CS 3719 (Theory of Computation and Algorithms) – Lecture 16

Antonina Kolokolova*

February 11, 2011

Let us define more formally what is a *computation* of a Turing machine. Recall that for regular languages we defined a computation as a sequence of states; for grammars we talked about derivations. 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.

A Turing machine M *recognizes* a language L if it accepts all and only strings in L: that is, $\forall x \in \Sigma^*$, M accepts x iff $x \in L$. As before, we write $\mathcal{L}(M)$ for the language accepted by M.

Definition 12. *A language L is called* Turing-recognizable *(also* recursively enumerable, r.e*, or* semi-decidable*) if $\exists$ a Turing machine M such that $\mathcal{L}(M) = L$.*

A language L is called decidable *(or* recursive*) if $\exists$ a Turing machine M such that $\mathcal{L}(M) = L$, and additionally, M halts on all inputs $x \in \Sigma^*$. That is, on every string M either enters the state q_{accept} or q_{reject} in some point in computation.*

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.

Remark 1. One reason that these languages are called "recursively enumerable" is that it is possible to "enumerate" all strings in such a language by a special type of Turing machine, enumerator. This Turing machine starts on blank input and runs forever; as it runs, it

*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.

outputs (e.g., on some additional tape or part of the main tape) all the strings in the language. It is allowed to print the same string multiple times, as long as it does not print anything not in the language and eventually print every string that is in the language. See Sipser's book for a formal definition and a proof of equivalence.

Example 1 (+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.

0.1 One-way vs. 2-way infinite tape

There are two ways of defining a Turing machine's tape. Either we assume that the tape is infinite in both left and right direction (like in the example before), or, as in Sipser's book, we allow the tape to be infinite only to the right, and require the input to start in the very first cell of the tape. Here we will show that it does not matter: both definitions are equivalent. That is, any Turing machine with 1–way infinite tape can be transformed into a Turing machine with two-way infinite tape, and vice verse.

Lemma 13. *For every TM M with a one-way infinite tape there exists a TM M' with two-way infinite tape accepting the same language.*

Proof. A Turing machine with one-way infinite tape behaves exactly like a two-way infinite tape TM, except when it is in its rightmost cell then any transition that tries to move left stays in the same place instead. We will simulate it as follows. Using a similar idea to marking the bottom of the stack in a pushdown automaton, start by putting a special symbol \$ at the blank cell to the left of the input. Now, every time M' moves onto this cell it has to come back to the previous place, remembering the state it was in. So, we add transitions $(q_i, \$) \to (q_i, \$, R)$ for every state $q_i \in Q$. Also, we will start in a state q_0' with new transitions $(q_0', a) \to (q_0', a, L)$ $\forall a \in \Sigma$, and $(q_0', \sqcup) \to (q_0, \$, R)$.

Finally, if this is a Turing machine producing output then it matters in which position its head is when it accepts/rejects its input; in this case, we need special treatment for the halting states. So the q_{accept} and q_{reject} of M would not be the halting states of M': instead, it will treat them as normal states and have a transition $(q_{accept}, \$) \to (q'_{accept}, \$, R)$ and similar for q_{reject} where q'_{accept}, q'_{reject} are accepting and rejecting states of M'. But what should we do if M halts not at the start of the tape moving left? Then we need two transitions moving, say, right and then back and entering the halting state on the last step. That is, accepting sequence will be $(q_{accept}, c) \to (q_a, c, R)$, then $(q_a, d) \to (q'_{accept}, d, L)$. Here, $c, d \in \Gamma - \{\$\}$ are any non-\$ symbols and q_a remembers that the state was accepting; we'll use a different state say q_r for reject.

So, $M' = (Q', \Sigma, \Gamma \cup \{\$\}, q'_0, q'_{accept}, q'_{reject})$, where $Q' = Q \cup \{q'_0, q_a, q_r, q'_{accept}, q'_{reject}\}$. In δ', there are all transitions in δ, plus there is an additional transition on \$ from every state, transitions for setting the marker and handling halting states as described above.

□

Lemma 14. *For every TM M with a two-way infinite tape there exists a TM M' with one-way infinite tape accepting the same language.*

Proof sketch. The first thing that M' needs to do is to mark the start of its tape. Now, there are two ways it can imitate the two-way infinite tape. First, it can move the whole string to the right every time M moves left to the marker. There we need a new state for every pair i, c where i is remembering that the state trying to move left was q_i and c is a symbol being moved. Another set of states, say $q_{i,\sqcup}$, will scan the string backwards ending with $(q_{i,\sqcup}, \$) \to (q_i, \$, L)$. In that case, those states are used also to move the string to the right after placing the marker.

A different way to simulate two-way infinite tape is to have a "double" symbol in each cell, e.g. (a, b), where a is the symbol in the i^{th} cell to the right of the start on the two-way infinite tape and b is the i^{th} symbol to the left of the start. Then, double each state to the "left-state" and "right-state", change the transition table to use the double symbols and reverse the direction of transition when working with the second symbols in "left-state" copies of the states. Of course, at the marker M' switches from looking at the first symbol to the second if moving left, and from second to the first moving right.

□