

Solution Set 5

Problem 1 Both recursive languages and recursively enumerable languages are closed under intersection. Any recursive language is accepted by a TM that is guaranteed to halt after a finite number of steps, whether or not the input string is in the language. Any recursively enumerable language has a TM that will eventually halt in an accepting state when run on any input that is in the language.

To compute $L_1 \cap L_2$, we can construct a TM that first simulates the TM for L_1 , then simulates the TM for L_2 , and halts in an accepting state if and only if both simulated TMs halt in accepting states.

If L_1 and L_2 are both recursive, then both simulations are guaranteed to halt after a finite number of steps, so the combined TM for $L_1 \cap L_2$ also halts after a finite number of steps.

Note that any input that is in $L_1 \cap L_2$ is also in both languages L_1 and L_2 . If L_1 and L_2 are both recursively enumerable, then when run on an input that is in $L_1 \cap L_2$, both simulations will halt in an accepting state, meaning that the combined TM for $L_1 \cap L_2$ will also halt in an accepting state.

Problem 2 Let L denote the language described in the problem statement. The language L is undecidable, but it is recursively enumerable.

To show that L is undecidable, we do a proof by contradiction using a reduction from L_u . Suppose that L is decidable, meaning that there exists a TM M_L that is an algorithm for L . Given an input $\langle M, w \rangle$, we would like to decide if $\langle M, w \rangle \in L_u$, i.e. if the TM M encoded by the string $\langle M \rangle$ would accept the input w . To do so, we first construct a new machine M' that uses tape alphabet $\Gamma = \{0, 1, 2, B\}$ and does the following: simulate the execution of machine M on input w (using only tape symbols 0, 1, and B to perform the simulation), and if the simulation halts in an accepting state, then write 2 onto the tape. Then we compute the encoding of M' and run the machine M_L on the input $\langle M', w, 2 \rangle$. Machine M' will write 2 to its tape if and only if machine M accepts the input w ; since we assumed machine M_L is an algorithm for L , meaning it always halts in finite time, the whole process of constructing the input $\langle M', w, 2 \rangle$ and running M_L on it constitutes an algorithm for L_u . But we know that L_u is undecidable, meaning that no algorithm for L_u exists. Therefore, our original assumption (that there exists an algorithm M_L for L) is false.

To show that L is recursively enumerable, we describe a Turing machine M_1 that accepts L . Given an input $\langle M, w, X \rangle$ for L , the TM M_1 simulates the machine M on the input string w . If the simulation of M ever writes X to its simulated tape, then M_1 accepts. If the simulation of M halts without writing X to its tape, then M_1 rejects.

Problem 3 When the machine M runs on input w , one of two things must happen for $\langle M, w \rangle$ to be in the language: either the tape head of M never moves to the left, or else the tape head of M never moves to the right. Therefore, all we need to do is to simulate the execution of M on w and keep track of the direction in which the simulated tape head moves. There are three cases:

1. If, during our simulation, the tape head of M ever moves in the opposite direction from the direction of its first move, we can immediately halt and reject the input $\langle M, w \rangle$.
2. If the simulation of machine M on w halts without ever moving in the opposite direction from its first move, we can halt and accept the input $\langle M, w \rangle$.
3. The final case is when the simulation of M on w never halts, but instead runs forever, always moving the tape head in the same direction. We can detect this non-halting case as follows: after the first time that machine M sees a blank on its tape, it has gone off the end of the input. As long as M continues moving in the same direction, it will only see blanks from this point forward.

Suppose machine M has $|Q|$ states. After M has seen a blank on its tape and then made $|Q|$ more transitions, it will have entered $|Q| + 1$ states and seen a blank on the tape in all of them. These states cannot all be distinct, since there are only $|Q|$ distinct states in M ; therefore, machine M must have looped back to a previously-seen state and will continue to loop forever, since it will continue to see a blank tape forever. Thus, once the simulation of M has reached a blank symbol on the tape and proceeded for $|Q|$ more steps without ever moving the tape head in the opposite direction, we can safely stop and accept the input $\langle M, w \rangle$.

