After this tutorial you should be able to:

1. Prove a language is not regular.
2. Design/program Turing Machines to do certain tasks (think of this as if you had a low-level programming language that has some finite internal memory and just a single data structure, a tape. You might find this simulator helpful: `http://morphett.info/turing/turing.html`)

---

1 Showing a language is not regular

Problem 1. Fix $\Sigma = \{a, b\}$. Prove that the following languages are not regular.

1. $\{a^i b^j : i > j\}$.
2. $\{a^n b^m : n$ divides m, or m divides $n\}$.
3. $\{a^{n^2} : n \geq 0\}$.
4. All strings $a^i b^j$ such that (a) i is even, or (b) $j < i$ and j is even. (hard)

Solution 1.

1. Let $x_i = a^i$. If $i < j$ then x_i, x_j can be distinguished by $z = b^i$. Indeed, $x_i z = a^i b^i \notin L$ while $x_j z = a^j b^i \in L$.
2. Recall that an integer $n > 1$ is prime if its only divisors are 1 and n. Let $p(i)$ be the ith prime number, e.g., $p(1) = 2, p(2) = 3, p(3) = 5, p(4) = 7, p(5) = 11, \ldots$.

   Let $x_i = a^{p(i)}$. Then if $i \neq j$ then x_i, x_j are distinguishable by $z = b^{p(i)}$. Indeed, $x_i z = a^{p(i)} b^{p(i)} \in L$ (since every number divides itself), while $x_j z = a^{p(j)} b^{p(i)} \notin L$ (since no prime divides any other prime).
3. Let $x_i = a^{i^2}$. If $i < j$ then x_i, x_j are distinguishable by $z = a^{2i+1}$. Indeed, $|x_i z| = i^2 + 2i + 1 = (i+1)^2$ and so $x_i z \in L$, while $j^2 < |x_j z| = j^2 + 2i + 1 < j^2 + 2j + 1 = (j+1)^2$ and so $x_j z \notin L$.
4. Let $x_i = a^{2i+1}$. If $j < i$ then x_i, x_j are distinguishable by $z = b^{2i}$. Indeed, $x_i z = a^{2i+1} b^{2i} \in L$ while $x_j z = a^{2j+1} b^{2i} \notin L$ since $j < i$ implies $2j + 1 < 2i + 1$ and in particular that $2j + 1 < 2i$.

2 Turing Machines

Problem 2. Prove that every regular language is decidable.

Solution 2. If L is regular, there is a DFA for it, say $(Q, \Sigma, q_0, \delta, F)$. Build a TM M with state set $Q \cup \{q_{\text{accept}}\}$, input alphabet Σ, tape alphabet $\Sigma \cup \{\_\}$, initial state q_0, and the following transitions: for $\delta(q, a) = q'$ add the transition $q\,a\,a\,R\,q'$ to M, and for every $q \in F$ add transitions $q\,\_\,\_\,*\,q_{\text{accept}}$.

To see that $L = L(M)$, note that M simulates the DFA for L, and when the DFA finishes reading the input, the TM's head is on the first blank after the input, and if the DFA were in a final state, the TM accepts.

Problem 3. Let $\Sigma = \{0, 1, \#\}$. For each of the following languages, build a decider for it.

1. The set of strings $u\#v$ for $u, v \in \{0,1\}^*$ such that $|u| < |v|$.
   - For instance 111#0000 should be accepted and 101#011 should not.
2. The set of strings $u\#v$ for $u, v \in \{0,1\}^*$ such that u is length-lexicographically smaller than v. This means that either (i) $|u| < |v|$, or (ii) $|u| = |v|$, $u \neq v$, and if i is the left-most position where u differs from v then $u_i = 0$ and $v_i = 1$.
   - For instance, 111#0000 and 0010#0101 should be accepted, but 100#1 and 010#001 should not.

Solution 3.

1. Our strategy is to erase one character from each of u and v. Repeating this procedure, if u and v are the same length, then they will be fully erased at the same time. On the other hand, if they are different lengths, then one of them will disappear before the other. Since the order and composition of the strings does not matter, it is convenient to erase from both far ends of the string.

We use the following syntax:
current-state current-symbol new-symbol direction new-state.

```
; * is a wildcard symbol
; _ is a blank

; start state
0 * * * at-leftmost

at-leftmost # * r check-v-nonempty
at-leftmost * _ r goto-rightmost

goto-rightmost _ * l at-rightmost
goto-rightmost * * r goto-rightmost
```

```
at-rightmost # * * halt-reject
at-rightmost * _ l goto-leftmost

goto-leftmost _ * r at-leftmost
goto-leftmost * * l goto-leftmost

check-v-nonempty _ * * halt-reject
check-v-nonempty * * * halt-accept
```

