Computability and Complexity

Instituto de Matemáticas

Exercises

The Church-Turing Thesis

Exercise 1 : *Understanding the definitions.*
Based on the formal definition of a Turing machine, answer the following questions and justify your answers:

1. Can a Turing machine write the blank symbol ␣ on its tape?
2. Can the tape alphabet Γ and the input alphabet Σ be equal?
3. Can a Turing machine's head be in the same location in two successive configurations?
4. Can a Turing machine contain a single state?

Answer:

1. *Yes, since* $␣ \in \Gamma$*: the blank symbol is in its tape alphabet, and so a Turing machine can write it.*
2. *No, since* $␣ \notin \Sigma$*.*
3. *Yes, if the machine's head was on the leftmost cell and it has to move Left, then it will remain on the leftmost cell.*
4. *No, since a Turing machine necessarily has an accept state and a reject state, so they have to be distinct.*

□

Exercise 2 : *A weird algorithm.*
Explain why the following is not a description of a Turing machine:

$M_{bad} =$ The input is a polynomial p over variables $x_1, \ldots, x_n$.

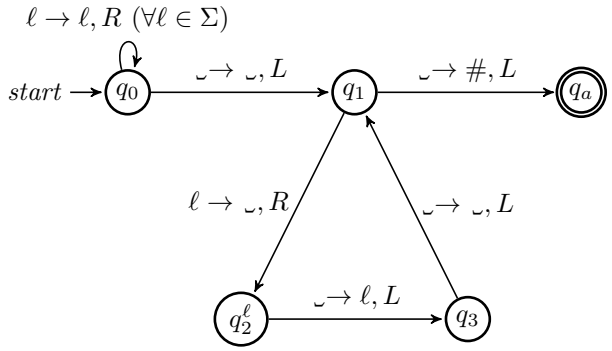
1. Try all possible settings of $x_1, \ldots, x_n$ to integer values.
2. Evaluate p on all those settings.
3. If any of these settings evaluate to 0, *accept*; otherwise, *reject*.

Answer: It is easy to see that step 1 will never finish, since the number of possible settings is infinite. Hence, the machine will never reach step 2, and thus M_{bad} *is not a valid description of a Turing machine.*

□

Exercise 3 : *Shifting the blame.*
Let w be a word over an alphabet Σ such that $\# \notin \Sigma$. Construct a formal-level Turing machine that takes input w and enters the accept state once its tape contains $\#w$. Explain why this machine is useful.
Answer: This machine is very useful, since it allows us to check whether we reach the leftmost cell of the tape or not. Indeed, in a classical Turing machine, as we saw in the course, we need a somewhat complicated construction to do this check. Now, all we have to do is to shift the input to the right and use a special character to mark the leftmost cell of the tape. Thus, bringing the head back to the beginning of the input will be easier.
First we go to the end of the word, then we shift each character to the right. Once we have shifted the first character of the input, the head will remain on the leftmost cell, which will contain the blank character.

We only have to write # and enter the accept state. Note that we need $|\Sigma|$ states q_2^ℓ: one for each symbol $\ell \in \Sigma$.



□

Exercise 4 : *Reusing machines*.
Let w be a word over an alphabet Σ such that $\# \notin \Sigma$ and such that w is of even length and not empty. Give the implementation-level description of a Turing machine that takes input w and enters the accept state once its tape contains the word where # is inserted in the middle of w.
Hint: You may use several tapes, and reuse the machine of Exercise 3.
Answer: The idea is to count every other character in w using a second tape, then to apply the Turing machine constructed in Exercise 3 to the second half of the input.

$M =$ *On input w:*

1. *Scan the first tape, writing a 1 and moving Right in the second tape every other character (so that the second tape contains $\frac{|w|}{2}$ in unary).*
2. *Move the second head to the leftmost cell of the second tape, and move the first head Left one step (so that it is on the last character of the input).*
3. *Do the following as long as the second head reads a 1:*
   A. *Execute the loop $q_1 - q_2^\ell - q_3 - q_1$ of the machine constructed in Exercise 3. Each time the machine loops back to q_1, move the second head Right one step.*
4. *Write a #, then* accept.

