

COMP 481: Automata, Formal Languages, and Computability

Spring 2009

Solutions to Homework Assignment #7

1. I used **R** and **D** interchangeably to denote the set of recursive (decidable) languages, and use **RE** and **SD** interchangeably to denote the set of recursively enumerable (semi-decidable) languages.
2. I make use of the facts that $A_{TM} = \{\langle M, w \rangle : M \text{ accepts } w\}$ is in **SD** but not in **D**, and that $\overline{A_{TM}} = \{\langle M, w \rangle : M \text{ does not accept } w\}$ is not in **SD**.
3. I write τ for a reduction function.

1. (a) L_1 **is in R**. TM M^* that decides the languages works as follows on input $\langle M \rangle$. It first finds the length of $\langle M \rangle$, and stores it. Then, it runs M on all inputs of length at most $|\langle M \rangle|$, for at most $|\langle M \rangle|$ steps, and accepts if M accepts at least one of the strings within the specified number of steps.

You might wonder why we limited the length of the strings. Since we bound the number of steps that M runs on an input, then there is no point on looking at any strings that are longer than that number, since if a TM is allowed to run for at most c steps, it is not possible for that TM to “process” any input symbol beyond the c^{th} symbol!

The number of possible inputs is finite, and the number of steps M runs on each input is finite, therefore M is guaranteed to halt and decide the language.

- (b) L_2 **is not in RE**. We prove this by a reduction from $\overline{A_{TM}}$. $\tau(\langle M, x \rangle) = \langle M' \rangle$. M' on input w : it runs M on x for $|w|$ steps; it rejects if M accepts x within $|w|$ steps, and accepts otherwise.

We now prove the validity of the reduction:

- $\langle M, x \rangle \in \overline{A_{TM}} \Rightarrow M$ does not accept $x \Rightarrow M'$ accepts all inputs, and in particular, all even numbers $\Rightarrow M' \in L_2$.
- $\langle M, x \rangle \notin \overline{A_{TM}} \Rightarrow M$ accepts x within k steps $\Rightarrow M'$ rejects all inputs w whose length is greater than or equal to $k \Rightarrow M'$ rejects an infinite number of even numbers $\Rightarrow M' \notin L_2$.

- (c) L_3 **is not in RE**. We prove this by a reduction from $\overline{A_{TM}}$. $\tau(\langle M, x \rangle) = \langle M' \rangle$. M' on input w : it runs M on x for $|w|$ steps; it rejects if M accepts x within $|w|$ steps, and accepts otherwise.

We now prove the validity of the reduction:

- $\langle M, x \rangle \in \overline{A_{TM}} \Rightarrow M$ does not accept $x \Rightarrow M'$ accepts all strings $\Rightarrow L(M')$ is infinite $\Rightarrow M' \in L_3$.
- $\langle M, x \rangle \notin \overline{A_{TM}} \Rightarrow M$ accepts x within k steps $\Rightarrow M'$ rejects all strings whose length is greater than or equal to $k \Rightarrow L(M')$ is finite $\Rightarrow M' \notin L_3$.

- (d) L_4 **is in R**. This is the language of all TM's, since there are no uncountable languages (over finite alphabets and finite-length strings).

- (e) • **L_5 is in RE.** M^* that semidecides the language run the two machines on ε (it interleaves the run between the machines), and accepts if both of them accept.
- **L_5 is not in R.** We prove this by a reduction from A_{TM} . $\tau(\langle M, x \rangle) = \langle M', M' \rangle$. M' on input w : it runs M on x and accepts if M accepts x .

We now prove the validity of the reduction:

- $\langle M, x \rangle \in A_{TM} \Rightarrow M$ accepts $x \Rightarrow M'$ accepts all strings, and in particular it accepts $\varepsilon \Rightarrow \varepsilon \in L(M') \cap L(M') \Rightarrow \langle M', M' \rangle \in L_5$.
- $\langle M, x \rangle \notin A_{TM} \Rightarrow M$ does not accept $x \Rightarrow M'$ does not accept any strings, and in particular does not accept $\varepsilon \Rightarrow \varepsilon \notin L(M') \cap L(M') \Rightarrow \langle M', M' \rangle \notin L_5$.

- (f) **L_6 is not in RE.** We prove this by a reduction from $\overline{A_{TM}}$. $\tau(\langle M, x \rangle) = \langle M_1, M_2 \rangle$. M_1 is a TM that halts and accepts on any input (i.e., $L(M_1) = \Sigma^*$). M_2 on input w : it runs M on x and accepts if M accepts w .

We now prove the validity of the reduction:

- $\langle M, x \rangle \in \overline{A_{TM}} \Rightarrow M$ does not accept $x \Rightarrow M_2$ does not accept any input $\Rightarrow L(M_1) \setminus L(M_2) = \Sigma^* \Rightarrow \varepsilon \in L(M_1) \setminus L(M_2) \Rightarrow \langle M_1, M_2 \rangle \in L_6$.
- $\langle M, x \rangle \notin \overline{A_{TM}} \Rightarrow M$ accepts $x \Rightarrow M_2$ accepts all inputs $\Rightarrow L(M_1) \setminus L(M_2) = \emptyset \Rightarrow \varepsilon \notin L(M_1) \setminus L(M_2) \Rightarrow \langle M_1, M_2 \rangle \notin L_6$.

- (g) **L_7 is not RE.** Reduction from $\overline{A_{TM}}$. $\tau(\langle M, w \rangle) = \langle M_1, M_2 \rangle$.

M_2 on input x : reject. (i.e., $L(M_2) = \emptyset$).

M_1 on input y : run M on w ; if M accepts w , check whether y is of the form $\langle M', z \rangle$, where M' is a TM, and z is a string. If so, run M' on z . If M' accepts, M_1 accepts.

