CS331: Homework #6

Due on March 14, 2013 at 11:59pm

Professor Zhang 9:00am

Josh Davis

Problem 1

Let $\Sigma = \{0, 1\}$. Consider the following language over Σ,

$$L = \{w_1\#w_2 : w_1w_2, \in \{0,1\}^* \text{ and } w_1 < w_2\}$$

Design a Turing machine using pseudocode that recognizes L.

$M =$" On input string w:

1. Start on the left side of the word, move all the way to the right of the input, move to stage $loop_>$
2. $loop_>$ Move to the first character in the second word that isn't an x. If on 0, mark with x and move to stage $cmp_{<0}$. If on 1, mark with x and move to $cmp_{>1}$. If on ␣, move to q_{reject}. If on #, move to stage **check**.
3. $loop_<$ If on 0, mark with x and move to stage $cmp_{>0}$. If on 1, mark with x and move to $cmp_{<1}$. If on ␣, move to q_{accept}
4. $cmp_{<0}$ Move to the first character in the first word that isn't an x. If 1, mark with x and move to $loop_>$. If 0, mark with x and move to $loop_<$. If ␣, move to $loop_<$.
5. $cmp_{<1}$ Move to the first character in the first word that isn't an x. If 1, mark with x and move to $loop_<$. If 0, mark with x and move to $loop_<$. If ␣, move to $loop_<$.
6. $cmp_{>0}$ Move to the first character in the first word that isn't an x. If 1, mark with x and move to $loop_>$. If 0, mark with x and move to $loop_>$. If ␣, move to $loop_>$.
7. $cmp_{>1}$ Move to the first character in the first word that isn't an x. If 1, mark with x and move to $loop_>$. If 0, mark with x and move to $loop_<$. If ␣, move to $loop_>$.
8. **check** Move to first character in the first word that isn't an x. If 1, move to q_{reject}. If 0, move to stage **check**. If ␣, move to q_{accept}."

This is valid because we can split up the input into four cases which each have two sub-cases as follows:
Case One Input is $0w_1\#0w_2$:

1. If $w_1 < w_2$, then $w_1 < w_2$
2. If $w_1 > w_2$, then $w_1 > w_2$

Case Two Input is $0w_1\#1w_2$:

1. If $w_1 < w_2$, then $w_1 < w_2$
2. If $w_1 > w_2$, then $w_1 < w_2$

Case Three Input is $1w_1\#0w_2$:

1. If $w_1 < w_2$, then $w_1 > w_2$
2. If $w_1 > w_2$, then $w_1 > w_2$

Case Four Input is $1w_1\#1w_2$:

1. If $w_1 < w_2$, then $w_1 < w_2$
2. If $w_1 > w_2$, then $w_1 > w_2$

Thus when the word ends, we just accept if we still satisfy $w_1 < w_2$.

Note When a given word runs out of characters, we just treat it if it had 0's appended to the beginning such that the length of the words are equal.

Problem 2

Prove that 2-dimensional Turing machines are no more powerful than standard Turing machines.

Proof. We will use a proof by construction to construct a new 2D TM, N such that it can be simulated by a normal TM, M. Let N and M be defined as follows:

$$N = (Q, \Sigma, \Gamma, \delta, q, q_{accept}, q_{reject})M = (Q_M, \Sigma_M, \Gamma_M, \delta_M, q_M, m_{accept}, m_{reject})$$

We will use a counting argument to show that the tape of N is identical to the tape of M in size.

Much like the way we show that the number of rational numbers is countably infinite by making a 1-to-1 correspondence to $\mathbb{N}$. We can do the exact same thing as this.

Achieving this will show that every square on 2D tape corresponds to tape on the normal TM.