□

Exercise 5 : *Languages*.
Give implementation-level descriptions of Turing machines that decide the following languages:

1. $\{w \in \{a, b\}^* \mid w$ contains as many a as $b\}$;
2. $\{a^n b^n c^n \mid n \geq 0\}$;
3. $\{a^n b a^{2n} b a^{3n} \mid n \geq 0\}$.

Draw a formal-level implementation of one of those machines.
Answer:

1. $M =$ *On input string w:*
   1. *Scan the tape and mark the first a that has not been marked. If the end of the input is reached, go to step 3. Otherwise, bring the head back to the leftmost cell of the tape.*
   2. *Scan the tape and mark the first b that has not been marked. If no unmarked b has been found,* reject*. Otherwise, bring the head back to the leftmost cell of the tape, and go back to step 1.*
   3. *Bring the head back to the leftmost cell of the tape. Scan the tape: if there is an unmarked b,* reject*; otherwise,* accept.

2. *We use a machine with two tapes. As seen in the course, this is equivalent to a machine with one tape. The idea is to use the second tape to count n in unary.*

   $M =$ *On input string w:*

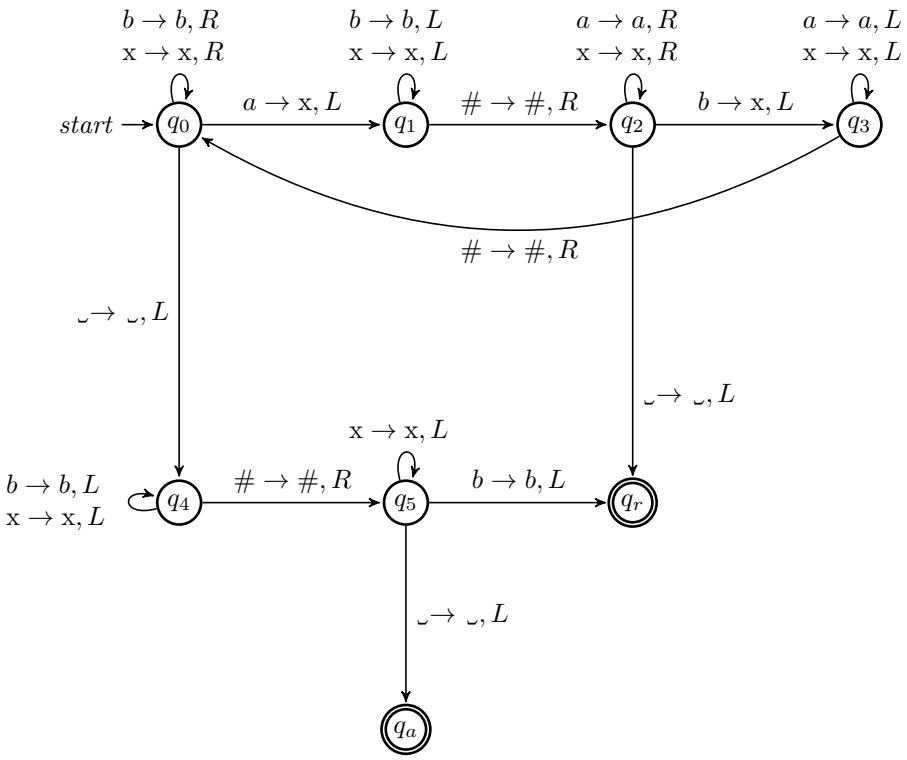
   1. *If the first cell of the first tape contains the blank character,* accept.
   2. *Scan the first tape to verify that the input is in* $a^+b^+c^+$. *Otherwise,* reject.
   3. *Bring the first head back to the leftmost cell of the first tape.*
   4. *Read all the a in the first tape. Each time an a is read, write a 1 in the second tape and move its head right.*
   5. *Bring the second head back to the leftmost cell of the second tape.*
   6. *Read all the b in the first tape. Each time a b is read, move the second head one step right.*
   7. *If the second head is not on a blank state,* reject *(we have less b than a). Otherwise, move the second head one step left. If it is not on a 1,* reject *(we have more b than a).*
   8. *Do steps 5-7, replacing the b by c.*
   9. Accept.

