

Proving decidability and recognizability

- Three levels of detail for describing TMs:
 - **Formal level:** Give $(Q, \Sigma, \Gamma, q_0, q_{accept}, q_{reject}, \delta)$, i.e. specify states, alphabets and transition function fully. Example: TM from first lecture that decides $\{w \in \{0, 1\}^* \mid w \text{ has the same number of 0's as 1's}\}$
 - **Implementation level:** Describe, in english, the actions of the TM, making reference to the movement of the tape head, the symbols on the tape, and possibly some of the more important states. Example: most of the TMs that we've seen so far, including the universal TM U .
 - **High level:** More like pseudocode, an english description of the algorithm that doesn't make reference to the internals of the TM or the representation of objects as strings. Example: we will see a description like this later today (the one in the proof of the second theorem).

Some useful theorems

- **Theorem:** If a language L is decidable, then \bar{L} is decidable.
- Proof: Let M be a TM that decides L . Define M' to be identical to M , except that it has the accept/reject states interchanged (i.e. the accepting state of M' is q_{reject} , and the rejecting state of M' is q_{accept}). To show that M decides \bar{L} , we need to show 1) if $x \in \bar{L}$ then M' accepts x ; and 2) if $x \notin \bar{L}$ then M' rejects x .
 - If $x \in \bar{L}$ then $x \notin L$, so M rejects x . Since M' is the same as M , but with interchanged accept/reject states, M' accepts x .
 - If $x \notin \bar{L}$, then $x \in L$, so M accepts x , and thus M' rejects x .
- Question: why does this not work for a recognizable language?
- **Theorem:** If L and \bar{L} are both recognizable, then L is decidable (and by the first theorem, \bar{L} is also decidable).
- Proof: Let M_1 be a TM that recognizes L , and let M_2 be a TM that recognizes \bar{L} . Before describing a TM that decides L , let's first look at a bad (but tempting) attempt at describing such a TM.

Bad Idea: $M =$ "on input w ...

1. Run M_1 on input w
2. If M_1 accepts, then accept
3. Run M_2 on input w
4. If M_2 accepts, then reject

The idea is tempting because either $w \in L$, in which case M_1 accepts it, or $w \in \bar{L}$, in which case M_2 accepts it; and as soon as one machine accepts we know what to do. The problem is that if $w \in \bar{L}$, then M_1 might never halt on input w (since M_1 just recognizes L , and doesn't necessarily decide it), in which case we never get to line 2. The solution is to run the two machines in "parallel":

Good Idea: $M =$ "on input $w...$

1. for $t \leftarrow 1, 2, 3, \dots, \infty$ do
2. run M_1 on input w for t steps
3. if M_1 accepts then accept
4. run M_2 on input w for t steps
5. if M_2 accepts then reject

- Claim: M decides L .
- Proof: we need to show 1) if $w \in L$ then M accepts w ; and 2) if $w \notin L$ then M rejects w .
 - Suppose $w \in L$. Then M_1 accepts w , and M_2 does not accept w . Let t_1 be the number of steps which M_1 takes to accept w . Then every iteration with $t < t_1$ eventually ends, since neither machine accepts and each machine is only simulated for a bounded number of steps. Thus the iteration $t = t_1$ is eventually reached, and in this iteration M_1 accepts w , causing M to accept.
 - Suppose $w \notin L$, i.e. $w \in \bar{L}$. Then M_2 accepts w , and M_1 does not accept w . Let t_2 be the number of steps which M_2 takes to accept w . Then every iteration with $t < t_2$ eventually ends (for reasons symmetric to those outlined above), and in the iteration with $t = t_2$, M_2 accepts w , causing M to reject.

Random access machines (RAMs)

- A RAM is an abstraction of a "real-world" computer. It consists of:
 - k registers R_1, \dots, R_k which store integers
 - A program $P = I_1, I_2, \dots, I_t$, where each I_j is an instruction like

ADD a, b, c	(set $R_a = R_b + R_c$)
SUB a, b, c	(set $R_a = R_b - R_c$)
MUL a, b, c	(set $R_a = R_b \times R_c$)
SET a, n	(set $R_a = n$)
IF $R_a > 0$ THEN GOTO i	(if $R_a > 0$ then execute I_i , else execute I_{j+1})
 - Other instructions similar to those above (e.g. arithmetic, set, branch, etc.)

- The input is a number n in register R_1 ; the other registers are initially set to zero.
- The language of a RAM is a set of integers, i.e. $L(M) \subseteq \mathbb{Z}$; but for $L \subseteq \mathbb{Z}$ one can define $L_{0,1} \subseteq \{0,1\}^*$ to be the set of binary strings w s.t. the number represented by w is in L .
- **Theorem:** If $L \subseteq \mathbb{Z}$ is recognizable/decidable by a RAM, then $L_{0,1}$ is recognizable/decidable by a TM.
- Proof (sketch): Let M be a RAM with k registers and program $P = I_1, I_2, \dots, I_t$. Define a TM M' as follows:
 - M' has k tapes corresponding to the registers of M
 - M' has states q_1, \dots, q_t corresponding to the instructions I_1, \dots, I_t ; M' will be in state q_j if in the simulation of M' , M' is just about to execute instruction I_j .
 - On input w , M' writes 0 on tapes 2, \dots , k (and keeps w on tape 1), and goes to state q_1 .
 - To simulate a step of M' : if M is in state q_j , the action depends on the instruction I_j .
 - If I_j is an arithmetic instruction, e.g. ADD a, b, c , then M' erases the contents of tape a and performs binary addition on tapes b and c , storing the result on tape a
 - If I_j is a set instruction, e.g. SET a, n , then M' erases tape a and writes the binary representation of n onto tape a (note that M has the integer n “hard-coded” into its program, and M' similarly has the binary representation of n hard-coded into its description)
 - If I_j is a branch instruction, e.g. IF $R_a = 0$ THEN GOTO i , then M' checks if tape a contains only zeros; if so, M' switches to state q_i , otherwise it switches to state q_{j+1} .