CS 3719 (Theory of Computation and Algorithms) – Lecture 15

Antonina Kolokolova[*]

February 10 2011

1 Turing machines

Now we are moving on to the model of computation which we will use for the rest of the class: the Turing machine.

Alan Turing was working on a problem posed by Hilbert: does there exist an algorithm that for any statement in the language of mathematics would state if this statement is provable? The original problem asked for an algorithm that for any statement of mathematics would state whether it is true or false; Gödel has shown (his famous Incompleteness Theorem) that there are statements of mathematics for which such answer cannot be given. There were several mathematicians working on this problem at that time; notably, Alonco Church solved this problem (to give a negative answer) at about the same time, by inventing lambda-calculus. Turing's approach is somewhat more computational: he defined a model of computation which we now call the Turing machine, equivalent to Church's model in terms of power, and used it to show undecidability results, thus giving a negative answer to Hilbert's problem.

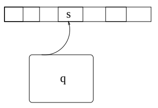
Definition 10 (Church-Turing thesis)**.** *Anything computable by an algorithm of any kind (our intuitive notion of algorithm) is computable by a Turing machine.*

Since this statement talks about an intuitive notion of algorithm we cannot really prove it; all we can do is that whenever we think of a natural notion of an algorithm, show that this can be done by a Turing machine.

[*]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.

2 Definition of a Turing machine

The weakness of finite automata was the lack of memory; pushdown automata had some memory, but only a limited access to it. A natural next step would be to take a finite automaton and add an unrestricted-access unlimited memory to it. This is essentially the intuition behind the definition of a Turing machine.



Memory is given to a Turing machine in form of an infinite tape. The Turing machine has a "head" which points to a cell on the tape. The head can both read and write in that cell, and can move in either direction. In the simplest definition, every cell contains just one symbol, and in one step of the computation the head can move by one position to the left or to the right.

Definition 11. *Formally, a Turing machine is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$. Here, Q is a finite set of states as before, 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.*

At the start of the computation the input is written on the tape and the head points to the first symbol of the input. Sometimes Turing machines are defined to have a tape which is only infinite in one direction; in that case, the first symbol of the input is in the first cell of the tape. We will prove soon that these two definitions are equivalent.

The simplest Turing machine halts on an input by entering the state q_{accept} or q_{reject} in some point in the computation. Usually we assume that there are no transitions from q_{accept} or q_{reject}. It *accepts* its input if it halts in q_{accept}.

Example 1. Recall that we have shown that the language $\{ww | w \in \{a, b\}\}$ is not context-free. Here we will show how to design a Turing machine accepting this language.

For simplicity, let's first design a Turing machine accepting the language $\{w\#w | w \in \{a, b^*\}$. Then we will say how to modify it to accept our original language.

We will be constructing a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}.q_{reject})$. The symbols in the input alphabet are $\Sigma = \{a.b.\#\}$.

M works as follows. It starts in a state q_0 pointing to the first symbol of the first copy of w. A symbol in the cell it is pointing to can be one of $\{a, b, \#, \sqcup\}$. If it is a or b, then M has to remember it (by going to q_1 or q_2, respectively) and scan till it is past the $\#$ sign. Then,

	a	b	$\#$	$\sqcup$	o
q_0	(q_1, o, R)	(q_2, o, R)	$(q_3, \#, R)$	q_{reject}	(q_0, o, R)
q_1	(q_1, a, R)	(q_1, b, R)	$(q_4, \#, R)$	q_{reject}	(q_1, o, R)
q_2	(q_2, a, R)	(q_2, b, R)	$(q_5, \#, R)$	q_{reject}	(q_2, o, R)
q_3	q_{reject}	q_{reject}	q_{reject}	q_{accept}	(q_3, o, R)
q_4	(q_6, o, L)	q_{reject}	q_{reject}	q_{reject}	(q_4, o, R)
q_5	q_{reject}	(q_6, o, L)	q_{reject}	q_{reject}	(q_5, o, R)
q_6	(q_6, o, L)	(q_6, o, L)	(q_6, o, L)	$(q_0, \sqcup, R)$	(q_6, o, L)

Figure 1: A transition table for TM M recognizing $\{w\#w | w \in \{a.b\}^*\}$.

it has to see if the corresponding first symbol of the second copy of w is a or b. After that, M returns to the second symbol of the first copy and repeats till it runs out of $a's$ and $b's$. If at that point it ran out of symbols on both sides of #, then accept, otherwise reject.

How do we make sure that the same symbol is not counted twice? Let's introduce a new symbol into Γ, call it o (the name does not matter, as long as it does not occur in Σ and is not $\sqcup$). We will use it to mark the cells that we have already visited on both sides of #, so we don't count them again.

Now, each iteration of the computation proceeds as follows. Start in state q_0; then move right changing to q_1, q_2 or q_3 depending on whether the cell contained a, b or #. In the latter case, we have matched all symbols in the first copy of w and need to check if there is anything left in the second copy; so scan right staying in q_3, and reject if seeing a or b; accept if got to a $\sqcup$ (end of input). In the first two cases, mark the cell with o and move right until # is found. Then, change state: from q_1 to q_4 and from q_2 to q_5: here, q_4 will be looking for an a and q_5 for a b. In these two states, skip over o's; if a wrong symbol is seen first, reject, otherwise change to the "going-back state" q_6.

If M has a doubly-infinite tape, then in q_6 M can move left until it hits a $\sqcup$ before the input, at which point it changes to q_0 and moves right. If the tape has a beginning, we'll need to use an extra state: scan till # in q_6 and then till the rightmost o of the first copy of w in q_7. The table lists all the transitions for the doubly-infinite tape case.

This Turing Machine accepts a language $\{w\#w | w \in \{a, b\}^*\}$. How can we modify it to accept the language $\{ww | w \in \{a, b\}^*\}$? The simplest way to do it is to find the middle (by going back-and-forth from the beginning to end marking cells; in this case we'd want different markers for a and b). Then, insert # in the middle and move the second copy one cell left. We are back to the $\{w\#w\}$ case, except need a different name for q_0 and our a and b became some kind of marked $\dot{a}$ and $\dot{b}$, so need to rename the columns (and add rejects for seeing old a and b)