Problem Set 5

Fall 09

Due: Thursday Dec 3 at 11:00 AM in class (i.e., Room 103 Talbot Lab)
Please follow the homework format guidelines posted on the class web page:
`http://www.cs.uiuc.edu/class/fa09/cs373/`

1. [**Points**: 15]

   (a) For each of the following PCP problems, either find a solution or prove that a solution does not exist.

   i.
   $$\left[\begin{array}{c}11\\101\end{array}\right]\left[\begin{array}{c}11\\11011\end{array}\right]\left[\begin{array}{c}110\\1\end{array}\right]$$

Solution:

$$\left[\begin{array}{c}110\\1\end{array}\right]\left[\begin{array}{c}11\\101\end{array}\right]\left[\begin{array}{c}110\\1\end{array}\right]\left[\begin{array}{c}11\\11011\end{array}\right]$$

   ii.
   $$\left[\begin{array}{c}10\\1\end{array}\right]\left[\begin{array}{c}10\\01\end{array}\right]\left[\begin{array}{c}01\\10\end{array}\right]\left[\begin{array}{c}0\\10\end{array}\right]$$

Solution:

This one has no solution: Since all the strings here have length at least one, the two string of the starting piece should start with the same character, therefore the only possibility for the starting piece is $(10, 1)$. Now the bottom string of the next domino should start with a zero, the only such piece is $(10, 01)$. Consider the sequence $(10, 1)(10, 01)$ we observe that again the next piece should start with a zero in its bottom string, that is we should play $(10, 01)$ again. Inspecting $(10, 1)(10, 01)(10, 01)$ we see that again the next piece should be $(10, 01)$ which means that the game has entered a loop and never finishes.

   iii.
   $$\left[\begin{array}{c}1110\\1\end{array}\right]\left[\begin{array}{c}1\\0111\end{array}\right]$$

Solution:

$$\left[\begin{array}{c}1110\\1\end{array}\right]\left[\begin{array}{c}1110\\1\end{array}\right]\left[\begin{array}{c}1110\\1\end{array}\right]\left[\begin{array}{c}1\\0111\end{array}\right]\left[\begin{array}{c}1\\0111\end{array}\right]\left[\begin{array}{c}1\\0111\end{array}\right]$$

   (b) Prove that PCP is undecidable even if we restrict its alphabet to two symbols, for example $\Sigma = \{0, 1\}$.

Solution:

Let's name this restricted version of PCP, PCP′. We reduce PCP to PCP′ (which proves that PCP′ is undecidable since PCP is undecidable). Let Σ be the alphabet of PCP. Our reduction function f replaces the i-th character in Σ with 0^i1. Note that the reduction is reversible, that is if $x, y \in \Sigma^*$ and $f(x) = f(y)$, then $x = y$. To reduce our instance of PCP to PCP′, we just apply function f to the strings of all domino pieces. Since f is reversible, some ordering of domino pieces is a solution for the reduced instance iff the same ordering is a solution for the original instance.

(c) Prove that PCP is decidable if we restrict its alphabet to one symbol, for example $\Sigma = \{0\}$.

Solution:

Note that the goal here is to pick some dominos such that the total number of zero's on the top of them equals the total number of zero's on the bottom of those selected dominos. If there is a domino with the same number of zeros on the top and bottom, then that domino itself is a solution, therefore we may assume that no domino has equal number of zeros on its sides. If all dominos have more zeros on top (bottom), then obviously any subset of dominos has more zeros on top (bottom) and therefore there is no solution. Finally assume that there is at least one domino (a, b) such that $a < b$ and on domino (c, d) such that $c > d$. Note that if we pick $b - a$ instances of (c, d) together with $c - d$ instances of (a, b), then we have a solution.

2. [**Points**: 15]

Consider the alphabet $\Sigma = \{a, b, c, d\}$. Let $R = \{(ab, \varepsilon), (ba, \varepsilon), (cd, \varepsilon), (dc, \varepsilon)\}$. Starting with a string $w \in \Sigma^*$, we can convert w to some other string w' by applying the following rule:
• If $(x, y) \in R$ or $(y, x) \in R$, and x is a substring of w, i.e. $w = w_1 x w_2$, then $w' = w_1 y w_2$.

The single player *Group Game* is started by some string w in Σ^*. At each round the player changes the string by applying the above rule. The game ends after a finite number of rounds. We say the pair of strings (w, w') is a *good pair* if, when the player starts the game with string w, he can finish it with string w'.

Given the pair (w, w') as input, is it decidable whether it is a good pair? Prove your answer.