	0	1	2	3	4
0	(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)
1	(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)
2	(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)
3	(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)
4	(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)

By moving in the order $(0, 0) \rightarrow (0, 1) \rightarrow (1, 0) \rightarrow (0, 2) \rightarrow \ldots$, the number of squares can be assigned with a 1-to-1 correspondence to $\mathbb{N}$. Thus this shows that a normal TM tape of M can simulate a 2D tape of N

Given this construction, it can easily be seen that now moving up, down, right, and left are possible by just moving to the corresponding square in the sequence.

Considering that is the hardest part, the rest of N can be defined as follows:

1. $Q = Q_M$ The states are the same
2. $\Sigma = \Sigma_M$ The alphabet is the same
3. $\Gamma = \Gamma_M$ The stack alphabet is the same
4. $q = q_M$ The start states are the same
5. $q_{accept} = m_{accept}$ The accept states are the same
6. $q_{reject} = m_{reject}$ The reject states are the same

Thus we have shown that a 2D TM, N is no more powerful than a normal TM, M by constructing N out of M. Our proof is complete. $\square$

Problem 3

Prove that the class of Turing-recognizable languages are closed under certain operations.

Part One Union: $L = L_1 \cup L_2$

Proof. Given two Turing machines, M_1 and M_2 that recognize the languages L_1 and L_2, respectively, we will construct a new Turing machine, M, that recognizes the union of the two languages.

Let M_1 and M_2 be defined as follows:

$$M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_{M10}, q_{M1accept}, q_{M1reject})$$
$$M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, q_{M20}, q_{M2accept}, q_{M2reject})$$

We can then construct M as follows:

1. $Q = Q_1 \cup Q_2$, the state machine
2. $\Sigma = \Sigma_1 \cup \Sigma_2$, the alphabet
3. $\Gamma = \Gamma_1 \cup \Gamma_2$, the stack alphabet
4. q_{accept} A new accept state for when both machines accept.
5. q_0 A new start state.
6. $M =$" On input string w:
   (a) Run the input w over the two TMs in parallel.
   (b) If either M_1 or M_2 accept the input, move to the accept state if both machines reject it, move to the reject state.

This works because we can run the TMs in parallel and accept if either of them accepts. This shows that the language will accept words that are in either the language of the first TM or the second one. □

Problem 3 continued on next page. . .

Part Two Intersection: $L = L_1 \cap L_2$

Proof. Given two Turing machines, M_1 and M_2 that recognize the languages L_1 and L_2, respectively, we will construct a new Turing machine, M, that recognizes the intersection of the two languages.

Let M_1 and M_2 be defined as follows:

$$M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_{M10}, q_{M1accept}, q_{M1reject})$$
$$M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, q_{M20}, q_{M2accept}, q_{M2reject})$$

We can then construct M as follows:

1. $Q = Q_1 \cup Q_2$, the state machine
2. $\Sigma = \Sigma_1 \cup \Sigma_2$, the alphabet
3. $\Gamma = \Gamma_1 \cup \Gamma_2$, the stack alphabet
4. q_{accept} A new accept state for when both machines accept.
5. q_0 A new start state.
6. $M =$" On input string w:
   (a) Run the input w over the two TMs in parallel.
   (b) If both M_1 and M_2 accept the input, move to the accept state if either machine rejects it, move to the reject state.

This works because we can just run the two TMs in parallel and only accept when they both accept. Thus a word will only appear in our new language when both TMs accept it and don't reject it. □

Problem 3 continued on next page. . .

Part Three Concatentation: $L = L_1 * L_2$

Proof. Given two Turing machines, M_1 and M_2 that recognize the languages L_1 and L_2, respectively, we will construct a new Turing machine, M, that recognizes the concatentation of the two languages.

