Handout: Turing Machines

Laura Kovács May 21, 2013

---

1. Syntax. In the previous lecture on context-free grammars (CFG), you have seen that context-free languages are exactly those languages that are accepted by push-down automatons (PDA) – see Slides-13 on the web. In other words, PDA are equivalent in power to CFG. That is, if a language is accepted by a CFG then it is also accepted by a PDA, and vice versa.

However, you have also seen that there are fairly simple languages that are not context-free – see Slides-12 on the web. For example, the languages $L_1 = \{\mathtt{0}^n\mathtt{1}^n\mathtt{2}^n \mid n \geq 0\}$ and $L_2 = \{w\#w \mid w \in \{\mathtt{0},\mathtt{1}\}^*\}$ are not context-free. The main limitation of a PDA is that its unbounded memory —the stack— can be accessed only in a restricted manner (last-in-first-out). In this lecture, we therefore replace the stack with an unbounded number of memory cells, each of which can be accessed (not only the top one). There are many equivalent definitions of the *resulting machines*, such as RAM (Random-Access Memory) machines, which are very close to how real computers work. For simplicity we stick to the simpler, classical definition of *Turing machines*, whose unbounded, arbitrary-access memory is arranged in form of an infinite "tape," i.e., an infinite sequence of memory cells that can be read and written.

A *Turing machine* $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ consists of

Q ... a finite set of states,
Σ ... a finite set of input symbols,
Γ ... a finite set of tape symbols,
$\delta\colon Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ ... a transition relation,
$q_0 \in Q$ ... an initial state,
$q_a \in Q$ ... an accept state,
$q_r \in Q$ ... a reject state.

We insist that one of the tape symbols is the "blank" symbol, denoted $\sqcup$, which is not an input symbol; that is, $\sqcup \in \Gamma \backslash \Sigma$. We also insist that $q_a \neq q_r$, and that $|\delta(q, \gamma)| \geq 1$ for all $q \in Q$ and $\gamma \in \Gamma$. Note that the transition relation is nondeterministic; the TM M is *deterministic* if for all $q \in Q$ and $\gamma \in \Gamma$, we have $|\delta(q, \gamma)| = 1$.

2. Semantics. A run is a sequence of machine configurations. For finite automata, a configuration is a state $q \in Q$; for push-down automata, a configuration is a pair (q, s) consisting of a state $q \in Q$ and the stack contents $s \in \Gamma^*$; for Turing machines, a configuration is a quadruple (u, q, a, v) consisting of

the tape contents $u \in \Gamma^*$ to the left of the read/write head,
a state $q \in Q$,
the tape symbol $a \in \Gamma$ under the read/write head, and
the tape contents $v \in \Gamma^*$ to the right of the read/write head.

The tape is finite to the left and infinite to the right, but it contains only finitely many non-blank symbols: the tape contents to the right of the read/write head really is v followed by infinitely many $\sqcup$ symbols.

Unlike for finite automata and push-down automata (or CFG), runs of TMs are infinite. A *run* r of M is an infinite sequence

$$(u_0, p_0, a_0, v_0) \to (u_1, p_1, a_1, v_1) \to (u_2, p_2, a_2, v_2) \to \cdots$$

of configurations with $p_0, p_1, \ldots \in Q$ and $a_0, a_1, \ldots \in \Gamma$ and $u_0, v_0, u_1, v_1, \ldots \in \Gamma^*$ such that

(1) $u_0 = \varepsilon$ and $p_0 = q_0$ [initially the r/w head is at the left end of the tape],
(2) for all $i \geq 0$, either $\delta(p_i, a_i) \ni (p_{i+1}, b, R)$ [move right] and $u_{i+1} = u_i b$ and

(a) $v_i = a_{i+1}v_{i+1}$ [non-blank symbol to the right of the r/w head] or
(b) $v_i = v_{i+1} = \varepsilon$ and $a_{i+1} = \sqcup$ [only blank symbols to the right of the r/w head];

or $\delta(p_i, a_i) \ni (p_{i+1}, b, L)$ [move left] and

(a) $u_{i+1}a_{i+1} = u_i$ and $v_{i+1} = bv_i$ [r/w head not at the left end of the tape] or
(b) $u_i = u_{i+1} = \varepsilon$ and $a_{i+1} = b$ and $v_{i+1} = v_i$ [r/w head at the left end of the tape].

