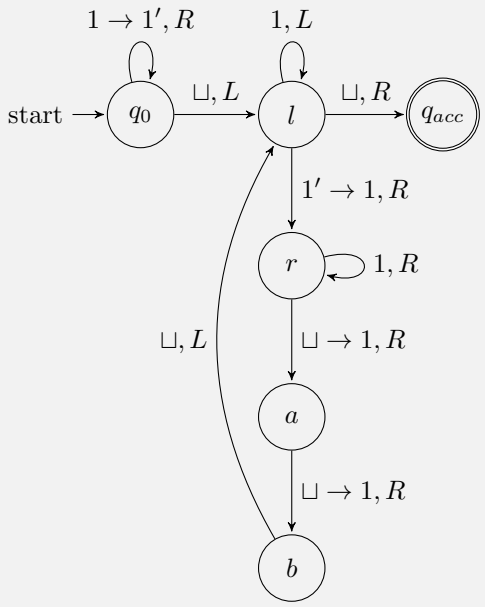
Q1 To show computability, we construct Turing machines computing them.

a) Let the tape symbols be $\Gamma = \{1, 1', \sqcup\}$. On a high level, the Turing machine marks all the 1s on the tape as $1'$; then, for every $1'$, the machine appends two 1s and changes the $1'$ back into 1. The Turing machine is as follows:

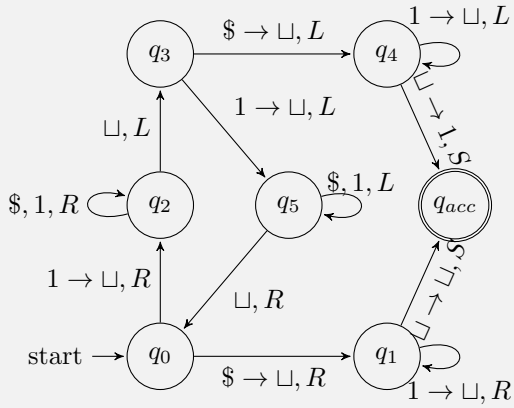
TM



In state q_0, the machine marks all 1 as $1'$. Then, the machine changes the leftmost $1'$ to 1 (state l) and adds two 1's at the end (states a and b). This continues until there is no $1'$. Then the machine accepts the input with head pointing at the first nonblank symbol on tape.

b) Let the tape symbols be $\Gamma = \{1, \$, \sqcup\}$. On a high level, the Turing machine erases 1s on the left and right ends of the configuration. And if the left side runs out of 1 first, then x is at most y. The Turing machine is as follows:

TM



In state q_0: if the machine reads 1, it erases the 1 and moves to the rightmost nonblank symbol (to state q_3 via q_2); otherwise, if the machine reads \$, then that means $x \leq y$, so it erases the entire tape content and accepts (outputting 0); Similarly, if the \$ is read in q_3, then the machine leaves only 1 on the tape because $x > y$.

Q2

Lemma. *Every Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ with one-way infinite tape with left and right moves is equivalent to some TM with one-way infinite tape with right move and left reset.*

Proof. We construct a TM M' with one-way infinite tape with right move and left reset so that it simulates M. On a high level, M' simulates a left move of M by marking the current tape position with a dot (meaning that if the head is pointing at x, then $\dot{x}$ will be written there) and "transporting" the tape content (including blanks) to the right end. The machine first marks all blank symbols on the left of the dotted position with a hat $\hat{}$; this means these blank symbols will be moved to the right end as well. We use the \$ sign to separate transported content from those that have not been transported; it will be placed at the first blank after the non-blank tape content. The moving is done by repeatedly cutting and pasting the leftmost character to the first blank after the non-blank portion of the tape. While transporting a character y from left to right, M' checks if y was 2 positions left of the marked character (e.g $\dot{x}$): if so, $\dot{y}$ will be written on the right instead of y, and the dot on x will be removed. The edge case where no such y exists happens when $\dot{x}$ is the first or second character on tape—this will be detected and handled separately. Also, since M' left shifts by moving tape content right, M' puts a bar $\bar{}$ over the first input symbol to mark the left end of the tape of M—call this the *virtual leftmost position*.

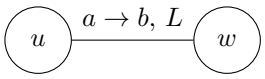
Formally, let $\Gamma' = \Gamma \cup \dot{\Gamma} \cup \{\$, \hat{\sqcup}, \hat{\sqcup}\} \cup \bar{\Gamma}$. The set of states and the transition function will be defined informally below. Note that this is valid since, in the following algorithm, only finite amount of data is stored (so finitely many states suffice).

Finally, we can define $M' =$"On input w:

(a) M' has the same tape content and state initially as M except M' puts a bar on the first input symbol. And in the future if M' wishes to overwrite this symbol, the new symbol will have a bar as well.

(b) If M makes a right move, M' moves right as well (this means $\delta' = \delta$ on the inputs that result in a right move).

(c) If M makes a right move, then M' does exactly what M does.

