CS 273: Formal Models of Computation, Spring '06
Problem Set 11: SOLUTIONS

1. Here is a list of possible closure properties for recursive and recursively enumerable languages

   (a) Recursive languages are closed under reversal.

   (b) Recursively enumerable languages are closed under reversal.

   (c) Define substrings(L) to be $\{w : \exists x \exists y$ such that $xwy \in L\}$. If L is recursive, then so is substrings(L)

   (d) If L is recursively enumerable, then so is substrings(L)

   (e) Define lengths(L) to be $\{1^n : \exists w \in L, |w| = n\}$. If L is recursively enumerable, then so is lengths(L)

   (f) Define biglengths(L) to be $\{1^n : \exists w \in L, |w| > n\}$. If L is any language whatsoever, biglengths(L) is regular. (Hint: this is true! Consider two cases: L is finite, and L is infinite.)

   For each of these properties, say if it is true or false. If you answer "true," give a short but convincing argument that supports your assertion, touching on the critical points of the proof. If you answer "false," give a brief intuition for why it's false. (Upgrading this intuition to a formal proof is unnecessary and might be very hard.)

   SOLUTION

   (a) True. Let L be a language and suppose that T is a Turing machine which accepts L and always halts. Construct a new Turing machine T' which reverses the contents of the input tape and then invokes T. T' accepts the reverse of L and always halts.

   (b) True. Let L be a language and suppose that T is a Turing machine which accepts L. Construct a new Turing machine T' which reverses the contents of the input tape and then invokes T. T' accepts the reverse of L.

   (c) False. To check whether an input string w is in substrings(L), you need to search through all possible values for x and y. There's no bound on the lengths of x and y that might be required, so there's no obvious way to decide that the search will never succeed and reject w. (Proving this would require a reduction, but we specifically didn't ask for a proof.)

(d) True. Suppose that L is r.e. and T accepts L. We can construct an acceptor T' for substrings of (L) as follows. T' generates all pairs (x, y), ordered by $|x| + |y|$. In parallel with generating pairs, it simulates T on each already-generated pair, using dovetailing to do all the simulations in parallel. If any of the simulations halts in an accepting state, T does so too.

(e) True. Suppose that L is r.e. and T accepts L. We can construct an acceptor T' for lengths(L) as follows. Given an input 1^n, T' generates all strings of length n (a finite set). Using dovetailing, T' runs T on all these strings in parallel. If any of the computations halts in an accepting state, so does T'.

(f) True. Case 1: L is finite. Let m be the maximum length of any string in L. Then biglengths(L) is $\{1^n : n < m\}$, which is finite and therefore regular.

Case 2: L is infinite. There are only finitely many strings with length $\leq m$, for any m. Therefore, there can't be any bound on the lengths of strings in L. I.e. for every natural number m, L contains a string of length $> m$. So biglengths(L) is $\{1^n : n \in N\}$, which is regular.

2. Define an unrestricted grammar as one for which all productions are of the general form $\alpha \rightarrow \beta$, where α and β are elements of $(V \cup T)^*$. Thus, a production allows rewriting of arbitrary strings. The language remains the set of strings of terminals w that can be generated by starting with the start symbol S.

   Given an unrestricted grammar G, show that $L(G)$ is recursively enumerable by describing a generator that generates all and only the elements of $L(G)$.

SOLUTION

The generator T does a breadth-first search through all possible derivations. The production rules for G are hard-coded into the state transitions of T.

Specifically, T has four tapes: an output tape, a queue tape, a form tape, and a work tape. (The form and work tapes aren't really necessary, but they make T easier to describe.) The queue tape contains a list of sentential forms, separated by # symbols. T starts by writing $\#S\#$ onto the queue tape and # onto the output tape. It then starts its main loop.

In each iteration of the main loop, T does the following:

- If the queue tape is empty, halt. (This only happens when the language happens to be finite.)
- Copy the first sentential form γ from the queue tape onto the form tape, and erase it from the queue tape.

- Check to see if γ consists entirely of terminals. If so, copy $\gamma\#$ onto the output tape.
- Scan through all positions p in γ (on the form tape), left to right.
  - For each production rule $\alpha \rightarrow \beta$, where α matches the string starting at p, copy γ to the work tape, substituting β for α.
  - Write # onto the end of the queue tape, followed by the contents of the work tape.
  - Erase the work tape.
  - Check off position p on the form tape, so the scanning process can tell which position it is at.
- Erase the form tape.

3. (a) Show that if L is recursively enumerable, then there is a generator that generates each word in L a finite number of times, and each word in $\overline{L}$ (the complement of L) an infinite number of times.

   (b) Show that if L is recursively enumerable, then there is a generator that generates each word in L an infinite number of times, and each word in $\overline{L}$ a finite number of times.

SOLUTION

Suppose that T is an acceptor for L.

(a) We build a new generator T' as follows. T' generates all possible strings, in order of length, and runs T on all these strings in parallel, using dovetailing.

Specifically, T' has a string tape containing the list of strings w_i and a work tape containing a list of IDs separated by #. For each string w_i on the string tape, the work tape contains the current ID for the simulation of T on w_i.

The main loop for T' looks like:

- Generate the next string w_i.
- Add w_i to the end of the string tape and add its starting ID $q_0 w_i$ to the end of the work tape,
- Scan through the string and work tapes in parallel. For each string w_i whose ID is not an accepting configuration, do:
  - Copy $w_i\#$ onto T''s output tape.
  - Advance the computation of T by one step, updating the ID on the work tape.

When a string w_i is accepted by T, T' generates it only finitely many times because the simulation of T on w_i eventually gets into an accepting configuration, at which point T' stops printing w_i. If w_i is not accepted by T, the simulation of T on w_i will never get into an accepting configuration, so T' will keep printing w_i each time it goes through the main loop.

