halt, but it contradicts with the assumption that B' halts.

Based on the above discussion and analysis, we find that the results obtained by our hypothesis are contradictory. So the hypothesis doesn't work, there is no algorithm that determines,for given Turing machine and input string, whether the machine terminates at that input or not.

Problem 1.4 Prove that there is no algorithm that enumerates all Turing machines that do not halt when started with the empty tape.

(Informally, enumeration is a process which produces one element of a set after another so that every element is included in this list. Exact definition: a set $X \subseteq A^*$ is called enumerable if it is the set of all possible outputs of some Turing machine E.)

Answer 1.4

Firstly, we need to show that the elements of an enumerable set canactually be produced one by one by an algorithmic process. Suppose that X is the set of all possible outputs of a Turing machine E. Let us try allpairs of the form (x, n) (where x is a string, and n is a natural number) andsimulate the first n steps of E for the input x. If E terminates during thefirst n steps we include its output in the list; otherwise we proceed to thenext pair (x, n). This way all elements of X are included (possibly, with repetitions).

Lemma A partial function $F : A^* \to \{0, 1\}$ is computable if and only if the sets $X_0 = \{x : F(x) = 0\}$ and $X_1 = \{x : F(x) = 1\}$ are both enumerable.

Proof Suppose that X_0 and X_1 are enumerable. Given an input string $y \in X_0 \cup X_1$, we run the enumerating processes for X_0 and X_1 in parallel. Sooner or later, y will be produced by one of the processes. If it is the process for X_0 , we announce 0 as the result, otherwise the result is 1.

Conversely, if F is computable, then there is a TM that presents its input x as the output if $F(x) = 0$, and runs forever if $F(x)$ is 1 or undefined. Therefore X_0 is enumerable. (Similarly, X_1 is enumerable.)

$$X_0 = [M] : M \text{ does halt for the empty input}$$
$$X_1 = [M] : M \text{ halts for the empty input}$$

Algorithm 3 Turing machine E

Input: $x = [M]$
Output: *answer*
  if M stops **then**
    answer←x ;
  else
    run forever;
  end if

If X_0 is enumerable, there would exist an algorithm for determining whether a given Turing machine M halts for the empty input. We have supposed X_0 is enumerable, and we can conclude that X_1 is enumerable.

We define F as the predicate to decide whether a Turing machine M halts for the empty input or not. Then X_0 and X_1 defined above are the according sets of F. According to the Lemma, as X_0 and X_1 are enumerable, F is decidable.

Then there exists a machine, an algorithm that determines whether a Turing machine halts for the empty input. But it contradicts with the conclusion of **Problem 1.3**. So the hypothesis dosen't work, there is no algorithm that enumerates all Turing machines that do not halt when started with the empty tape.

Problem 1.5 Let $T(n)$ be the maximum number of steps performed by a Turing machine with $\leq n$ states and $\leq n$ symbols before it terminates starting with the empty tape. Prove that the function $T(n)$ grows faster than any computable total function $b(n)$, i.e., $\lim_{n\to\infty} T(n)/b(n) = \infty$.

Answer 1.5

Assume that $b(n)$ is an arbitrary computable function $b : \{0,1\}^* \to \{0,1\}^*$, so there exists a Turing machine M_b to compute b(n). This machine has a constant number of states and a constant number of symbols, which we call p_1 and q_1.

For any input length n, there exists a machine M_n that writes n on the tape. Taking binary representation for example, it only needs $\log n$ cells to write. So this machine has a constant number of symbols, denoted as q_2, and at most $O(\log n)$ states.

Based on M_b and M_n, we construct a new Turing machine.

First, with the length n, it writes $b(n)$ and n on the tape, using at most $p_1 + O(\log n)$ states and $q_1 + q_2$ symbols.

Next, we multiply n and $b(n)$ to get $nb(n)$. As n is expressed within $O(\log n)$ cells, each cell representing a state to do the multiply, the multiplication need at most $O(\log n)$ states and a constant number of symbols, denoted as q_3.

At last, the machine do the decrement to $nb(n)$, which means counting down from $nb(n)$ to 0. This operation just a constant number of states, denoted as p_4, and a constant number of symbols, denoted as q_4. By the way, the machine has been done, we call it M.

Now let's analyze M. In the process, it counts down from $nb(n)$ to 0, so the number of steps performed by it is at least $nb(n)$. Meanwhile, M has $p_1+O(\log n)+O(\log n)+p_4 = O(\log n)$ states, and $q_1 + q_2 + q_3 + q_4 = O(1)$ symbols.

