Solution to Problem 1

1. We construct a turing machine that lists all possible strings in the alphabet in order and simulates the turing machine M on each of the possible strings; the machine lists each string accepted by M. Since M may not halt on some strings it does not accept, we have to interleave the simulations. If \( w_0,w_1,w_2 \ldots \) are the strings in the alphabet, in round \( k \) we simulate each string \( w_i \) \( i \leq k \) for \( (k - i) \) steps; once a string is accepted, we of course remove that string from the list of strings to be simulated. For example, in round 1 we run M on \( w_1 \) for just 1 step; then in the next round we run M on \( w_1 \) for 2 steps and on \( w_2 \) for 1 step and so on.
2. Show that \( \{ a^n b^n c^n d^n e^n f^n g^n h^n \mid n \geq 1 \} \) can be written as the intersection of two context-free languages.

\[ \{ a^i b^j c^k d^l e^k g^k h^l \mid i, j, k, l \geq 1 \} \cap \{ a^m b^m c^o d^o e^o f^o g^o h^o \mid m, o \geq 1 \} \]

Both languages are context-free, because only two numbers are being compared at a time in each language. By intersecting these two languages, the numbers of each of \( a, b, c, d, e, f, g, h \) are equal.

Different letters are used in the superscripts (i,j,k,l and m,o,p) for each language to illustrate that it’s not possible to intersect two languages and guarantee that the same variable name is equal between languages.
Homework 9 Problem 3 Solution

Problem: Show that is the halting problem for Turing machines starting with a blank tape was decidable, then whether a Turing machine M halts when started on input x would be decidable.

Solution: Construct a new machine M' such that M' starts with a blank tape. Its first step is to write the input x on the tape. It will then run as a usual Turing machine with input x on the tape. If the halting problem for a Turing machine starting on a blank tape were decidable, then clearly this problem would be decidable, because M' started on a blank tape. But, this problem has been shown to be undecidable, so it follows that the problem of whether a Turing machine M halts when started on input x is undecidable.
Since $L_1, L_2, \ldots, L_k$ are R.E. each has a turing machine that halts on accepting these, let these be $M_1, M_2, \ldots, M_k$.

Construct $M_1'$ --- as such

$M_1'$ accepts if $M_1$ accepts and $M_1'$ rejects if any of $M_2, M_3, \ldots, M_k$ accept. Since any input has to be in one of the languages $L_1, L_2, L_3, \ldots, L_k$ (the union of $L_1, L_2, \ldots, L_n$ is $\Sigma^*$ as given) and cannot be in two languages (given) $M_1'$ will either accept or reject. So it is recursive. Similarly we can show that for each language we can build a turing machine that halts on all inputs and accepts the strings in the language.
Problem 5

We will give a rough sketch of the solutions. Hence the solutions will not precise at some places. We will assume that the two languages in the question are \( L_1 \) and \( L_2 \) and they are accepted by Turing machines \( M_1 \) and \( M_2 \). \( M_1 \) and \( M_2 \) always halt if languages are recursive, otherwise, they are not guaranteed to halt on the inputs not in the language.

Another piece of notation: If we claim that a particular language is \( RE \) or \( R \), we construct an appropriate Turing machine, which will be called \( N \) throughout the problem.

(a)

Both \( RE \) and \( R \) sets are closed under union. For \( RE \) sets, given an input \( x \), \( N \) simulates \( M_1 \) and \( M_2 \) both on the input \( x \) (on two different tapes), time sharing between them (one step at a time). If any of the machine halts and accepts, \( N \) does so too. If \( x \in L_1 \cup L_2 \), at least one machine halts and accepts, in which case \( N \) also accepts. If \( N \) accepts, then one of \( M_1 \) or \( M_2 \) must have halted and accepted, so \( x \in L_1 \cup L_2 \).

The construction for recursive languages is the same except that we do not have to time-share (although it is okay to time share). The same proof as above goes through.

(b)

This case is very similar to the last part except that the condition for \( N \) to accept is that both \( M_1 \) and \( M_2 \) accept.

In the \( RE \) case, we can choose not to do the time sharing (although it does hurt to do the share time), since both machines need to accept the input for \( N \) to accept it.

