CS 3719 (Theory of Computation and Algorithms) – Lecture 2

Antonina Kolokolova

January 11, 2012

Last lecture we started describing a Turing machine. Let us now give a formal definition of it.

Definition 12. *Formally, a Turing machine is a 7-tuple* $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$. *Here,* Q *is a finite set of states, with three special states* q_0 *(start state),* q_{accept} *and* q_{reject}. *The last two are called the halting states, and they cannot be equal.* Σ *a finite input alphabet.* Γ *is a tape alphabet which includes all symbols from* Σ *and a special symbol for blank,* $\sqcup$. *Finally, the transition function is* $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ *where* L, R *mean move left or right one step on the tape.*

The Turing machine M works as follows on input string $x \in \Sigma^*$.
Initially x appears on the tape starting from its left end, with infinitely many blanks to the right, and with the head pointing to the leftmost symbol of x. (If x is the empty string, then the tape is completely blank and the head is pointing to the leftmost square.) The control is initially in state q_0.
M moves according to the transition function δ.
M may run forever, but if it halts, then it halts either in state q_{accept} or q_{reject}. (We will mainly be interested in whether M accepts or rejects; there are definitions of what it means for a Turing machine to output a string: e.g., the output is what is left on the tape when the machine halts)

Definition 13. M accepts *a string* $x \in \Sigma^*$ *if* M *with input* x *eventually halts in state* q_{accept}. *We write* $\mathcal{L}(M) = \{x \in \Sigma^* | M \text{ accepts } x\}$, *and we refer to* $\mathcal{L}(M)$ *as the language accepted by* M.

(Notice that we are abusing notation, since we refer both to a Turing machine accepting a string, and to a Turing machine accepting a language, namely the set of strings it accepts.)

Example 1. PARITY= the set of all binary strings that have an even number of 1s $= \{x | x \in \{0,1\}^*, x \text{ has even number of occurrences of symbol } 1\}$.

A very simple Turing machine recognizes the set of such strings in time linear in the number of symbols in the strings. For that, it moves once through the string from left to right, at each point remembering whether the number of 1s it has seen so far is even or odd. If by the time it reaches a blank symbol it has seen an even number of 1s, the Turing machine accepts, and if it has seen an odd number of 1s, it rejects.

State q	*Symbol s*	*Action* $\delta(q,s)$
q_0	0	$(q_0, \not b, R)$
q_0	1	$(q_1, \not b, R)$
q_0	$\not b$	$(q_{accept}, \not b, L)$
q_1	0	$(q_1, \not b, R)$
q_1	1	$(q_0, \not b, R)$
q_1	$\not b$	$(q_{reject}, \not b, L)$

A more complicated example is a Turing machine recognizing the set of palindromes.

Example 2. PAL = the set of even length palindromes = $\{yy^r | y \in \{0,1\}^*$, where y^r means y spelled backwards.

We will design a Turing machine M that accepts the language PAL$\subseteq \{0,1\}^*$. M will have input alphabet $\Sigma = \{0,1\}$, and tape alphabet $\Gamma = \{0,1,\not b\}$. (Usually it is convenient to let Γ have a number of extra symbols in it, but we don't need to for this simple example.) We will have state set $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$.

If, in state q_0, M reads 0, then that symbol is replaced by a blank, and the machine enters state q_1; the role of q_1 is to go to the right until the first blank, remembering that the symbol 0 has most recently been erased; upon finding that blank, M enters state q_2 and goes left one square looking for symbol 0; if 0 is not there then we *reject*, but if 0 is there, then we erase it and enter state q_3 and go left until the first blank; we then goes right one square, enter state q_0, and continue as before.
If we read 1 in state q_0, then we operate in a manner similar to that above, using states q_4 and q_5 instead of q_1 and q_2.
If we read $\not b$ in state q_0, this means that all the characters of the input have been checked, and so we accept.
Formally, the transition function δ is as follows. (Note that when we accept or reject, it doesn't matter what we print or what direction we move in; we arbitrarily choose to print $\not b$ and move right in these cases.)

State q	*Symbol s*	*Action* $\delta(q,s)$
q_0	0	$(q_1, \not b, R)$
q_1	0	$(q_1, 0, R)$
q_1	1	$(q_1, 1, R)$
q_1	$\not b$	$(q_2, \not b, L)$
q_2	1	$(q_{reject}, 0, R)$
q_2	$\not b$	$(q_{reject}, 0, R)$
q_2	0	$(q_3, \not b, L)$
q_3	0	$(q_3, 0, L)$
q_3	1	$(q_3, 1, L)$
q_3	$\not b$	$(q_0, \not b, R)$
q_0	1	$(q_4, \not b, R)$
q_4	0	$(q_4, 0, R)$
q_4	1	$(q_4, 1, R)$
q_4	$\not b$	$(q_5, \not b, L)$
q_5	0	$(q_{reject}, 0, R)$
q_5	$\not b$	$(q_{reject}, 0, R)$
q_5	1	$(q_3, \not b, L)$
q_0	$\not b$	$(q_{accept}, 0, R)$

Even though we will mainly talk about Turing machines just accepting or rejecting a string, it is possible to define a Turing machine producing an output. In that case, the Turing machine

halts in an accepts state, with the tape clear except for the output and head pointing to the first symbol of the output.