When n is large enough, M is a Turing machine with $\leq n$ states and $\leq n$ symbols. The steps of M is at least $nb(n)$, indicating that $T(n) \geq nb(n)$. So for any computable total function $b(n)$, when $n \to \infty$, $T(n)/b(n) \geq n$, $\lim_{n\to\infty} T(n)/b(n) = \lim_{n\to\infty} n = \infty$.

Problem 1.6 Prove that a 2-tape Turing machine working in time $T(n)$ for inputs of length n can be simulated by an ordinary Turing machine working in time $O(T^2(n))$.

Answer 1.6

We describe briefly a single-tape machine M_1 that simulates a two-tape machine M_2. The alphabet of M_1 is rather large: one symbol encodesfour symbols of M2, as well as four additional bits. The symbol in the k-th cell of M_1 represents the symbols in the k-th cells on the input tape, theoutput tape and both work tapes of M_2; the additional bits are used to markthe places where the heads of M_2 are located. The control device of M_1 keeps the state of the control device of M_2 and some additional information.

The symbol set of M_2 is called S, then we use symbol set $S_1 = S \cup S_2$ for M_1, where $S_2 = (S \times \{X, \sqcup\})^4$ and $X \notin S$.

The machine M_1 works in cycles. Each cycle imitates a single step of M_2.Suppose the start position of the four tapes correspond to the same cell in M_1, for the k-th step of M_2, M_1 has used at most $2k + 1$ cells.

At the beginning of each cycle the head of M_1 is located above theleftmost cell. Each cycle consists of two passes. First Mi moves from left toright until it finds all of the four marks; the contentsof the corresponding cells are stored in the control device. On the way backactions imitating one step of M_2 are carried out. Each such action requires $O(1)$ steps, thus the process needs $O(2k + 1) + O(1) + O(2k + 1) = O(k)$ steps.

So if M_2 works in time $T(n)$, M_1 need to work in $\sum_{k=1}^{T(n)} O(k) = O(\sum_{k=1}^{T(n)} k) = O(T^2(n))$ time.

Problem 1.7 Prove that a 3-tape Turing machine working in time $T(n)$ for inputs of length n can be simulated by a 2-tape machine working in time $O(T(n)\log T(n))$.

Answer 1.7

The main problem in efficient simulation of a multitape TM on an ordinary TM is that the heads of the simulated machine may be farfrom each other. Therefore the simulating head must move back and forthbetween them to imitate a single step of the multitape TM.

We denote the 2-tape ordinary machine as M_2, and the 3-tape Turing machine as M_3. In order to simulate M_3 on M_2, we need to use recursion and multilevelcaching.

Suppose we have already simulated $t = 2^k$ steps of M_3 on M_2 in $T(t)$ steps, we hope to continue to simulate another t steps. Below we consider the case of simulating $2t$ steps on M_2.

The head can move t units at most in t steps, so in order to realize the operation, we only need to copy the t field of the nearest head, which we call t-cache. Next, we give the following operations to complete the target $2t$ step simulation.

1. Copy $2t$-neighborhoods and t-neighborhoods of the heads in M_3 onto the tapes of M_2 as $2t$-cache and t-cache, which we call A and B.
2. Simulate t steps of M_3 on B recursively, then B changes to B'.
3. Copy the result of $t-$cache B' back to the $2t-$cache A, then A changes to A'.
4. Copy t-neighborhoods of the new head positions in A' onto the other work tape of M_2 as the $t-$cache C.
5. Simulate t steps of M_3 in on C recursively, then C changes to C'.
6. Copy the result of changed $t-$cache C' back to the $2t-$cache A', then A' changes to A''.

The length of the sequence is $O(t)$, and various operations also require $O(t)$ steps, so the recursive expression is $T(2t) = 2T(t) + O(t)$.

Through the knowledge of ordinary differential equations, we can assume $O(t) \leq ct$ and substitute in $t = 2^k$. After proper transformation, we can get the recurrence relation: $\frac{T(2^{k+1})}{2^{k+1}} \leq \frac{T(2^k)}{2^k} + \frac{c}{2}$, that is $\frac{T(t)}{t} \leq T(1) + kc$.

Therefore, we get the relation $T(t) \leq c_1 t + c_2 t \log 2(t)$, which implies $T(t) = O(t \log t)$.

Above all, A 3-tape Turing machine working in time $T(n)$ for inputs of length n can be simulated by a 2-tape machine in time $O(T(n)\log T(n))$.

