Handout 9: Turing Machines

9.1 Syntax. We have seen that there are fairly simple languages that are not context-free, such as $L_1 = \{0^n1^n2^n \mid n \geq 0\}$ and $L_2 = \{w\#w \mid w \in \{0,1\}^*\}$. The main limitation of PDAs is that its unbounded memory —the stack— can be accessed only in a restricted manner (last-in-first-out). 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,
δ: $Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ ... 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 $|\delta(q, \gamma)| = 1$.

9.2 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 FAs and PDAs, 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) $w = u_0 a_0 v_0$ [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 $L(M) = L$.

9.3 Examples. The languages L_1 and L_2 from Section 9.1 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.

9.4 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 Section 9.1 are recursive.

9.5 Decision problems, instances, encodings. A *decision problem* (I, f) has the form "Given $x \in I$, is $f(x) =$ YES?", for some set I of possible inputs, and some function f that maps all inputs $x \in I$ to either YES or NO. Here are three examples of decision problems:

GRAPHREACHABILITY Given a directed graph (V, E) and two nodes $u, w \in V$, is there a path from u to w?

HILBERTSTENTH Given a multivariate polynomial equation e, does e have an integer solution?

INTEGERARITHMETIC Given a closed first-order formula f over the integers with comparison, addition, and multiplication, is f true?

An *instance* of the decision problem "Given $x \in I$, is $f(x) =$ YES?" is a particular input $x \in I$. Here are three sample instances, one for each of the above problems:

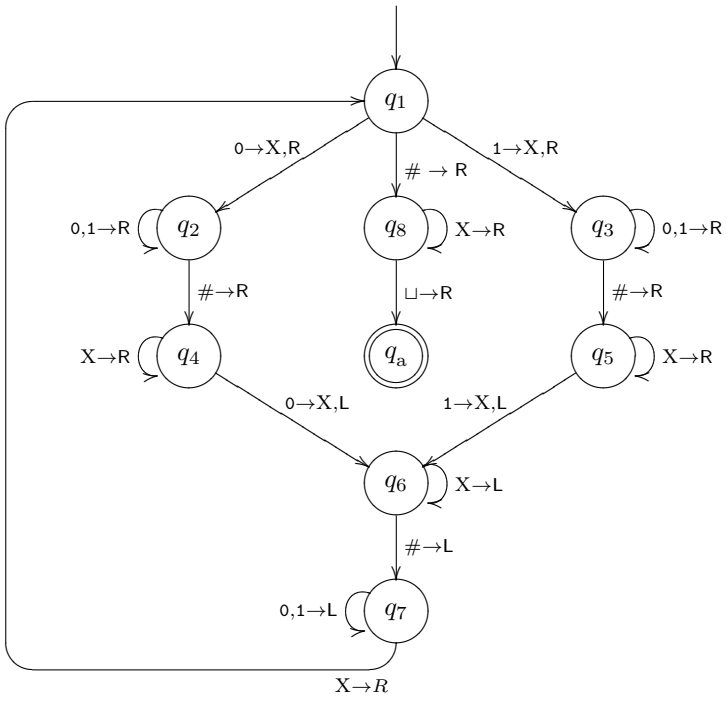

Figure 1: State diagram for $M_2 = \big(\{q_1, \ldots, q_8\}, \{0, 1, \#\}, \{0, 1, \#, \text{X}, \sqcup\}, q_1, q_a, q_r\big)$.

GraphReachability

$V = \{v_0, v_1, v_2, v_3\}$
$E = \{(v_0, v_1), (v_1, v_0), (v_1, v_1), (v_1, v_2), (v_3, v_2)\}$
$u = v_1$
$w = v_3$

This is a No-instance, because this particular graph has no path from v_1 to v_3.

HilbertsTenth

e: $\quad 3x^2y + 27x^{13}z^2 - xyz + 5z = 0$

This is a Yes-instance (why?).

IntegerArithmetic

f: $\quad (\forall x)(\exists y)(y > x \ \wedge \ (\forall z_1)(\forall z_2)(z_1 > 1 \ \wedge \ z_2 > 1 \ \Rightarrow \ z_1 z_2 \neq y))$

This is a Yes-instance (why?).

In order to write instances down, or input them into a machine, we must work with *encodings* of instances. An encoding $\langle x \rangle \in \Sigma^*$ of an instance $x \in I$ is a word over some alphabet Σ. Here are sample encodings for two of the above instances, taking Σ to be the set of ASCII symbols:

GraphReachability

$\langle V, E, u, v \rangle$: `{v0,v1,v2,v3},{(v0,v1),(v1,v0),(v1,v1),(v1,v2),(v3,v2)},v1,v3`

HILBERTSTENTH

$\langle e \rangle$: `3*x**2*y+27x**13*z**2-x*y*z+5*z=0`

There are many possible encoding schemes $\langle \cdot \rangle$ for the instances of a problem. For our present purposes, encoding schemes are reasonable as long as they are intertranslatable by deterministic Turing deciders.

9.6 Decidability. Let QUESTION be the decision problem "Given $x \in I$, is $f(x) =$ YES?", and let $\langle \cdot \rangle$: $I \to \Sigma^*$ be a reasonable encoding scheme for QUESTION. The set

$$L(\text{QUESTION}) = \{\langle x \rangle \mid x \in I \ \wedge \ f(x) = \text{YES}\}$$

of YES-instances is a language (a subset of Σ^*). The decision problem QUESTION is *recursive*, or *decidable*, if the language L(QUESTION) is recursive. Decidable problems can be solved by deterministic Turing machines: for a recursive problem QUESTION, there is a deterministic Turing decider M such that if you wish to know whether $f(x) =$ YES for some $x \in I$, you can run M with the initial tape contents $\langle x \rangle$ and wait until it halts. If M halts in q_a, then the answer is YES; if it halts in q_r, then the answer is NO. Since M is a Turing decider, it is guaranteed to halt.

9.7 Church-Turing thesis. Since a DTM is a very simple computational apparatus (you surely can build a simulator of DTMs in your favorite programming language), it follows that every recursive problem can be solved by computational means. The Church-Turing thesis states the converse, namely, that all problems which can be solved by computational means are recursive. This is a claim that cannot be proved, because any proof would require an unassailable, general definition of what it means to solve a problem "by computational means." But the claim has been proved to be true for all specific definitions of "computational means" that people have suggested. For example, all problems that can be solved by Java programs, even when run on idealized machines with unbounded memory, are recursive. So in a very fundamental sense, Java programs are no more "powerful" than DTMs: both Java programs and DTMs can solve the same problems (namely, the recursive ones). And the same can be said for every programming language. Soon we will see that even nondeterminism does not allow us to solve more than the recursive problems.

9.8 Semi-decidability. The decision problem QUESTION is *r.e.*, or *semi-decidable*, if the language L(QUESTION) is r.e. For an r.e. problem QUESTION, there is a DTM M such that if you wish to know whether $f(x) =$ YES for some $x \in I$, you can run M with the initial tape contents $\langle x \rangle$ and wait until it halts. Since M accepts the language L(QUESTION), it will always halt (in q_a) if the answer is YES; but since M is not necessarily a Turing decider, it may not halt (it may loop) if the answer is NO!

9.9 Examples. The problem GRAPHREACHABILITY is decidable (why?). The problem HILBERTSTENTH is semi-decidable (why?). It can be shown that HILBERTSTENTH is not decidable, and that INTEGERARITHMETIC is not even semi-decidable. The proofs of these two results were major mathematical breakthroughs in the 20th century.