The run is *accepting* if $p_n = q_a$ for some $n \geq 0$, and $p_i \notin \{q_a, q_r\}$ for all $0 \leq i < n$; the run is *rejecting* if $p_n = q_r$ for some $n \geq 0$, and $p_i \notin \{q_a, q_r\}$ for all $0 \leq i < n$; the run is *looping* if $p_i \notin \{q_a, q_r\}$ for all $i \geq 0$. Every run is either accepting, rejecting, or looping. Given an input word w, the run r is a run *over* w if

(3) $w = u_0a_0v_0$ [the initial tape contents is w], or $w = u_0 = v_0 = \varepsilon$ and $a_0 = \sqcup$ [in case $w = \varepsilon$].

If M is deterministic, then for every input word w, there is exactly one run of M over w; this run may accept, reject, or loop. If M is nondeterministic, then there may be many (even infinitely many) runs of M over w; some of them may accept, some reject, some loop. The *language* of M is

$L(M) = \{w \in \Sigma^* \mid \text{there is an accepting run of } M \text{ over } w\}$.

A language L is *recursively enumerable* (*r.e.*) if there is a deterministic TM M such that $L(M) = L$.

3. Examples. The languages L_1 and L_2 given in paragraph 1 are r.e. A high-level description of a TM M_2 that accepts L_2 can be given as follows:

1. if the current tape symbol is `0` or `1`, then go to Step 2, else go to Step 8;
2. remember (in the state) if the current tape symbol is `0` or `1`, and overwrite it with `X`;
3. move right across `0` and `1` symbols to the next # symbol;
4. move right across `X` symbols to the next non-`X` symbol; if it is the same as the remembered symbol, then overwrite it with `X`;
5. move left across `X` symbols to the # symbol;
6. move left across `0` and `1` symbols to the next `X` symbol;
7. move one tape cell to the right and go to Step 1;
8. check that the current tape symbol is # and move right across `X` symbols to the next non-`X` symbol;
9. check that the current tape symbol is $\sqcup$ and accept.

If in any step, these instructions cannot be followed, then reject (e.g., if in Step 3, a `X` or $\sqcup$ is encountered before a #). A state-transition diagram of M_2 is shown in Figure 1. Note that in Figure 1 the (short-hand notation) label $\mathtt{0}, \mathtt{1} \to \mathsf{R}$ is used on the transition going from q_3 to itself. This label means that M_2 stays in q_3, moves to the right when it reads a `0` or a `1` in state q_3, and does not change the symbol on the tape. To simplify the figure, we do not show the reject state q_r or the transitions going to the reject state q_r. Those transitions occur implicitly whenever a state lacks an outgoing transition for a particular symbol.

4. Turing deciders. A run of a TM which is either accepting or rejecting is called *halting*. A TM M is a *Turing decider* if for all input words $w \in \Sigma^*$, all runs of M over w are halting. A language L is *recursive* if there is a deterministic Turing decider M such that $L(M) = L$. For example, the languages L_1 and L_2 from paragraph 1 are recursive.

5. Robustness. Many variations of the Turing machine model are equally powerful. This can be shown by TM simulations. For example, a DTM M with two tapes can be simulated by a standard DTM M' (with one tape):

A configuration $(q, u_1, a_1, v_1, u_2, a_2, v_2)$ of the two-tape machine *corresponds* to the configuration $(q, \varepsilon, \#, u_1\$a_1v_1\#u_2\$a_2v_2)$ of the one-tape machine (the \$ symbols mark the positions of the two r/w heads). The two machines start in corresponding configurations. For every transition t of the two-tape machine M, the one-tape machine M' performs a finite sequence $t_1, \ldots, t_n$ of transitions such that the two machines are again in corresponding configurations.