Problem 1.8 Let M be a (single-tape) Turing machine that duplicates the input string (e.g.,produces "blabla" from "bla"). Let T(n) be its maximum running time when processing input strings of length n. Prove that $T(n) \geq \epsilon n2$ for some ϵ and for all n. What can be said about $T'(n)$, the minimum running time for inputs of length n?

Answer 1.8

To copy a string of length n, the simplest way is to move and copy bit by bit. To move a bit, the head should pass through at least n units and then return, which requires $\Omega(n)$ steps. So by this way, the running time is $\Omega(n) \times n = \Omega(n^2)$.

At the same time, the Turing machine has a finite number of states, it carries only $O(1)$ bits by distance 1 at each step. Therefore, if the running time is at least $T(n) = \Omega(n^2)$, then there exists ϵ, for all n, it satisfies $T(n) \geq \epsilon n^2$.

Next, consider $T'(n)$. We need to construct a Turing machine M, which can copy a string in time $\Omega(n \log n)$ under certain circumstances. This situation is usually ideal. First, we need to check whether the string is composed of n identical symbols, and if so, the running time is the shortest.

Without loss of generality, we can assume that this number is 0^n, we need to count it with a Turing machine, and then produce the same number of zeros. In this process, M traversal checks whether the string contains only 0, which requires $O(n)$ steps, and then $O(\log n)$ cells are needed for storage when calculating and saving, so we need at least $O(\log n) \times n = O(n \log n)$ steps. If there are different symbols other than 0, M is copied in $O(n^2)$, but the least is still $O(n \log n)$. Therefore, $T'(n) = O(n \log n)$.

Problem 1.9 Consider a programming language that includes 100 integer variables, the constant 0, increment and decrement statements, and conditions of type "variable = 0". One may use `if - then - else` and `while` constructs, but recursion is not allowed. Prove that any computable function of type $Z \to Z$ has a program in this language.

Answer 1.9

Consider an arbitrary computable function $F : Z \to Z$, there exist a corresponding Turing machine M that can compute it.

The alphabet of M is S. We use the S system to represent the elements in this alphabet, so it can be represented by numbers from 0 to $|S| - 1$, and integers can also be uniquely represented.

We assume that the entire tape of the Turing machine is stored in one cell, and at the same time, it is looking for a cell as the origin. In addition, the position of the head can also be expressed by looking for a variable.

The state of the Turing machine is limited and can be represented by a finite set Q, so the state of the machine at this time can also be represented by a variable in any case.

Therefore, the configuration of the Turing machine at any time can be represented by a triple $\langle \sigma; p; q \rangle$. We only need to prove that the triples can be transformed to each other to prove that the proposition is true.

Consider the transfer function:$\delta : Q \times S \to Q \times S \times \{-1, 0, +1\}$.

We know that the domain of the function is clearly defined, and Q and S are limited, so we can use the `if-then-else` and `while` structures in programming technology to realize the transformation between different states. Because the function type is Z → Z, we need to implement basic digital operations, such as addition, subtraction, multiplication, division, and exponentiation, as well as comparison of numbers.

Comparison of numbers: Using the `if-then-else` structure, we can compare two numbers in advance and store the result in a variable.

Algorithm 4 Comparison of numbers

Input: two integers a,b
Output: *variable*

```
if a ≠ b then
    variable ← 0 ;
else
    variable ← 1;
end if
```

Addition(Subtraction): Use a variable to record the add (subtract), and use the `while` loop to call the increment (subtract) statement.

Algorithm 5 Addition(Subtraction)

Input: a
Output: $answer$
  while $a > 0$ **do**
    $answer \leftarrow answer \pm 1$
    $a \leftarrow a - 1$
  end while

Multiplication: Use a variable to record the multiplier, which can be seen as doing several additions, so use the `while` loop to call the addition statement.

Algorithm 6 Multiplication

Input: a,n
Output: $answer$
  $answer \leftarrow 0$
  while $n > 0$ **do**
    $answer \leftarrow answer + a$
    $n \leftarrow n - 1$
  end while

Exponentiation: Use a variable to record the exponentiation. The exponentiation operation can be regarded as a number of multiplications, so the multiplication statement is called with a `while` loop.

Division and remainder operations: use a variable to record the divisor. Division can be seen as subtraction until it can't be subtracted. The remaining number is the remainder, and a variable is used to record the number of subtractions. This can be achieved using a `while` loop.