CS516 Homework #1 Solutions

Prepared by Damodar Goud Nagapuram

February 4, 2011

Remark by Dang: I only post the solutions to Problems 1, 2, 3, 4, 6. These solutions are selected from Damodar's submitted homework. I will talk about Problems 5 and 7 in class. Problems 1.2 and 3 are adapted from Sisper, and Hopcroft and Ullman. I don't remember where the problem 2 is from. Problem 5 is from an old paper of Ibarra.

Solution #1.1

The smallest class of languages accepted by a 10-change TM is regular language.To establish this fact we need to prove that the 10-change TM accepts any regular language and a FA(a regular language) accepts the language of 10-change TM.

- A 10-change TM accepts regular languages. This is trivial because 10-change TM has all the moves of FA besides being able to make 10 changes .

- We need to simulate a 10-change TM,M by a FA. Since the TM is free to move in any direction, I choose a Non-deterministic 2-way FA, $M\prime$, to simulate it. Now to simulate 10-change TM, I need to remember the positions where the M makes changes to its tape contents. But, I can not remember position in a FA. So, we can not directly simulate the TM M by 2FA $M\prime$. hence we need to indirectly simulate M by a 2FA.

  I'll prove that the language accepted by M is regular. Let us consider a string accepted by 10-change TM. I'll simulate the acceptance of the same string by $M\prime$. Since $M\prime$ is 2-way all of the possible moves(L,R,S) of 10-change TM are simulated by it. When 10-change TM does not change cell content this is exactly same as the $M\prime$. But what to do when M changes its tape content?

  My machine $M\prime$ is non-deterministic and can guess an input. When M reads a but does not overwrite $M\prime$ exactly does the same. My machine is powerful enough to guess any input symbol, and it has input (c,d) corresponding to the position where M overwrites the symbol c by d. Now what does $M\prime$ do when it encounters this symbol? Will it read c or will it read d?

  I use my finite memory to break this tie. Whenever M makes the first change(C1), say c to d, $M\prime$ encounters (c,d). At that moment, my machine $M\prime$ stores C1 in its control unit. So when the symbol (c,d), corresponding to C1, is encountered, $M\prime$ checks its control memory if the change has been made. If the change is recorded in its control

memory then it reads d otherwise it reads c. This can be extended to finite number of changes because I can store only finite changes in my control memory.

When M accepts $M\prime$ also accepts. But, we have to note here that $M\prime$ has tape content different from M. But, it is fact that even if change/drop a symbol from the alphabet of a regular language, the language is still regular.Thus the language accepted by the 10-change TM is a regular language.

Solution #1.2

The problem is to prove that a *10-change-per-cell TM* can simulate any arbitrary TM. In *10-change-per-cell* TM there at most ten changes occuring per cell. I would like to use the concept of copying to simulate an arbitrary TM by *10-change-per-cell* TM.

The new TM $M\prime$ can make at most 10 changes unlike a standard TM M which can make any number of changes per cell. So, I just need to explain how $M\prime$ can simulate the changes made by M. In fact, a 2-change-per-cell TM is powerful enough to simulate M. Whenever M makes a move without overwriting the contents of cell, $M\prime$ exactly does the same. When M overwrites a symbol,say c to d, $M\prime$ can also overwrite the symbol. This constitutes a change and there is a problem that a cell may be overwritten more than 2 times. To avoid this, whenever a change is made, $M\prime$ does not immediately make the necessary change. Instead it marks the position to be changed and copies the entire string to the right. Note that copying(shifting to right the whole string by length of string done by marking current symbol by blank,going right to the first blank and placing the symbol erased) requires at most 1 change. While copying the symbols to right, $M\prime$ checks when erasing each symbol if it is marked i.e. symbol to be overwritten. When it encounters the marked symbol(c), it writes the new symbol(d) instead of old symbol(c) in the new copy. My machine $M\prime$ passes through a sequence of configurations as the original TM's configurations, by above instructions. $M\prime$ makes at most 2 changes per cell,one when overwriting the blank symbol by alphabet, and the other when erasing the symbol. When M accepts $M\prime$ accepts.