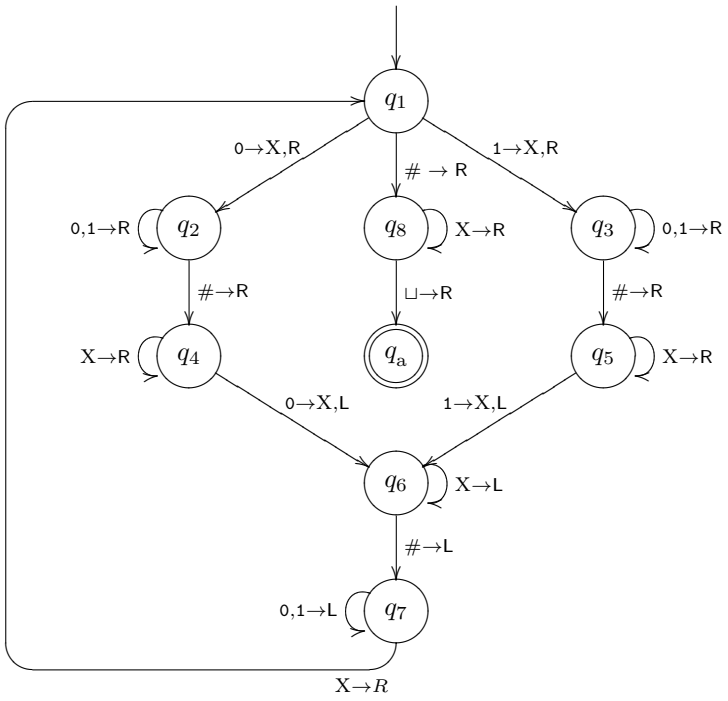

Figure 1: State diagram for $M_2 = \big(\{q_1, \ldots, q_8\}, \{0, 1, \#\}, \{0, 1, \#, \mathrm{X}, \sqcup\}, q_1, q_a, q_r\big)$.

Note that this simulation preserves acceptance, rejection, and looping. Since the simulation preserves acceptance, it follows that the two-tape TMs accept no more than the r.e. languages. Since the simulation preserves also looping, it follows that the two-tape Turing deciders accept no more than the recursive languages.

6. Nondeterminism. A nondeterministic TM M can be simulated by a three-tape DTM D, as follows. We view M's computation on an input w as a tree. Each branch of the tree represents one branch of nondeterminism (i.e. one possible run of M on w). Each node of the tree is a configuration of M; the root of the tree is the starting configuration of M. We then construct D to explore in a breadth first order all possible branches of M's nondeterministic computation tree. D uses its three tapes in a particular way. Tape 1 of D always contains the input word and is never altered. Tape 2 keeps track of D's location in M's nondeterministic computation tree (i.e. tape 2 remembers the tree node that we currently explore). Tape 3 maintains a copy of M's tape on some branch of its nondeterministic computation (i.e. tape 3 simulates M upto the tree node stored in tape 2). If D encounters an accepting configuration using tapes 2 and 3, then D accepts the input w stored on tape 1. The construction of D is such that for every input word w,

1. if some run of M over w accepts, then the run of D over w accepts;
2. if all runs of M over w reject, then the run of D over w rejects;
3. if no run of M over w accepts, and some run of M over w loops, then the run of D over w loops.

Further, using a simulation like the one of paragraph 5, the DTM D with three tapes (and hence M) can be simulated by a standard DTM M' with only one tape.

Therefore, the nondeterministic TMs accept the r.e. languages, and the nondeterministic Turing deciders accept the recursive languages.

7. Positive boolean operations. Given two DTMs M_1 and M_2, we can construct a single DTM M that encodes the two tapes of M_1 and M_2 similar to the construction of paragraph 5, and alternates between simulating a transition of M_1 and simulating a transition of M_2.

Union As soon as one of M_1 or M_2 accepts, let M accept. Once both M_1 and M_2 have rejected, let M reject. Consequently, if neither M_1 nor M_2 accepts, and M_1 or M_2 loops, then M will loop.

Intersection As soon as one of M_1 or M_2 rejects, let M reject. Once both M_1 and M_2 have accepted, let M accept. Consequently, if M_1 or M_2 loops, and neither M_1 nor M_2 rejects, then M will loop.

It follows that the recursive languages are closed under union and intersection, and that the r.e. languages are closed under union and intersection.

8 Complementarity. Given a DTM M, let $\overline{M}$ be the DTM that results from M by swapping q_a and q_r. If M is a Turing decider, then $L(\overline{M}) = \Sigma^* \backslash L(M)$. It follows that the recursive languages are closed under complementation. However, if M loops on some inputs, then $L(\overline{M}) \subset \Sigma^* \backslash L(M)$. So, the r.e. languages may not be closed under complementation. A language L is *co-r.e.* if there is a DTM M such that $L(M) = \Sigma^* \backslash L$; that is, M accepts the complement of M. For r.e. languages L, there is a deterministic TM that is guaranteed to accept all inputs that are in L, but it may not halt on inputs that are not in L. For co-r.e. languages L, there is a deterministic TM that is guaranteed to reject all inputs that are not in L, but it may not halt on inputs that are in L.

Theorem. A language L is recursive iff L is both r.e. and co-r.e.

We will soon see a language that is r.e. but not recursive, namely, the membership problem for DTMs. It follows that r.e. and co-r.e. are different language classes.