After this tutorial you should be able to:

1. Show how to simulate TMs with certain properties by TMs with other properties (typically to show that the class of TMs is robust).
2. Show that certain languages are decidable by providing basic, nondeterministic, or multi-tape deciders for them.

Variations of the basic TM

Problem 1. Show that every left-bounded TM is equivalent to a doubly-infinite TM.

Note. In class you saw how to show that every doubly-infinite TM is equivalent to a left-bounded TM. This question is not asking you to "undo" that construction.

Solution 1. Idea: Start by moving left one cell and placing a special symbol, say $\vdash$, to be used as a left-end marker. Then move right one cell. Then simulate the given left-bounded TM. Whenever the head is over the left-end marker, simply move it one cell to the right.

Problem 2. Show the every TM M with input alphabet $\Sigma = \{0,1\}$ is equivalent to a TM M' with $\Gamma = \{0,1,B\}$, in other words, except for the blank tape symbol B and the input alphabet, there are no additional tape symbols.

Solution 2. Suppose $\Gamma = \{0,1,2,\cdots,N\}$. Here is a high-level description. Then the machine M' simulates M by writing $0^i 1^{N-i}$ instead of $i \in \Gamma$ on the tape. E.g., 0 is encoded by 1^N, and 1 is encoded by 01^{N-1}, and 2 is encoded by 001^{N-2}, etc. Note that all these strings have the same length, i.e., N. Here is an implementation-level description. To start, it needs to replace the input word $u \in \{0,1\}^*$ by its encoding, e.g., the input 110 is replaced by $01^{N-1}01^{N-1}1^N$. Then M' reads/writes a block of N bits to know what M was reading/writing.

Problem 3. Give a high-level and implementation-level description of a multi-tape Turing machine M deciding the language of binary strings that read the same forward as backwards. E.g., $0, 010, 11011011$ are accepted and $110, 01$ are not.

Can you describe the single-tape Turing machine that is equivalent to M (using the construction from lectures). What is its alphabet?

Solution 3. We give an implementation-level description.

The machine M has tape alphabet $\Gamma = \{0,1,\_\}$ has two tapes.

1. Moving the head on tape 1 to the end of the input (i.e., move right until it sees a blank, then move left one cell).

2. Copy the reverse of the input onto tape 2 (i.e., write under head 2 the current symbol under head 1, and move head one left one cell, head two right one cell, until the start of the word is reached on tape 1).
3. Check the two words are equal (i.e., move head 2 to the start of the word, check the symbol under head 1 is equal to the symbol under head 2, if so move both heads one cell to the right and repeat, otherwise reject; accept if the heads pass the end of the words).

The one-tape machine works in the same way, except that it's tape alphabet is consists of the possible 'columns' of the tapes. That is, let $\bullet$ be a new symbol (to mark the position of a head). Then the new tape alphabet is $(\{0,1,\_\} \times \{bullet,\_\})^2$. E.g., the tape symbol $(0,\bullet,1,\_)$ means that the first track has a 0 written on it and its head is in that position, the second track track has a 1 written on it and its head is not in that position.

Problem 4. Give a high-level description and an implementation-level description of a nondeterministic TM N that decides the language $\{0^{n_1}10^{n_2}1\cdots 0^{n_k}1 : n_i = n_j \text{ for some } i,j\}$ over alphabet $\{0,1\}$.

E.g., 00010010100100001 should be accepted, but 00010010100001 should not.

Can you describe the deterministic Turing machine that is equivalent to N (using the construction from lectures).

Solution 4. The NTM takes a left to write pass, nondeterministically marking two of the segments, say $0^{n_i}1$ and $0^{n_j}1$ (e.g., it can delete all the bits in the unchosen segments). It then uses a deterministic algorithm to check if $n_i = n_j$ (similar to the TM for $\{0^n1^n : n \geq 0\}$.

Decidability

Problem 5. Each of the following languages is decidable. Pick any two and show they are decidable.

$$\begin{aligned}
L_{\text{NFA-acceptance}} &= \{\langle N, w\rangle \mid N \text{ is an NFA that accepts } w\}\\
L_{\text{RE-acceptance}} &= \{\langle R, w\rangle \mid R \text{ is a RE and } w \in L(R)\}\\
L_{\text{DFA-emptiness}} &= \{\langle D\rangle \mid D \text{ is a DFA and } L(D) = \emptyset\}\\
L_{\text{DFA-equivalence}} &= \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}
\end{aligned}$$

Solution 5.

1. For NFA-acceptance: apply the subset construction to get a DFA D equivalent to N, and then simulate D on input w.

2. For RE-acceptance: form a DFA D equivalent to R and then simulate D on input w.
3. For DFA-emptiness: use graph-search (e.g., DFS, BFS) to check if any accepting state of D is reachable from the initial state of D.
4. For DFA-equivalence: form DFAs for $L_1 = L(A) \setminus L(B)$ and $L_2 = L(B) \setminus L(A)$ and check that $L_1 = \emptyset$ and $L_2 = \emptyset$ using the DFA-emptiness decider.

Problem 6. Argue that every regular language is Turing-decidable.

Note. This is similar to, but not the same as the DFA-acceptance problem.

Solution 6.

Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$ build a TM M_D that takes $w \in \Sigma^*$ as input and simualtes D as follows. It does this with the following transitions: for every transition $\delta(q, a) = q'$ of the DFA we add a transition of M_D that maps (q, a) to (q', a, R), i.e., don't change the input, move to the right, and change state. In addition, we have a transition that maps $(q, \_)$ to $(q_{accept}, \_, R)$ for every $q \in F$, and a transition that maps $(q, \_)$ to $(q_{reject}, \_, R)$ for every $q \notin F$.