Can I do that with 1-change-per-cell TM? (Not required by question,but I am trying to do this).

If I constrain the string between delimiters and make $M\prime$ to read only the symbols between the current delimiters, $M\prime$ makes only one change. Note that the content of a cell is changed only once during initial copying and it is left unchanged to blank.

Solution #2

The language accepted by a 1-turn Turing machine is regular. But,according to the instruction in class, first I'll prove the language to be CFL and then prove that the language is regular.

Proving that language accepted by 1-turn TM is CFL

I need to show that any 1-turn TM is accepted by a PDA. To do this,we need to simulate a 1-turn TM,M by a PDA $M\prime$. As M keeps moving to right reading the input string, $M\prime$ pushes symbols onto the stack moving right

and making necessary state transmissions. When M overwrites a symbol and stays(S) in the current position, $M\prime$ simply pops the top most symbol and pushes the new symbol. When the TM M starts moving left(makes first L move), $M\prime$ simply pops and reads the symbols from top of the stack. During its leftward motion if the TM M overwrites a symbol, $M\prime$ need not consider it because it does not require to read this symbol anymore. Note that S can be easily simulated a by a pop followed by a push operation. When M accepts $M\prime$ accepts. The other way i.e. proving that any language accepted by a PDA is accepted by a 1-turn TM is quite simple if we consider acceptance of PDA by final state.

Proving that the language accepted by a 1-turn TM is regular

It is easy to see that 1-turn TM can simulate any FA. So, I will give the simulation of 1-turn TM,M by a FA $M\prime$. This machine is again complex like the one in problem in 1.1. $M\prime$ is a 2-way NFA. Again the problem is to remember the changes made by the TM M during its rightward moment (similar to problem 1.1). I use the same technique explained in the solution #1.1 here. There I need to register the change in the finite memory because multiple turns are possible there. Unlike that here we have at most one turn allowed by M. So $M\prime$ has input tape with compound symbols, e.g,(c,d) when M overwrites c by d. But *what if M overwrites a symbol more than once the contents of a single cell during its rightward moves?*, $M\prime$ just makes sure that the second component of symbol is the final symbol written by M. Once TM M starts moving left, $M\prime$ exactly simulates M for unchanged symbols. For overwritten symbols, $M\prime$ encounters compound symbols when it reads the second part of that symbol which is same as the symbol read by M. Proceeding in this way, N' accepts when M accepts.

Solution #3

The problem is to prove that a jump-TM can simulate any TM. The jump TM is allowed to make a J move to the beginning of the input tape instead of L move. I'll construct jump-TM $M\prime$ to simulate any arbitrary TM M.

I'll use the concept of copying to prove the result(as in problem 1.2). All the moves of M are simulated by $M\prime$ except L (left) move. So, I need to simulate the L move of M by $M\prime$. When M makes a left move, say from (n+1)th position to nth position, then $M\prime$ marks the (n+1)th position on its tape. $M\prime$ moves to the extreme right of input string and places a delimiter symbol after the last occupied position. It jumps to the left i.e. makes J move. Then it moves until it encounters the first non-blank symbol. It stores this symbol in its memory and overwrites the symbol by a blank symbol. Then it reads the next symbol and checks if it is marked.

- If the next symbol is not marked, $M\prime$ moves to the right and replaces first blank symbol by the content of memory. It again makes a J move and repeats the moves explained above.
- If the next symbol is marked, $M\prime$ moves to the right and replaces first blank symbol by the content of memory. It now marks this position and makes a J move. Note here that this mark is the position in the input string where M(original TM) has its head reading the input tape.

The 1-turn TM $M\prime$ repeats the above instructions until the end of the string

(till it encounters the delimiter). After copying the final character of the string,it starts moving right until the mark is found. When it encounters the marked symbol, it determines that this is the position to read the input character from to simulate M. $M\prime$ accepts when M accepts.