3. *We could again use a second tape to count n in unary, and then make sure that we count twice then thrice the second tape for the second and third series of consecutive a. However, another option is to use three tapes: one will count* $2n$ *and the other one* $3n$. *Again, this is equivalent to having only one tape.*

   $M =$ *On input string w:*

   1. *If the input is bb,* accept.
   2. *Scan the first tape to verify that the input is in* $a^+ba^+ba^+$. *Otherwise,* reject.
   3. *Bring the first head back to the leftmost cell of the first tape.*
   4. *Read all the first consecutive a in the first tape (so, until you reach the first b). Each time an a is read, write a 1 in the second tape and move its head right before writing another 1 and moving again its head right; and on the third tape apply three times the following: write a 1, move the head right.*
   5. *Bring the second head back to the leftmost cell of the second tape.*
   6. *Read all the next consecutive a in the first tape (so, until you reach the second b). Each time an a is read, move the second head one step right.*
   7. *If the second head is not on a blank state,* reject *(we have less than* $2n$ *a). Otherwise, move the second head one step left. If it is not on a 1,* reject *(we have more than* $2n$ *a).*
   8. *Bring the third head back to the leftmost cell of the third tape.*
   9. *Read all the final consecutive a in the first tape (so, until you reach the first blank character). Each time an a is read, move the third head one step right.*
   10. *If the third head is not on a blank state,* reject *(we have less than* $3n$ *a). Otherwise, move the third head one step left. If it is not on a 1,* reject *(we have more than* $3n$ *a).*
   11. Accept.

We draw the formal-level implementation of the first machine. Note that we use a slightly modified input $\#w$, *so that the character # marks the leftmost cell of the tape. See Exercise 3 to see how we can do this (basically we execute the Turing machine constructed in Exercise 3 before executing the following one). We denote* q_{accept} *by* q_a *and* q_{reject} *by* q_r.
In state q_0, *we look for the first uncrossed a. If no such a is found, we go to state* q_4; *otherwise we go to state* q_1. *In state* q_1 *we bring the head back to the leftmost cell. In state* q_2, *we look for the first uncrossed b. If no such b is found, we go the reject state; otherwise we go to state* q_3. *In state* q_3 *we bring the head back to the leftmost cell, before going back to* q_0. *In state* q_4 *we bring the head back to the leftmost cell. Then in state* q_5, *we scan the tape. If we read b, then we go to the reject state, and if we reach the end of the input, then we go to the accept state.*



□

Exercise 6 : *Turing's elementary school.*
Give implementation-level descriptions of Turing machines that compute the following functions (in every case, we assume the numbers are not empty):

1. A function that takes a binary number, and deletes every useless 0 (so every 0 before the first 1);
2. The increment function on binary numbers (the input is a binary number w, and we want to compute $w + 1$);
3. The decrement function on binary numbers (the input is a binary number w, and we want to compute $w - 1$) (assume $w \neq 0$);
4. The binary-to-unary conversion function (the input is a binary number w, and we want to compute the unary number equal to w);
5. The binary addition function (the input is $w_1 \# w_2$ where w_1 and w_2 are binary numbers, and we want to compute $w_1 + w_2$);
6. The binary multiplication function (the input is $w_1 \# w_2$ where w_1 and w_2 are binary numbers, and we want to compute $w_1 w_2$).

You may use several tapes, and reuse machines that you already described or constructed.
Answer:

1. *This machine is easy to construct, and may produce the empty word:*

   $M_0 =$ *On input w:*
   1. *Repeat as long as we do not read a 1 or the blank character:*
      A. Read a 0, remove it, shift the whole input to the left.
   2. Accept.