You can now prove that: (1) If $\langle M, w \rangle \in \overline{A_{TM}}$ then $L(M_1) = \emptyset$ (in this case $L(M_1) \leq_m L(M_2)$ since $\emptyset \leq \emptyset$, and hence $\langle M_1, M_2 \rangle \in L_7$), and (2) if $\langle M, w \rangle \notin \overline{A_{TM}}$ then $L(M_1) = A_{TM}$ (in which case $L(M_1) \not\leq_m L(M_2)$, since $A_{TM} \not\leq \emptyset$; why?).

Therefore, $\overline{A_{TM}} \leq L_7$ and hence L_7 is not in RE.

- (h) **L_8 is in R.** A TM M' that decides it works as follows on input x .

If $x \neq \langle M \rangle$, where $\langle M \rangle$ is a description of a DFA, then reject.

Construct a PDA M'' where $L(M'') = \{ww^R : w \in \{a, b\}^*\}$.

Construct a PDA M^* where $L(M^*) = L(M'') \cap L(M)$.

Check if $L(M^*) = \emptyset$. If so, reject; otherwise, accept. (we've seen how to decide whether the language of PDA is empty).

2. (a) **No.** L_{Σ^*} is not in SD (we saw the proof in class). Hence, L_{Σ^*} is not SD-Complete.

- (b) **Yes.** A_{TM} is in SD. We now prove that every language $L \in SD$ reduces to A_{TM} .

If L is in SD, then there exists a TM, M that semidecides it. The reduction from L to A_{TM} is as follows:

$\tau(w) = \langle M, w \rangle$, where M is the machine that semidecides L .

Prove the validity of the reduction. It's simple!

3. There exists an undecidable unary language, since the number of unary languages is uncountable whereas the number of decidable languages is countably infinite! You need to know (understand and be able to prove) these facts.
4. \overline{A} is in RE, and M_1 semi-decides it.
 \overline{B} is in RE, and M_2 semi-decides it.
 We describe language C by a TM, M , that decides it:
 M on input w :
 run M_1 and M_2 on w (in interleaved mode).
 If M_1 accepts, M rejects; If M_2 accepts, M accepts.
 Notice that M always halts; we take $C = L(M)$.
5. L is in R. $L = \{0\}$ or $L = \{1\}$, and for each possibility there is a TM that decides it!
6. **NO!** Take $A = \{a^n b^n : n \geq 0\}$, and $B = a^*$, and let the reduction from A to B be: if w is of the form $a^n b^n$, erase all the b 's; otherwise, return b .
7. **May be.** Take $L_1 = \emptyset$ and $L_2 = \Sigma^*$, both of which are decidable. We know that $L_1 \subseteq A_{TM} \subseteq L_2$, and A_{TM} is not decidable. On the other hand, we know $L_1 \subseteq a^* \subseteq L_2$, and a^* is decidable.
8. (a) **NEVER TRUE.** reductions are transitive, and since A reduces to B , and B reduces to C , we conclude that A reduces to C . Therefore, if $A \notin R$, it can't be that $C \in R$.
 (b) **MAYBE TRUE.**
 (c) **CERTAIN to be TRUE.** If C is in R, and since D reduces to C , then by the reduction theorem, it follows that D is in R. Since R is closed under complement, then the complement of D is also in R.
 (d) **CERTAIN to be TRUE.** C is in RE implies that both B and D are in RE (by the reduction theorem), and so is their intersection (RE langs are closed under intersection).
9. If M is a standard single tape TM, then it is also a doubly infinite tape TM that does not move its reading head to the left of a certain cell. Therefore, M is a special case doubly infinite tape TM.
 Now assume M is a doubly infinite TM, and we show an equivalent standard single tape TM M' that recognizes the same language. M' is created from M with the following modifications:
 - (a) M' first puts a special symbol (that is not in the tape alphabet of M) $\#$ to the left of the input string.
 - (b) After that, M' works exactly like M with the difference that whenever M' moves its reading head left and encounters $\#$, it shifts its tape contents by one cell to the right, thus vacating one cell immediately to the right of the $\#$ symbol, and places the reading head under that cell.

In other words, we simulate the doubly infinite tape TM by shifting the content of the tape to the right every time the machine has to move left of its “left boundary”.

10. A TM with stay put instead of left is equivalent to a finite automaton (DFA). Clearly, a DFA can be simulated by a TM of this type.

In the other direction, suppose we have a TM M of this type.

- $\delta(p, a) = (q, b, R)$ in M can be simulated by $\delta(p, a) = q$ in the NFA. Notice that remembering that the a was replaced by a b when moving right is not needed, since the TM cannot go back to the left and use that b . In other words, replacing the a by a b in M does not affect its computation.
- $\delta(p, a) = (q, b, S)$ in M can be simulated by $\delta(p, a) = q_b$ in the NFA, so that any move in M on (q, b) can be simulated in the NFA by a move on (q_b, ε) .

Therefore, M can be simulated by an NFA, and hence this type of TMs is equivalent to finite automata, and hence they recognize exactly the set of regular languages.

11. Let L be an infinite language in RE. Then, there is an infinite enumerator E for L . Let E' be an enumerator which works as follows:

—Let $x = \varepsilon$

—Run E

—When E prints string y , if $y \geq x$ (in lexicographical order)

print y .

$x = y$.

E is an infinite enumerator. E' prints the first string that E prints. After that, E' prints strings in lexicographical order. Notice that the number of strings that are lexicographically shorter than a string x that's been already printed is finite. Since L is infinite, E would eventually print a string that is lexicographically larger or equal to x . Further, E would print an infinite number of such strings.

Therefore, E' is an infinite lexicographical enumerator of a subset L' of L . Hence, L' is an infinite recursive subset of L .