Solution #4

I need to simulate two counters(C1 and C2) by a queue Q. To do this I need to simulate increment, decrement, test zero operations on C1 and C2 by operations of Q viz. enqueuing, dequeuing and isEmpty operations.
The queue Q has a marker ++ to distinguish the two counters C1 and C2. C1 is represented by the part of the queue to the left of marker ++. The other part represents C2. The number of symbols to the left/right of marker are the count of C1/C2. We can enqueue at the left end of Q and dequeue at the right end of Q.

Operations on C1

- *Increment operation, C1++* This can be simulated by enqueuing a symbol at the left end of Q. The length of the left part increases by one resulting in the increment of C1.

- *Decrement operation, C1–* Note that the decrement operation is valid only when counter¿0. I enqueue a marker ** at the left of queue. Now, I dequeue symbols from right and enqueue them at the left end of the queue, until I encounter the old ++ marker. Now, I remove the first symbol after the ++ marker from the queue and drop it. Again I continue dequeuing symbols one after the other and enqueuing them at the left end until the new ** marker is encountered. I drop this marker from the queue. Now the left part of queue(C1) has one symbol less than before equivalent to C1–.

- *Iszero operation, C1==0?* This is checking if the length of the left part of the queue is zero i.e. checking if there are no symbols to the left of the marker ++.

Operations on C2

- *Increment operation, C2++* I enqueue a maker ** at the left end. Enqueue a new symbol next to that marker. Now, I dequeue symbols from right and add to the left of Q, one by one, until I encounter the marker **. At this point, I just dequeue it and stop. The right part of the queue now has one symbol more than before. This is equivalent to C2++ operation.

- *Decrement operation, C2–* This is equivalent to dequeuing a symbol from the right of the queue Q. The number of symbols to the right of marker reduces by 1 resulting in the operation C2–.

- *Iszero operation, C2==0?* Similar to the counter C1, but checks if there are no symbols to the right of the ++ marker.

Solution#5

Talk in class.

Solution#6
The problem is to prove that emptiness of a language with given instructions is decidable. The instructions given are:

```
x := x + 1
x := x - 1
x == 0 ?
x == B ?
0<= x <= B always satisfied.
```

It is noticed that B is not changed during any computation. So, B is fixed for a given computation. This gave me an idea of simulating the instructions by LBA. The length of LBA constrained by delimiters gives the B. And for given LBA , it is fixed. Other instructions are easily simulated. But, the problem with LBA is it's emptiness is undecidable.
The above idea gave me hint that B need to be fixed. In fact, a NFA can simulate the given graph(program). For any given computation B is fixed. Why can't I have that many symbols on the tape of my FA? Yes. I assume that the given tape has length B. Two delimiters at both ends indicate the tape length.
How does the computations work and when should I accept?
The problem is to determine whether there exists non-negative initial values so that the graph is terminating. In terms of my machine, it is the same as asking" Can you start from some position on the tape of length B and finally arrive at an accepting state?". This position gives the starting value of x. When the graph walks thru instruction x = x+1, my head is moved right. (Similarly for x = x-1). What about x==0? and x == B?. I check if my head is at the beginning or end marker to answer those questions. Can I decrement/increment from x=0/x=B. The given program crashes when there is no allowed instruction but to increment x more than B or decrement less than zero. My machine also crashes if it needs to fall off the left/right end of the tape. When the given graph reaches a final node, my machine accepts.
Few things to be noted here are the initial position of FA and length of tape. For given B of the program, my machine has a tape of length B. And for any non-negative value of x, I can start from corresponding position on the tape. If it really needs to start from left-end marker, I can do that by just having my head start from left-end marker and move to right till the position x before execution of the first instruction. And, of course,I can accept in the right end of tape by moving to the right end of the tape after reaching the final state. In fact, this is the behavior of any ordinary FA which starts at left end and terminates at the right end. And we know that the emptiness of a FA is decidable. Hence there is a way to check for the termination of the graph G.
Solution#7
talk in class.