Solution:

At each step of the game, we shorten the length of the string by deleting two adjacent a, b or c, d or we enlarge the length of the string by inserting two adjacent a, b or c, d. For any string w, let $f(w)$ be the string obtained from w by removing adjacent pairs of a, b and c, d iteratively (note that the order of removing these pairs is not important). By definition of f, no move of the game can shorten the length of $f(w)$. Moreover, by the rules of the game, both $(w, f(w))$ and $(f(w), w)$ are good pairs.

We prove that (w, w') is a good pair iff $f(w) = f(w')$ (and therefore the problem is decidable, since we can compute and compare $f(w), f(w')$ to figure out whether (w, w') is a good pair).

If $f(w) = f(w') = y$, then we know both (w, y) and (y, w') are good pairs and we can start the game by string w, followed by the game of the pair (w, y) to obtain y, followed by the game (y, w') to obtain w'. Therefore (w, w') is a good pair.

If (w, w') is a good pair then we know that we can start the game by w and reach w'. Now continue the game by removing adjacent a, b and c, d pairs from w' to obtain $f(w')$. So we have a game that starts from w and finishes with $f(w')$. Remove from this game all the steps that insert a pair to obtain a game from w to $f(w')$ that includes just removing adjacent a, b and c, d pairs. But the outcome of such a game ($f(w')$ here), by definition of f, must be $f(w)$. Therefore $f(w) = f(w')$.

3. [**Points**: 10]

   Let $A_1 \subset \{0,1\}^*$ and $A_2 \subset \{0,1\}^*$ be Turing-recognizable languages such that $A_1 \cup A_2 = \{0,1\}^*$ and $A_1 \cap A_2 \neq \emptyset$. Prove that $A_1 \leq_m (A_1 \cap A_2)$, where $A \leq_m B$ means that language A is mapping-reducible to language B.

Solution:

Let M_1 be a TM recognizing $A_1$and M_2 a TM recognizing A_2. Further, since we know that $A_1 \cap A_2 \neq \emptyset$, let y be some string in $A_1 \cap A_2$. We describe a reduction f such that $x \in A_1$ iff $f(x) \in A_1 \cap A_2$ as follows:

On input x
    Dovetail the computations of M_1 and M_2 on x, halting
        when either M_1 or M_2 halts and accepts x.
    if M_1 accepts x then output y
    else output x

Notice that since $A_1 \cup A_2 = \{0,1\}^*$, we know either M_1 or M_2 must accept x; so we will definitely halt in the dovetailing of the machines. Now, if when dovetailing M_1 and M_2 we find that $x \in A_1$ then $f(x) = y \in A_1 \cap A_2$. On the other hand, if we find that $x \in A_2$ then $f(x) = x \in A_1 \cap A_2$ if and only if $x \in A_1$. Thus, either way, $x \in A_1$ iff $f(x) \in A_1 \cap A_2$.

4. [**Points**: 15] Let A be Turing-recognizable, but not Turing-decidable. Consider $A' = \{0w \mid w \in A\} \cup \{1w \mid w \notin A\}$. For both A' and its complement, are they Turing-decidable or Turing-recognizable? Prove your point.

Solution:

We prove that neither A' nor its complement are recursively enumerable. First assume that A' is recursively enumerable, i.e. there exists a TM M that accepts A'. We use M to build a machine M_A that decides A and this contradicts the assumption that A is undecidable (not recursive). On input w, M_A simulates M on both $0w$ and $1w$ in parallel. If $w \in A$, then $0w \in A'$ and therefore M

accepts $0w$ in a finite number of steps. If on the other hand $w \notin A$, then $1w \in A'$ and therefore M accepts $1w$ in a finite number of steps. Since each string w either belongs to A or does not belong to A, on input w, exactly one of $0w$ or $1w$ is in A' and therefore, by looking at which one is accepted by M, M_A can decide whether $w \in A$ or not. The argument for the complement of A' being not r.e. is similar.

5. [**Points**: 10] Determine whether each language is decidable and prove your answer without using Rice's theorem.

   (a) $A = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Solution:

Since accepting no string is not a trivial property, Rice's Theorem hints that A is undecidable. Therefore, we are looking for a reduction as proof. Let's take A_{TM} ans show $A_{TM} \leq_m A$. For $\langle M, w \rangle$, we build M' such that

M': On input x
    if M accepts w, reject
    otherwise, accept.

We see that

(1) If $\langle M, w \rangle \in A_{TM}$, then $L(M') = \{\}, \langle M' \rangle \in A$.
(2) If $\langle M, w \rangle \notin A_{TM}$, then $L(M') = \Sigma^*, \langle M' \rangle \notin A$.

   (b) $A = \{\langle M \rangle \mid$ given a blank input$M$ uses at most 20 cells of tape $\}$

Solution:

If M uses at most 20 cells then the number of distinct configurations in which M can be is finite and is bounded by $20|\Gamma|^{20}|Q|$, because there are $|\Gamma|^{20}$ possible combinations of symbols on 20 cells, M can be in at most $|Q|$ states and the tape head can be in at most 20 different positions.