2. *Here, we first need to shift the input to the right (using the machine in Exercise 3), in case we need one more bit. Then, we go to the right of the word, and change all the 1 to 0 before reaching a 0 that we change to 1. If we have not reached the special character, we reshift the whole input to the left (such a machine is easy to construct by adapting the machine in Exercise 3).*

$M_+ =$ *On input w:*

1. *Shift the input to the right.*
2. *Bring the head to the rightmost character of the input.*
3. *Repeat as long as we do not read a 0 or the special character #:*
   A. *Read a 1, change it to a 0, move Left.*
4. *If the current character is a 0, change it to a 1, shift the whole input to the left, then* accept*. Otherwise, change the character to a 1, then* accept*.*

3. *This is the reverse of the incrementer: we change the 0 to 1 until we reach a 1, that we then change to 0. This time, we do not need to shift the word (maybe a useless 0 will be created, but we have M_0 to take care of that!), since we know that the word is not 0.*

$M_- =$ *On input w:*

1. *Bring the head to the rightmost character of the input.*
2. *Repeat as long as we do not read a 1:*
   A. *Read a 0, change it to a 1, move Left.*
3. *Read a 1, change it to a 0, then* accept*.*

4. *This is very easy to construct by using the decrement machine: just verify if the input is 0, and otherwise we write a new 1 each time we decrement the binary number.*

$M_{unary} =$ *On input w:*

1. *Execute M_0. If, at the end of the execution, we get an empty tape, write 0 and* accept*.*
2. *Shift the input to the right twice, so that we have $\#\#w$. We will write the unary number between the two #.*
3. *Move the head to the first character after the second #. Execute M_0; if, at the end of the execution, we have nothing left at the right of the second #,* accept *(the computation is over).*
4. *Execute M_- on the word at the right of the second #. Move the head to the second #. Shift everything from your position one step right, then go back to the new blank space on the left of the second # and write 1. Go back to step 3.*

5. *For this one, we apply the following algorithm: while $w_2 \neq 0$, increment w_1 and decrement w_2. Good thing we have the machines M_+ and M_-! We could use several tapes, but this is not necessary.*

$M_{add} =$ *On input $w_1\#w_2$:*

1. *Move the head Right until you have reached the first character of w_2.*
2. *Execute M_0, considering the first character of w_2 as the leftmost character of the tape. If, at the end of the execution, w_2 is empty (checked by verifying that we read a blank character on the right of the #),* accept*.*
3. *Execute M_- on w_2. Move the head to the leftmost cell on the tape.*
4. *Execute M_+ on w_1 (this can be done by modifying M_+ such that the head stops going right in step 2 when encountering #).*
5. *Go to step 1.*

6. *Again, we will use a simple algorithm: we add w_1 to itself and decrement w_2 as long as $w_2 \neq 0$. In this case, it could be easier to use several tapes, but we can also simply have a first step where we modify the input.*

$M_{mul} =$ *On input $w_1\#w_2$:*

1. *Modify the input so that we have $0\#w_1\#w_2$ on the tape.*
2. *Move the head Right until you have reached the first character of w_2.*

3. *Execute M_0, considering the first character of w_2 as the leftmost character of the tape. If, at the end of the execution, w_2 is empty (checked by verifying that we read a blank character on the right of the second #),* accept *(the result is then the first part of the input, up until the first #).*
4. *Execute M_- on w_2. Move the head to the leftmost cell on the tape.*
5. *Change the tape (which contains $w\#w_1\#w_2$) into $w\#w_1\#w_1\#w_2$ (we want to save the value of w_1). Execute M_{add} on $w\#w_1$ (again, this can be done by considering that the tape ends when hitting the second #). Now the tape contains $(w+w_1)\#\#w_1\#w_2$, so remove the first #.*
6. *Go to step 2.*

□

Exercise 7 : *Several stacks.*
For this exercise, I assume that you know everything that we saw in the class and exercises on context-free languages.