Example 3 (+1 operation)**.** Here we will show a Turing machine M which takes as its input a string in binary and adds 1 to it. That is, it will halt in an accept state pointing to the first non-empty symbol, and the content of the tape will be input plus 1.

$M = (Q, \{0,1\}, \{0,1,\sqcup\}, \delta, q_0, q_{accept}, q_{reject})$.

To add 1 to a binary number, the usual algorithm is to start from the last bit, add 1 to it, and propagate the possible carry as much as needed, to the closest 0 or blank from the end. For example, to add 1 to string 1011 we need to propagate the carry to the second symbol, 0, while flipping the 1s.

	0	1	$\sqcup$
q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	$(q_c, \sqcup, L)$
q_c	$(q_n, 1, L)$	$(q_c, 0, L)$	$(q_{accept}, 1, R)$
q_n	$(q_n, 0, L)$	$(q_n, 1, L)$	$(q_{accept}, \sqcup, R)$

Our Turing machine implements this algorithm as follows. It starts by scanning the input to the end of the input string, staying in q_0. After that, it starts moving backward, using two states: a "carry" q_c state and "no carry" state q_n. In the "carry" state, it flips a bit, and if the bit was a 1, remains in the carry state, otherwise goes to non-carry. In no carry state it scans back to the start of the string and there goes to the accept state.

Definition 14. *Let us define more formally what is a* computation *of a Turing machine. To describe a computation of a Turing machine we need to specify, at every point in time, its state, head position, and (non-blank) content of the tape. This information we will call a* configuration *of a Turing machine. For convenience, we will write a configuration as the string that is the sequence of non-blank symbols of the tape (assuming there are no blanks between non-blanks), with the symbol for the current state in a position right before the cell it points to. For example, the starting configuration of a Turing machine on input 0011 is* $q_0 0011$*; after a transition* $(q_0, 0) \to (q_5, 1, R)$ *the machine goes to the configuration* $1q_5011$*.*

Now, a Turing machine M accepts *a string* w *if there is a sequence of configurations starting from* $q_0 w$ *and ending in a configuration containing* q_{accept}*, with every configuration in the sequence resulting from a previous one by a transition in* δ *of* M*.*

So far we have looked at Turing machines for which for any state+symbol pair (q_i, a) there is only one possible transition: e.g., $(q_i, a) \to (q_j, b, \{L, R\})$ (where $\{L, R\}$ means left or right, whichever the transition specifies). If this rule holds, then the Turing machine is *deterministic*. Otherwise, it is called non-deterministic. The idea is that a non-deterministic Turing machine can pick any of the possible transitions; as long as at least one ssequence of choices leads to an accepting configuration, it is said to accept. That is, rather than talking about a single computation here we have a "tree" of computations, where each branching point corresponds to a choice of a transition, and each path from the root (starting

configuration) to a leaf (halting configuration) is a computation path. The non-deterministic Turing machine accepts if there exists at least one accepting computation.

Example 4. Consider the following non-deterministic Turing machine which tries to determine if its input contains a symbol 1. It will scan the input from left to right; when it sees a 1, it would either accept or continue scanning as if nothing has happened.

State q	*Symbol* s	*Action* $\delta(q, s)$
q_0	0	$(q_0, 0, R)$
q_0	1	$(q_0, 1, R)$
q_0	1	$(q_{accept}, \not\flat, R)$
q_0	$\not\flat$	$(q_{reject}, \not\flat, R)$

Now, on input 1101, there are four possible computation paths:

1) $q_0 1101 \rightarrow 1 q_a 101$
2) $q_0 1101 \rightarrow 1 q_0 101 \rightarrow 11 q_a 01$
3) $q_0 1101 \rightarrow 1 q_0 101 \rightarrow 11 q_0 01 \rightarrow 110 q_0 1 \rightarrow 1101 q_a$
4) $q_0 1101 \rightarrow 1 q_0 101 \rightarrow 11 q_0 01 \rightarrow 110 q_0 1 \rightarrow 1101 q_0 \rightarrow 1101 q_r$

Since at least one of them (three, actually) is accepting, this Turing machine accepts 1101.