We can decide if M started on blank input ever uses more than 20 cells as follows: simulate M for $20|\Gamma|^{20}|Q| + 1$ steps. If it used more than 20 cells, reject, otherwise accept.

If M has used at most 20 cells in that many steps, then, by the pigeonhole principle, some configuration of M must have been repeated, which means that M is in an (infinite) loop, and will never use more than 20 cells.

   (c) $A = \{\langle M \rangle \mid L(M)$ is finite and Turing-decidable $\}$

Solution:

A is undecidable with reduction being the same as 5(a).

(d) $A = \{\langle M \rangle \mid L(M)$ contains some string that is a palindrome $\}$

Solution:

We will prove that A is undecidable by reducing A_{TM} to it. Suppose A is decidable and there's a TM M_A that decides it.

M': On input $\langle M, w \rangle$

Construct the coding $\langle T_{M,w} \rangle$ of $T_{M,w}$, such that $\langle T_{M,w} \rangle$ is in A iff $\langle M, w \rangle$ is in A_{TM}
Run M_A on input $\langle T_{M,w} \rangle$. Accept if M_A accepts and reject if it rejects.
If M_A accepts then $\langle T_{M,w} \rangle \in A$ which implies $\langle M, w \rangle \in A_{TM}$,
otherwise $\langle T_{M,w} \rangle \notin A$ which implies $\langle M, w \rangle \notin A_{TM}$

Now we describe how to construct $T_{M,w}$ from $\langle M, w \rangle$. $T_{M,w}$ works as follows: on input x, $T_{M,w}$ begins by running M on w. If M accepts then $T_{M,w}$ passes its input x to a TM M_1 that accepts exactly the set of palindromes. If M does not accept w then $T_{M,w}$ doesn't accept any string, hence does not accept any palindrome, so $\langle T_{M,w} \rangle \notin A$; on the other hand if M accepts w then $L(T_{M,w})$ is exactly the set of palindromes, and thus $\langle T_{M,w} \rangle \in A$. Therefore $\langle T_{M,w} \rangle \in A$ iff $\langle M, w \rangle \in A_T M$, as desired.

6. [**Points**: 15] For each of the following either prove that the language is undecidable or present an algorithm to solve the problem, that halts on all inputs. M_i is the i-th Turing Machine in some fixed enumeration of all Turing Machines.

(a) Given i and w, decide whether M_i only writes to the cells initially occupied by w

Solution:

We will use a TM with a tape that is infinite in both directions to simplify a proof. The idea is similar to problem 5b. If M only writes on the cells that w occupies, then the number of distinct tape configurations will be $|\Gamma|^w$. But M may be in any of $|Q|$ states and may move its head any number of cells off to either side of w, so the number of configurations of M appears to be unbounded. However, if M spends more than $|Q| + 1$ consecutive steps to the left (right) of w, without writing, then it must be in a loop. Thus, if M never writes to the left or right of w, and M doesn't get into loop, then the head of M must remain on one of the cells within distance $|Q|$ of w on either side. Thus, the number of distinct configurations is bounded by $(2|Q| + w)|\Gamma|^w|Q|$. The rest of the proof is similar to problem 5b.

(b) Given i and w, decide whether M_i ever moves its head to the left on input w

Solution:

The idea is similar to 6a (and 5b). First, consider how long M_i can remain on the same cell without moving its head: since configuration depend only on current state and the symbol being scanned, M_i cannot remain more than $|Q| \times |\Gamma|$ steps on the same cell, without entering the loop. M_i can also make no more than $|w| + |Q|$ moves to the right (see problem 6a) without

entering the loop. Therefore, the maximum number of steps M_i can go without looping is $|Q| \times |\Gamma| \times (|w| + |Q|)$. The rest of the proof is similar to 6a and 5b.

(c) Given i and j, decide whether $L(M_i) = L(M_j)^c$, where A^c is the complement of A

Solution:

Suppose the problem was decidable, that is there were a TM C, which given an input $i\#j$, would determine whether or not $L(M_i) = L(M_j)^c$. Then we would be able to devise a TM E to decide $E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$, which is known to be undecidable (see problem 5a).

First, we construct a TM L that accepts Σ^* and find its number j_0. And now we construct a E to be:

E = "On input i:
Append $\#j_0$ to the input
Run C on the result. Accept if C accepts, and reject otherwise"

If C accepts $i\#j_0$ then $L(M_i) = L(M_{j_0})^c = (\Sigma^*)^c = \emptyset$ and E accepts, otherwise $L(M_i) \neq L(M_{j_0})^c \neq \emptyset$ and E rejects. Thus we have an algorithm for deciding E_{TM}, contradiction.