1. Prove that a pushdown automata with two stacks is more powerful than a pushdown automata with one stack.
2. Prove that you can simulate a Turing machine with a pushdown automata with two stacks.
3. What does that imply for pushdown automatas with more than two stacks?

Answer:

1. *Clearly, a pushdown automata with two stacks recognize all context-free languages (just ignore the second stack). Now we prove that we can recognize more languages using two stacks than using one stack. The language $\{a^n b^n c^n \mid n \geq 0\}$ is not context-free (as was proved in the course), but using two stacks we can easily recognize it. When reading the a, push 1 in both stacks; then when reading the b pop in the first stack; and when reading the c pop in the second stack. By being careful, we can make sure that we recognize exactly this language (so we reject if any stack is nonempty or if a stack is empty but its corresponding characters have not all been read). Thus, pushdown automatas with two stacks are more powerful than basic pushdown automatas.*
2. *The idea is to translate the configuration uqv (so we are in state q with the head on the first character of v in the input) by having the first stack containing u (with the first character at the bottom) and the second stack containing v (with the first character on the top).*

   We initialize the simulation by reading the input, pushing every character in the first stack. Then we pop every character from the first stack, pushing them in the second stack. Thus, the second stack contains the whole input, with the first character on top and the last on the bottom.

   Now, assume that we are in configuration uqv, and with a being the first character of v. If the machine replaces a with b and moves Right, we pop a from the second stack and push b in the first stack. If the machine replaces a with b and moves Left, we replace a by b on top of the second stack, then pop the character on top of the first stack and push it on top of the second stack. It is easy to see that, in both cases, the new configuration is simulated correctly by our pushdown automata with two stacks.
3. *Since Turing machines compute everything that we can compute, and that a pushdown automata with two stacks is equivalent to a Turing machine, it means that adding more stacks does not give more computing power.*

□

Exercise 8 : *The Double Infinity Gauntlet.*
A Turing machine with a doubly-infinite tape is a Turing machine where the tape does not have a left end. If you imagine the tape of a Turing machine as a table with indices in the set of natural numbers $\mathbb{N}$, then the doubly-infinite tape is a table with indices in the set of integers $\mathbb{Z}$. The computation is exactly the same, but the head will never encounter the leftmost end of the tape.

Prove that the Turing machines with a doubly-infinite tape recognizes the class of Turing-recognizable languages.

Answer: The idea here is to use two tapes to simulate a Turing machine with a doubly-infinite tape. Since Turing machines with two tapes recognize the Turing-recognizable languages, this will prove the result.

At the beginning of the computation, the input is on the first tape. Now, we shift the input to the right and add a special symbol on the leftmost cell of the first tape, and do the same for the second tape. Informally, call t the doubly-infinite table representing the tape of the Turing machine with a doubly-infinite tape. The first tape will represent $t[0], t[1], t[2] \ldots$ and the second tape will represent $t[-1], t[-2], \ldots$. We will know that we have to switch tapes when we encounter the special symbol.

In general, the computation in our two-tapes Turing machine is the same than in the Turing machine with a doubly-infinite tape. When we are working on the second tape, we need to revert the Left and Right movements. In both tapes, when we encounter the special symbol, we just change the tape we are working on. More formally:

- *The input alphabet is the same, and the tape alphabet is the same but with a special symbol added.*
- *The accepting and rejecting states are the same.*
- *Every other state is doubled: there will be one state for working on the first tape and one for working on the second tape.*
- *The transitions for the states on the first tape are the same, with an exception: when we move Left, we check if we have reached the special symbol. If this is the case, we go to the equivalent state on the second tape.*
- *The transitions for the states on the second tape are the same, but with the movement of the head reversed (so L become R and R becomes L). Furthermore, when we move Left (so Right in the original Turing machine), we check if we have reached the special symbol, in which case we go to the equivalent state on the first tape.*

It is easy to see that such a machine will recognize the same language as the associated Turing machine with a doubly-infinite tape, proving the result.

□