# Practice Problems and Solutions from Chapter # 3

**3.15** Show that the collection of Turing-recognizable languages is closed under the operation of

- $^{A}$**a.** union.
- **b.** concatenation.
- **c.** star.
- **d.** intersection.
- **e.** homomorphism.

Sol.

**b.** For any two Turing-recognizable languages $L_1$ and $L_2$, let $M_1$ and $M_2$ be the TMs that recognize them. We construct a NTM $M'$ that recognizes the concatenation of $L_1$ and $L_2$:

```
"On input $w$:
    1. Nondeterministically cut $w$ into two parts $w = w_1w_2$.
    2. Run $M_1$ on $w_1$. If it halts and rejects, *reject*.
    3. Run $M_2$ on $w_2$. If it accepts, *accept*. If it halts and rejects, *reject*."
```

If there is a way to cut $w$ into two substrings such $M_1$ accepts the first part and $M_2$ accepts the second part, $w$ belongs to the concatenation of $L_1$ and $L_2$ and $M'$ will accept $w$ after a finite number of steps.

**c.** For any Turing-recognizable language $L$, let $M$ be the TM that recognizes it. We construct a NTM $M'$ that recognizes the star of $L$:

```
"On input $w$:
    1. Nondeterministically cut $w$ into parts so that $w = w_1w_2 \cdots w_n$.
    2. Run $M$ on $w_i$ for all $i$. If $M$ accepts all of them, *accept*. If it halts and
       rejects any of them, *reject*."
```

If there is a way to cut $w$ into substrings such $M$ accepts the all the substrings, $w$ belongs to the star of $L$ and $M'$ will accept $w$ after a finite number of steps.

**d.** For any two Turing-recognizable languages $L_1$ and $L_2$, let $M_1$ and $M_2$ be the TMs that recognize them. We construct a TM $M'$ that recognizes the intersection of $L_1$ and $L_2$:

```
"On input $w$:
    1. Run $M_1$ on $w$. If it halts and rejects, *reject*. If it accepts, go to stage 3.
    2. Run $M_2$ on $w$. If it halts and rejects, *reject*. If it accepts, *accept*."
```

If both of $M_1$ and $M_2$ accept $w$, $w$ belongs to the intersection of $L_1$ and $L_2$ and $M'$ will accept $w$ after a finite number of steps.

**3.18** A ***Turing machine with doubly infinite tape*** is similar to an ordinary Turing machine, but its tape is infinite to the left as well as to the right. The tape is initially filled with blanks except for the portion that contains the input. Computation is defined as usual except that the head never encounters an end to the tape as it moves leftward. Show that this type of Turing machine recognizes the class of Turing-recognizable languages.

Sol.

A TM with doubly infinite tape can simulate an ordinary TM. It marks the left-hand end of the input to detect and prevent the head from moving off of that end.

To simulate the doubly infinite tape TM by an ordinary TM, we show how to simulate it with a 2-tape TM, which was already shown to be equivalent in power to an ordinary TM. The first tape of the 2-tape TM is written with the input string and the second tape is blank. We cut the tape of the doubly infinite tape TM into two parts, at the starting cell of the input string. The portion with the input string and all the blank spaces

to its right appears on the first tape of the 2-tape TM. The portion to the left of the input string appears on the second tape, in reverse order.

**3.19** A ***Turing machine with left reset*** is similar to an ordinary Turing machine, but the transition function has the form

$$\delta\colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{R}, \mathrm{RESET}\}.$$

If $\delta(q, a) = (r, b, \mathrm{RESET})$, when the machine is in state $q$ reading an $a$, the machine's head jumps to the left-hand end of the tape after it writes $b$ on the tape and enters state $r$. Note that these machines do not have the usual ability to move the head one symbol left. Show that Turing machines with left reset recognize the class of Turing-recognizable languages.

Sol.

We simulate an ordinary TM with a reset TM that has only the RESET and R operations. When the ordinary TM moves its head right, the reset TM does the same. When the ordinary TM moves its head left, the reset TM cannot, so it gets the same effect by marking the current head location on the tape, then resetting and copying the entire tape one cell to the right, except for the mark, which is kept on the same tape cell. Then it resets again, and scans right until it find the mark.

**3.21** A ***queue automaton*** is like a push-down automaton except that the stack is replaced by a queue. A ***queue*** is a tape allowing symbols to be written only on the left-hand end and read only at the right-hand end. Each write operation (we'll call it a *push*) adds a symbol to the left-hand end of the queue and each read operation (we'll call it a *pull*) reads and removes a symbol at the right-hand end. As with a PDA, the input is placed on a separate read-only input tape, and the head on the input tape can move only from left to right. The input tape contains a cell with a blank symbol following the input, so that the end of the input can be detected. A queue automaton accepts its input by entering a special accept state at any time. Show that a language can be recognized by a deterministic queue automaton iff the language is Turing-recognizable.

Sol.

First, we show that any queue automaton $Q$ can be simulated by a 2-tape TM $M$. The first tape of $M$ holds the input, and the second tape holds the queue. To simulate reading $Q$'s next input symbol, $M$ reads the symbol under the first head and moves it to the right. To simulate a push of $a$, $M$ writes $a$ on the leftmost blank square of the second tape. To simulate a pull, $M$ reads the leftmost symbol on the second tape and shifts that tape one symbol leftward. Multi-tape TMs are equivalent to single tape TMs, so we can conclude that if a language can be recognized by a queue automaton, it is Turing-recognizable.

Now we show that any single-tape, deterministic TM $M$ can be simulated by a queue automaton $Q$. For each symbol $c$ of $M$'s tape alphabet, the queue alphabet of $Q$ has two symbols, $c$ and $\dot{c}$. We use $\dot{c}$ to denote $c$ with $M$'s head over it. In addition, the queue alphabet has end-of-tape marker symbol \$.

Automaton $Q$ simulates $M$ by maintaining a copy of $M$'s tape in the queue. $Q$ can effectively scan the tape from right to left by pulling symbols from the right-hand side of the queue and pushing them back on the left-hand side, until the \$ is seen. When the dotted symbol is encountered, $Q$ can determine $M$'s next move, because $Q$ can record $M$'s current state in its control. Updating the queue so that it represents $M$'s tape after the move requires another idea. If $M$'s tape head moves leftward, then the updating is easily done by writing the new symbol instead of the old dotted symbol, and moving the dot one symbol leftward. If $M$'s tape head moves rightward, then updating is harder because the dot must move right. By the time the dotted symbol is pulled from the queue, the symbol which receives the dot has already been pushed onto the queue. The solution is to hold tape symbols in the control for an extra move, before pushing them onto the queue. That gives $Q$ enough time to move the dot rightward if necessary.