2. The overall strategy is similar, but this time we care about the composition of the strings so a more sophisticated approach is required. We care about the first point at which the strings differ. To this end, it makes sense to erase both strings starting from the right, and detect when the first change occurs. Then, once we have erased both strings, we use the information we have (which string is longer, and which string had a 0 vs. a 1 at the point where the first difference was) to decide whether to accept the string.

```
; * is a wildcard symbol
; _ is a blank

; start state
0 * * * at-leftmost

at-leftmost # * r check-v-nonempty ;u was exhausted by last pairing
at-leftmost 0 X r goto-rightmost
at-leftmost 1 Y r goto-rightmost

goto-rightmost _ * l at-rightmost
goto-rightmost X * l at-rightmost
goto-rightmost Y * l at-rightmost
goto-rightmost * * r goto-rightmost

at-rightmost # * * halt-reject  ; |u|>|v|
at-rightmost 0 X l goto-leftmost
at-rightmost 1 Y l goto-leftmost

goto-leftmost _ * r at-leftmost
goto-leftmost X * r at-leftmost
goto-leftmost Y * r at-leftmost
goto-leftmost * * l goto-leftmost

check-v-nonempty 0 * * halt-accept ; |u|<|v|
check-v-nonempty 1 * * halt-accept ; |u|<|v|
check-v-nonempty _ * * halt-reject ; u = v = epsilon
```

```
check-v-nonempty * * * restore-right

restore-right * * r restore-right
restore-right _ * l restore-left

restore-left X 0 l restore-left
restore-left Y 1 l restore-left
restore-left _ * r eq-leftmost
restore-left * * l restore-left

eq-leftmost 0 _ r eq-leftmost-0
eq-leftmost 1 _ r eq-leftmost-1
eq-leftmost # * * halt-reject  ; u=v

eq-leftmost-0 * * r eq-leftmost-0
eq-leftmost-0 # * r eq-leftmost-0-v

eq-leftmost-1 * * r eq-leftmost-1
eq-leftmost-1 # * r eq-leftmost-1-v

;unreachable? eq-leftmost-0-v _ * * halt-reject
eq-leftmost-0-v 1 * * halt-accept ; first difference was ui=1 vi=0
eq-leftmost-0-v 0 Z l eq-gotoleft ; no difference
eq-leftmost-0-v * * r eq-leftmost-0-v

;unreachable? eq-leftmost-1-v _ * * halt-reject
eq-leftmost-1-v 0 Z * halt-reject ; first difference was ui=0 vi=1
eq-leftmost-1-v 1 Z * eq-gotoleft ; no difference
eq-leftmost-1-v * * r eq-leftmost-1-v

eq-gotoleft * * l eq-gotoleft
eq-gotoleft _ * r eq-leftmost
```

Problem 4. Let $\Sigma = \{1, \#\}$. For each of the following languages, build a decider for it.

1. The set of strings of the form $u\#v$ for $u, v \in \{1\}^*$ where $|u|$ divides $|v|$.
   - For instance, 11#1111 should be accepted, but 11#111 should not.
2. The set of strings of the form $u\#v\#w$ for $u, v, w \in \{1\}^*$ such that $|u|$ multiplied by $|v|$ equals $|w|$.
   - For instance 11#11#1111 should be accepted, but 11#111#11 should not.

Problem 5. Devise a TM that recognises the language $\{w\#w \mid w \in \{0,1\}^*\}$.

You might find it easier to write a semi-formal description first, then write the transition table in full.

Solution 5. Here is the formal description.

q_{start}	if read 0, write X, move right to state q_{0a} if read 1, write X, move right to state q_{1a} if read #, write #, move right to state q_{end} if read X, move right else reject
q_{0a}	move right through any 0's and 1's if read #, move right and change to state q_{0b} if read $\sqcup$, reject
q_{0b}	move right through any X's if read 0, write X, move left to state q_{return} if read 1 or $\sqcup$, reject
q_{1a}	move right through any 0's and 1's if read #, move right and change to state q_{1b} if read $\sqcup$, reject
q_{1b}	move right through any X's if read 1, write X, move left to state q_{return} if read 0 or $\sqcup$, reject

q_{end}	if read $\sqcup$, write $\sqcup$, move right to state q_{accept}
	if read X, move right
	else reject

q_{return}	move left until read a $\sqcup$, then move right to state q_{start}

See `http://morphett.info/turing/turing.html?e89552037b48de4bd3f966fee26eba60` for a commented solution in the Morphett notation.

Problem 6. In this problem you will see that TMs can also compute functions (and not just decide things).

Devise a TM M that, given an input string of the form 0^n10^m reaches an accepting configuration with 0^{n+m} written on the tape.

1. Give an implementation description of M.
2. Give the formal description of M, i.e., states, transitions, etc.
3. Give the computation, i.e., sequence of configuration, of M on input 00001000.

Solution 6.

1. (a) Move to the right until we reach a 1 (skipping 0s)
   (b) Rewrite the 1 with a 0
   (c) Move to the right until we reach a blank (skipping 0s)

(d) Move to the left once, to rewrite the last 0 with a blank

2. - $Q = \{q_0, q_1, q_2, q_{accept}, q_{reject}\}$
   - $\Sigma = \{0, 1\}$
   - $\Gamma = \{0, 1, \sqcup\}$
   - The transitions are given by

	0	1	$\sqcup$
q_0	$(0, R, q_0)$	$(0, R, q_1)$	$(\sqcup, R, q_{accept})$
q_1	$(0, R, q_1)$		$(\sqcup, L, q_2)$
q_2	$(\sqcup, R, q_{accept})$		

Blank entries are of the form $(0, R, q_{reject})$.

This machine takes a string of the form 0^n10^m as input and when it finnishes computing it has written 0^{n+m} on the tape. In other words, TM can also be used to *compute functions* (not just to decide languages). Note the similarity with computers.

3.

$$q_0 00001000$$
$$0 q_0 0001000$$
$$00 q_0 001000$$
$$000 q_0 01000$$
$$0000 q_0 1000$$
$$00000 q_1 000$$
$$000000 q_1 00$$
$$0000000 q_1 0$$
$$00000000 q_1$$
$$0000000 q_2 0$$
$$0000000 \sqcup q_{accept}$$