(b) The generator T'' for this part is similar to T', but its main loop looks like:

- Generate the next string w_i.
- Add w_i to the end of the string tape.
- Copy $w_i\#$ onto T'''s output tape. (If you regard zero as a finite number, this step can be omitted.)
- Add its starting ID $q_0 w_i$ to the end of the work tape,
- Scan through the string and work tapes. For each string w_i, examine the corresponding ID on the work tape and
  - If the ID for w_i is an accepting configuration, copy $w_i\#$ onto T'''s output tape.
  - Otherwise, advance the computation of T by one step, updating the ID on the work tape.

Strings not accepted by T are printed exactly once. When a string w_i is accepted by T, T' starts printing it every time through the main loop, so it gets printed infinitely many times.

4. Prove that $\{i : i \in L(M_i)\}$ is recursively enumerable

SOLUTION

Let's call this language L. We construct a Turing machine T accepting L as follows. T takes its input string w and copies it to create a new string $w\#w$. It feeds this new string to the universal Turing machine M_u. If M_u halts in an accepting state, so does T.

5. Prove that $\{i : L(M_i)$ contains the string "CS273ROCKS"$\}$ is undecidable.

SOLUTION

Let's call this language L and assume that there exists a Turing machine T that decides L. We will use T to construct a Turing machine S that decides the universal language. (Or you can use another problem known to be undecidable, e.g. the halting problem.)

First, for any Turing machine M and input string w, let's define M_w to be the Turing machine whose code does the following:

- Erases the contents of its input tape (regardless of what the input value was) and writes the fixed string w onto the input tape.
- Runs M (i.e. on the input w).

Notice that $L(M_w)$ is either Σ^* or $\emptyset$, depending on whether M accepts w.

Now, we construct the Turing machine S that solves the universal language. Given an input $M\#w$, S constructs the code for M_w and feeds it to T.

If $w \in L(M)$, then $L(M_w)$ is Σ^*. So, in particular, $L(M_w)$ contains "CS273ROCKS". So T will accept M_w. So S accepts $M\#w$.

If $w \notin L(M)$, then $L(M_w)$ is $\emptyset$. So, in particular, $L(M_w)$ does not contain "CS273ROCKS". So T will reject M_w. So S rejects $M\#w$.

But we've already proved that the universal language is undecidable. So S can't exist and, therefore, T can't exist either.

6. Is it decidable, given any number i, to determine whether or not TM M_i when started with input "CS273ROCKS" ever moves its head to the left? Briefly explain why or why not.

SOLUTION

This is decidable.

First, let's assume that our Turing machines always move left or right at each step, i.e. they don't have the "stay" operation.

To tell if M_i ever moves its head to the left when started with input "CS273ROCKS", we simulate M_i for $|Q| + 11$ steps. After that many steps, M_i has either moved its head to the left, halted, or is visibly in an infinite loop.

Specifically, we first watch the simulation as M_i reads rightward across the input string. It takes 10 steps for M_i to go off the end of the input string. Notice that it doesn't matter what M_i writes on the tape, because it can never read those symbols without moving left.

After this, M_i is off the righthand end of the input and, thus, always reading a blank symbol. So we watch the state of M_i. If M_i hasn't halted after $|Q| + 1$ steps, M_i has repeated some state q_x. That is, it has twice been in a configuration where it was in state q_x and reading a blank symbol. Since the state and the symbol being read are the only inputs to its transition function, M_i must be in an infinite loop.

This is sufficient for full credit. You did not need to worry about the stay operation, especially since we said "briefly explain" not "prove."

If we do allow M_i to use the "stay" operation, the argument is similar but more messy. At each tape position, M_i can write some or all of the symbols in Γ onto this tape cell

before it is forced to move or loop. So it can spend as much as $|\Gamma||Q|$ time at each tape position. Therefore, it can spend $10|\Gamma||Q|$ steps traversing the input string and $|\Gamma||Q|$ steps after the end of the input string, before it necessarily loops. Therefore, we have to watch the simulation for $1 + |Q||\Gamma| * 11$ steps before we are guaranteed that M_i halts, moves left, or has repeated a (state,symbol) combination and is thus in an infinite loop.

7. Is it decidable, given any number i, to determine whether or not TM M_i when started with input "CS273ROCKS" ever moves its head to the left three times in a row? Briefly explain why or why not.

SOLUTION

This is not decidable.

(The following is more detailed than you needed to be, to get full credit.)

Suppose that we had a Turing machine R that could determine whether any M_i moves its head to the left three times in a row on this input. We can use R to construct a Turing machine T that decides the language in problem 5.

(I picked the language from problem 5 because it already includes the silly string "CS273ROCKS", but you could have used any other language known to be undecidable e.g. the halting problem or the universal language.)

Specifically, given the code for a Turing machine M_i, define a new Turing machine M_i' as follows:

- M_i' simulates M_i, except that every time M_i does a leftward move, M_i' does the same move, followed by an extra right move and then an extra left move (both of which don't modify the tape contents).
- The accepting state of M_i is replaced by a new, non-final state q_x.
- M_i' contains new transitions from state q_x which move left three times in a row and then enter M_i''s accepting state.

Notice that M_i' moves left three times in a row on an input w if, and only if, M_i accepts w.

To accept the language from problem 5, our new Turing machine T takes the input code for a Turing machine M_i and rewrites it into code for M_i'. T then passes the code for M_i' to our hypothetical Turing machine R. T accepts exactly when R accepts.

M_i accepts "CS273ROCKS" exactly when M_i' moves left three times in a row, and this happens exactly when R accept M_i'. So T will accept exactly the language from problem 5.

But we know the language from problem 5 is undecidable, so T can't exist and therefore R can't exist.