(c)

We recall the definition of concatenation.

\[
\text{Concatenation}(L_1, L_2) = \{ z | z = x_1x_2, x_1 \in L_1, x_2 \in L_2 \}.
\]
We claim that $RE$ as well as $R$ languages are closed under concatenation. The following algorithm works for both cases.

Given an input $z$, the machine $N$ guess one index between 1 and $|z|$ for where to partition the string $z$. It partitions the string $z$ at the guessed index, writes two parts on two different tapes and simulate $M_1$ on the first part. If this simulation halts and accepts, then it goes on to simulate $M_2$ on the second part. If this simulation also accepts, then the machine $N$ accepts.

It is easy to see that if $z \in L_1 \cdot L_2$, then there is at least one run of the Turing machine $N$ which accepts $z$ and if Turing machine $N$ accepts $z$ on some run, then $z \in L_1 \cdot L_2$.

The only tricky part is that we do not have to time-share even in the $RE$ case, which is not surprising since $M_1$ and $M_2$ both have to accept for $N$ to accept.

(d)

We recall the definition of $L_i^*$ below.

$$L_i^* = \{x_1x_2 \ldots x_k | k \in \mathbb{N}, x_i \in L_i \text{ for all } i = 1, 2, \ldots, k\}.$$  

The algorithm for this part is almost similar to the last part with one minor difference. The machine $N$ first guesses a number $k$ for how many segment the input string should be partitioned in, and then goes on to guess $k$ distinct indices for the partition. Then it runs $M_1$ on the first partition. If the first simulation reject, $N$ rejects too. If it accepts, then it simulates $M_1$ on the second partition, and so on. If all simulations accepts, $N$ accepts too.

The proof is a minor modification to the last part’s proof.

(e)

Let $h : \Sigma \rightarrow \Gamma^*$ be a homomorphism. For the $RE$ languages, we want to prove that if $L_1 \subset RE$, then $L_2 = h(L_2) = \{y | \exists x \in L_1 \text{ with } h(x) = y \} \subset RE$.

Given an input string $y \in \Gamma^*$, the machine $N$ guesses a string $x \in \Sigma$, checks that $h(x) = y$ (if not, then reject), and if $h(x) = y$ then starts simulating $M_1$ on $x$. If $M_1$ accepts, so does $N$, if $M_1$ rejects, so does $N$. If $M_1$ does not halt, $N$ also keeps on simulating $M_1$.

It is not that hard to see that if $y \in L_2$, then there indeed exists a string $x \in L_1$ whose $h$-image is $y$. If that is the case, there is a computation path for $N$ which will accept the string $y$. On the other hand, if $y \notin L_2$, then no such string $x$ exists. In that case, all paths of $N$ will either not terminate, or will reject.

For recursive language, we show that $h(L_1)$ might not be recursive. For this, we take a machine $M_{HP}$ which accepts Halting Problem ($HP$). We define valid computations of $M_{HP}$ to be a (finite) sequence of computation (separated by $\#$ symbols) with each one following from the previous one by one step of $M_{HP}$, the state of the first configuration being the start state of $M_{HP}$ and the state in the final configuration being an accept state of $M_{HP}$. We also assume that all (tape alphabet) symbols in the first configuration are hatted (except that the state is not hatted). We claim that the language of all valid computations as defined
above is recursive. The argument is straightforward, a Turing machine can check that one configuration follows from the last one by one step of \( M_{HP} \).

Now we think of a homomorphism which maps all hatted symbols to \( \epsilon \). The resulting language after applying homomorphism is the set of all input accepted by \( M_{HP} \), which is clearly not recursive.

(f)

We prove that both \( RE \) as well as \( R \) is closed under inverse homomorphism. We consider the homomorphism defined in the last part. Given a language \( L_2 \subseteq \Gamma^* \), we recall the definition of \( h^{-1}(L_2) \).

\[
h^{-1}(L_2) = \{ x | h(x) \in L_2 \}.
\]

The machine \( N \) works as follows. Give an input string \( x \), \( N \) computes \( y = h(x) \) and simulates \( M_2 \) on \( y \). If the simulation accepts, \( N \) accepts and if the simulation rejects then \( N \) rejects too. If simulation does halt, then \( N \) does not halt either.

It is an easy argument to show that the language accepted by \( N \) is precisely \( L_1 \). Also, \( N \) is recursive if \( M_2 \) is. This finishes the proof.