7. [**Points**: 20] Consider the following Register Machine model which has:

   (a) A finite number of (arithmetic) registers: $R0, R1, R2, \ldots$

   (b) An infinite number of available memory locations (each location can be accessed by its index)

   (c) An instruction set (defined below)

   The contents of each register and memory location is a non-negative integer not bigger than K, for some fixed integer K. The instruction set is:

   (a) $ADD\ Rx, Ry, Rz$ – add contents of reg Rx and reg Ry, putting result into Rz

   (b) $LOADC\ Rx, NUM$ – place constant NUM into reg Rx

   (c) $LOAD\ Rx, M$ – put contents of memory location M into Rx

   (d) $LOADI\ Rx, M$ – (load indirect) put $value(value(M))$ into Rx

   (e) $STORE\ Rx, M$ – store contents of Rx into location M

   (f) $JUMP\ Rx, M$ – if value of Rx is 0 then jump to instruction at location M, else continue normal execution

   (g) $HALT$ – halt the execution

   The machine starts with program instructions written in contiguous block of memory, starting at location 0; a string w written in a block of memory at location M; the value of register $R0$ is M, all other registers are set to 0.
   The machine begins by executing instruction at memory location 0. After executing current instruction and modifying the contents of registers and memory, the machine proceeds to the next

instruction. The next instruction is the adjacent instruction in the next memory location (with bigger index), or the instruction located at the memory address specified in the second parameter of $JUMP$ instruction, if $JUMP$ was executed.
Note that program is located in memory and thus machine can even alter itself during execution.
The Register Machine M accepts the string w, if it halts with a non-zero value in register $R0$.
Prove that this machine is equivalent to a Turing machine. That is, a language is Turing recognizable if and only if there exists a Register Machine, as defined above, which can recognize it.

Solution:

Simulating Register Machine (RM) on TM. We will simulate RM using TM with multiple tapes. Our TM maintains:

(a) Register tape. Stores values of registers as a list, separated by #: $\#reg.num., reg.contents\#....$ Both $reg.num$ and $reg.contents$ are in unary alphabet, i.e. sequences of 1s.

(b) Memory tape. Stores a memory of RM in a form similar to register tape: as a list of memory location and a value stored at that location. If the contents of the memory location is a number (data), we store it as $\#mem.loc., D, mem.contents\#$, and if the contents is an instruction we store it as $\#mem.loc., X, param1, param2\#$ where X is: A for ADD, C for $LOADC$, I for $LOADI$, L for $LOAD$, S for $STORE$ and J for $JUMP$. We store $HALT$ instruction as $\#mem.loc., X\#$.

(c) Current instruction location tape (PC-tape, for Program Counter). It stores a location of the instruction to be executed, initially contains 0.

(d) Next instruction tape, used to calculate the next instruction to execute after the current one is executed

(e) Work tape for carrying computations

TM begins with blank register and work tape, 0 written on PC-tape, program encoded on memory tape as described above.
For each RM step, TM:

(a) Copies the value of PC-tape to the next instruction tape

(b) Increases the value on the next instruction tape by 1 (by simply appending 1 to the end)

(c) Reads PC-tape and uses its value to search memory tape for the instruction

(d) Executes the instruction, changing register, memory, work and next instruction tapes as needed

(e) Copies the value of the next instruction tape to the PC tape

To update the value of register Rx with value val written on the work tape, our TM:

(a) Finds the record in the list on the register tape

(b) It then removes this record, shifting all others to the left to close the gap

(c) Appends the new record $\#x, val\#$ to the end of the list, copying val from the work tape

Updating the value of memory location is similar. Each instruction is executed as follows:

(a) $ADDRx, Ry, Rz$: TM searches register tape for register x and copies it value on work tape (overwriting the contents), then it appends value of register y to tape. The result is a sum of values. It then updates the register z with the value sum, which is the value written on the work tape

(b) $LOADC\ Rx, NUM$: TM copies NUM to the work tape and then updates the value of register x with it

(c) $LOAD\ Rx, M$: TM copies the value of memory location M to the work tape, then update register x with it

(d) $LOADI\ Rx, M$: TM copies the value v of memory location M to the work tape, then it copies the value of memory location v to then work tape, overwriting the previous contents. It then updates the register x with the value of the work tape

(e) $STORE\ Rx, M$: TM copies the value of register x to the work tape, it then updates the value of memory location M with it

(f) $JUMP\ Rx, M$: Finds the value of register x on the register tape. If its value is not 0 then it proceeds to the next execution cycle. Otherwise, copies value at location M on memory tape to the next instruction tape

(g) $HALT$: TM searches for value of register 0 on register tape and accepts if it's non-zero

Simulating TM using RM is impossible, since we can only address a finite amount of memory locations, because the value of registers is bound by K.