Problem 4 The proof is by a reduction from L_{all} . Suppose that L is recursive, meaning that there exists a TM M_L that is an algorithm for L . Using the machine M_L , we can construct an algorithm for L_{all} .

Given an input $\langle M \rangle$, we'd like to decide if $\langle M \rangle \in L_{all}$. We do this by first constructing a TM M^* that always accepts, regardless of its input. Then we run the machine M_L on the input $\langle M^*, M \rangle$. Since we assumed M_L is an algorithm for L , the execution of M_L will halt after a finite amount of time. If M_L accepts the input, then $L(M^*) \subseteq L(M)$. Since the machine M^* accepts all strings, i.e. $L(M^*) = \Sigma^*$, this implies that $L(M) = \Sigma^*$ as well, or in other words, $\langle M \rangle \in L_{all}$. If M_L rejects the input, then $L(M^*) \not\subseteq L(M)$, meaning that $\langle M \rangle \notin L_{all}$. Thus we can use the machine M_L to construct an algorithm for L_{all} . However, we know that L_{all} is undecidable, meaning that no algorithm for L_{all} exists. Therefore, our initial assumption that L is recursive is false, and in fact L is non-recursive.

Problem 5

- a. \emptyset
- b. All palindromes

- c. Suppose that L_{NR} is decidable. Then there is an algorithm for L_{NR} , i.e. a TM that always halts. We can use this TM, which we'll call M_{NR} , to solve the halting problem as follows: given a halting problem input $\langle M, w \rangle$, construct $\langle \widehat{M} \rangle$ and run machine M_{NR} on input $\langle \widehat{M} \rangle$. Since M_{NR} is an algorithm, it will eventually halt. Note that $L(\widehat{M})$ is either the empty language, which is regular, or the language of palindromes, which is non-regular. If M_{NR} halts in an accepting state, that means that $L(\widehat{M})$ was the language of palindromes, meaning that machine M halted on input w , so the input $\langle M, w \rangle$ is in L_H . If M_{NR} halts in a rejecting state, that means that $L(\widehat{M})$ was the empty language, meaning that machine M did not halt on input w , so the input $\langle M, w \rangle$ is not in L_H . Thus, by constructing a TM that runs M_{NR} on the input $\langle \widehat{M} \rangle$, we can construct an algorithm for L_H . But this is impossible, since L_H is undecidable; therefore our original assumption was incorrect, and in fact, L_{NR} is undecidable.

Problem 6 The proof uses a reduction from L_e . Given an input $\langle M \rangle$ for the problem L_e , we will construct an input for the problem L as follows: construct a TM M^* that accepts all inputs, compute its encoding $\langle M^* \rangle$, and concatenate that encoding with $\langle M \rangle$ to produce the input $\langle M^*, M \rangle$.

Suppose that L is recursively enumerable, meaning that there exists a TM M_L that will halt and accept on any input string that is in L . Using the machine M_L , we can construct a Turing machine that accepts the language L_e . Given an input $\langle M \rangle$ for L_e , we construct the input $\langle M^*, M \rangle$ and run the machine M_L on this input. Since $L(M^*) = \Sigma^*$, the intersection of the languages of machines M^* and M is empty if and only if $L(M) = \emptyset$. Therefore, if M_L accepts the input, we can conclude that $\langle M \rangle \in L_e$. Furthermore, whenever $\langle M \rangle \in L_e$, the machine M_L will eventually halt and accept its input, since $\langle M^*, M \rangle \in L$ and we assumed that M_L halts and accepts on all input strings in L . But this implies that we can construct a Turing machine that accepts L_e , even though L_e is not recursively enumerable, which is a contradiction. Thus we conclude that the initial assumption that L is recursively enumerable is false.