(d) If M makes a left move: 

i. If the head is at the virtual leftmost position (meaning that the current symbol has a bar on it), then M' writes $\bar{b}$ at the current position and transitions into state w, just as M would.

ii. If the head is not at the virtual leftmost position and the head is second on the actual tape (can be checked by doing a left reset and moving right once), then M' writes b, transition to state w, and does a left reset.

iii. If the head is not at the virtual leftmost position and the head is not second on the actual tape (this implies that the head is at least third on the tape), then:

A. M' writes $\dot{b}$ in place of a and puts \$ at the first blank after the tape content;

B. M' marks every $\sqcup$ as $\hat{\sqcup}$ on the left of \$;

C. For every symbol on the left of \$, M' erases it from the tape and transports it to the the right of \$ (transportation of a symbol y can be done by using a state of the form (q_{trans}, y));

D. While transporting a symbol y, M' checks if it used to be 2 positions to the left of $\dot{a}$, if so; M' puts a dot on y.

E. After transportation, M' removes the hats on blank symbols and the dot on $\dot{x}$.

F. Finally, M' resets and moves right until it sees a dotted symbol $\dot{y}$; it removes the dot, moves right once more, and transitions to state w.

" □

Theorem. *Turing machines with left reset recognizes the same languages as standard Turing Machines.*

Proof. Given any Turing machine M' with left reset, we can simulate it using a Turing machine M with one-way infinite tape and L, R moves. Indeed, M can simulate right moves by doing exactly the same. Also, M keeps a bar on the leftmost symbol on the tape and when M' does a left reset, M moves left until a barred symbol is reached. This simulates a left reset using M. This means if a language is recognizable by M', then M recognizes it too. So the recognizable languages of left reset TM's is a subset of that of singly infinite tape TM's (with L, R moves).

By lemma above, for any singly infinite TM M with L, R moves, there is some left reset TM M' that simulates it. This shows the recognizable languages of singly infinite tape TM's (with L, R moves) is a subset of that of left reset TM's.

Recall that that Turing machines with one way infinite tape recognizes the same set of languages as the standard Turing machines. Thus, we are done. □

Q3

Theorem. *A language $A \subseteq \Sigma^*$ is semi-decidable if and only if there is a decidable binary relation $R \subseteq \Sigma^* \times \Sigma^*$ such that for any $x \in \Sigma^*$, $x \in A$ if and only if there is a some $y \in \Sigma^*$ for which $(x, y) \in R$.*

Proof. ($\longrightarrow$) Suppose $A \subseteq \Sigma^*$ is semi-decidable[1]. This means there is an enumerator E for A (this fact is proved in lecture). Let $a \in \Sigma$. Define a binary relation $R \subseteq \Sigma^* \times \Sigma^*$ by xRa^n if and only if x is the nth output of E.

Clearly, any string $x \in \Sigma^*$ has some a^k such that xRa^k if and only if x is in A.

To see that R is decidable, we construct a Turing machine M that decide it. Consider the Turing machine defined as follows: on input x, y (take the tape alphabet to be $\Sigma \cup \{,\} \cup \{E\text{'s tape alphabet}\}$),

- If y is not of the form a^k for some $k \in \mathbb{N}_{>0}$, reject;

[1]Note that if the language is finite, then the theorem is trivial: any finite language is decidable and clearly the relation $R \subseteq \Sigma^* \times \Sigma^*$ defined by $xR\epsilon$ if and only if $x \in A$ is finite as well; R satisfies the requirement in the theorem and it is decidable because it is finite. So, assume A is infinite.

- Execute E until it produced k outputs;
- If the kth output is x, accept; otherwise, reject.

Our machine always terminates and it clearly decides R: M accepts some input x, a^k iff x is the kth input of E iff $x \in A$.

($\longleftarrow$) Suppose some Turing machine M decides the binary relation R satisfying the condition given in the theorem, we construct a Turing machine M' that recognizes A. Consider the Turing machine defined as follows: on input x,
For each $i = 1, 2, \ldots$,

- Generate the ith string in Σ^*'s lexicographical order, call it s_i[2];
- Run M on x, s_i.
- If M accepts x, s_i, accept; otherwise, continue to the next i;

This Turing machine clearly recognizes A: M' accepts x iff M accepts x, s_i for some s_i iff (since M decides R) xRs_i iff $x \in A$. □

Q4

Theorem. *A language $L \subseteq \Sigma^*$ is decidable if and only if some enumerator E prints elements of L in lexicographic order.*

Proof. ($\longrightarrow$) Suppose $L \subseteq \Sigma^*$ is decidable by some Turing machine M. We define the enumerator E as follows:
For $i = 1, 2, \ldots$,

- Generate the ith string s_i (lex. order) in Σ^*;
- Run M on s_i; if M accepts, E prints s_i onto the tape, followed by a delimiter.

It is obvious that E enumerates L lexicographically: the strings are generated lexicographically and a string s_i is printed if and only if M accepts s_i. Note also that since strings are generated lexicographically, every string in L will eventually be fed into M and subsequently be printed.

($\longleftarrow$) Suppose some enumerator E prints elements of $L \subseteq \Sigma^*$ in lexicographic order.

Case 1: L is infinite. Define the Turing machine M as follows: on input x,
For each output e of E,

- Check if $x = e$; if so, accept;
- If $|e| > |x|$, reject.

Clearly, M decides L. Indeed, L being infinite means eventually $|e| > |x|$, causing the machine to terminate. Also, M accepts a string if and only if it is printed by E iff and only if the string is in L.

Case 2: L is finite Any finite language is decidable. So this case is trivial. □

[2]This is possible since we showed in tutorial that a TM can generate strings in lex. order