CS 3719 (Theory of Computation and Algorithms) – Lecture 3

Antonina Kolokolova [*]

January 14, 2013

Last lecture we did a simple example of a Turing machine recognizing a language of all binary strings that recognize an even number of $1s$ (which we called PARITY). Note that our Turing machine never used the tape for anything other than reading input bits in order; it never even needed to revisit a bit and read it a second time. A model of computation that we will look at closer to the end of this course, a Finite Automaton (or Finite State Machine), is just that, a "Turing machine without a tape". In that case, there are still states, input alphabet and the transition table; transitions are of the form $(q_i, a) \to q_j$ (so there is no "writing" or "moving" part, and a is the next input symbol). The description also specifies the start state; rather than having a single accept and a single reject state, some states are marked as accepting and the rest rejecting.

For example, such finite automaton for the PARITY would have just two states: one, call it q_{even}, codes that the automaton has seen an even number of 1s so far, and the other, q_{odd}, that it has seen an odd number of 1s, respectively. The automaton starts in state q_{even} (which also happens to be the only accepting state); every time 1 is read it switches between the two states; when a 0 is read, it stays in the same state. If it ended up in q_{odd}, it must have seen an odd number of 1s, and otherwise an even number.

A more complicated example is a Turing machine recognizing the set of palindromes. This language, though still fairly simple, is not doable by a Finite Automaton described above; we will actually prove it by the end of the course.

Example 1. 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 $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$.

[*]The material in this set of notes came from many sources, in particular "Introduction to Theory of Computation" by Sipser and course notes of U. of Toronto CS 364. Special thank you to Richard Bajona for sharing scribed lecture notes.

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 2 (+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.