Let M_1 and M_2 be defined as follows:

$$M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_{M10}, q_{M1accept}, q_{M1reject})$$
$$M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, q_{M20}, q_{M2accept}, q_{M2reject})$$

We can then construct M as follows:

1. $Q = Q_1 \cup Q_2$, the state machine
2. $\Sigma = \Sigma_1 \cup \Sigma_2$, the alphabet
3. $\Gamma = \Gamma_1 \cup \Gamma_2$, the stack alphabet
4. q_{accept} A new accept state for when both machines accept.
5. q_0 A new start state.
6. $M =$" On input string w:
   (a) Split the input w into two parts, $w = xy$. We can do this nondeterministically.
   (b) Then run M_1 over the first part, x, if it accepts it go to the next stage, if it rejects it, move to the reject state
   (c) Next, run M_2 over the second part, y. If it accepts it, move to the accept state, if it rejects it, move to the reject state.

This works because we can use the power of Nondeterministic TMs to split the string and run each TM sequentially. This is valid because it is shown in the book that NTMs are no more powerful than normal TMs. In splitting the string, we can assure that every possibility will be checked and thus it will show that the first language is the concatenation of the second one. □

Problem 4

Prove the following languages aren't Turing decidable.

Part One $L_B = \{\langle M\rangle : M$ will write "V" somewhere on the tape $\}$.

Proof. We will use a proof by contradiction to show that L_B is not Turing-decidable.

Assume L_B is Turing-decidable. That means there is a decider, D, that decides L_B.

Now we will use D to decide A_{TM} by constructing a TM M that decides A_{TM}:

```
M =" On input ⟨M′, w⟩:
  1. Construct new TM, $M'_w$ = "On no input:
      (a) Run $M'$ on $w$
      (b) If $M'$ accepts, write "V" on the tape."
  2. Run $D$ on $M'_w$
  3. If $D$ accepts, accept, if $D$ rejects, reject."
```

This is a contradiction because if we can solve L_B, that means we can solve A_{TM}. Since A_{TM} is undecidable that must mean that L_B is also undecidable. Thus our proof is complete. □

This works because by constructing a new TM, we encapsulate w and only write "V" on the tape if it accepts. We then run our decider D on this newly constructed TM, if it accepts, then we know we have determined if the input TM M' accepts w, thus it decides A_{TM}. The converse is also true and can be trivially justified as well.

Part Two $L_U = \{\langle M\rangle : M$ halts on all words except **one** $\}$.

Proof. We will use a proof by contradiction to show that L_U is not Turing-decidable.

Assume L_U is Turing-decidable. That means there is a decider, D, that decides L_U.

Now we will use D to decide $HALT_{TM}$ by constructing a TM M that decides $HALT_{TM}$:

```
M =" On input ⟨M′, w⟩:
  1. Construct new TM, $M'_w$ = "On no input:
      (a) If $w$ equals some string, say "V", accept
      (b) Run $M'$ on $w$
      (c) If $M'$ accepts, halt."
  2. Run $D$ on $M'_w$
  3. If $D$ accepts, accept
  4. If $D$ rejects, reject."
```

This is a contradiction because if we can solve L_U, that means we can solve $HALT_{TM}$. Since $HALT_{TM}$ is undecidable that must mean that L_U is also undecidable. Thus our proof is complete. □

Problem 5

Prove or disprove that the language, L_s is Turing decidable.

Proof. We will show that the language L_s is in fact Turing decidable.

To show that this is true, we will use the idea of Configuration History to prove it.

Given a TM, $M = (Q, \Sigma, \Gamma, \delta, q, q_{accept}, q_{reject})$, we can keep track of the configurations. This can be represented as follows:

$$c_0 \rightarrow c_1 \rightarrow \ldots \rightarrow c_{n-1} \rightarrow c_n$$

c_0 will be the starting state while c_n will be the accepting state (if it halts).

Since we have this configuration, we can deterministically represent all the possible states that the configurations can be in. The total number of possibilities is $n^k(k)(m)$ where $k = |\Gamma|$, $m = |Q|$.

Thus each configuration step we move along, we can compare it to the last. If the configuration is in the same state, it must be in a loop and we reject. If it isn't in the same state, continue.

Doing this we can turn the language, L_s into a decidable language by bounding the number of movements the head can make. □