Hw 5

Theory of languages and machines

Dr. movaghar

Sara Azarnoush

98170668

Contents

1

1.1

a) L = {a^iba^j| $0 \leq i < j$}

TM M = on input string ω:

1. Determine if input is not a member of a+ba+ -> reject. return to the left end tape.
2. go right untill b. continue until an unmarked off a -> seen it. If empty -> reject, else marked the founded a. return to the left end tape.
3. Go right untill first uunmarked symbol occurs. If a-> go to previous step. If b-> continue scanning until an unmarked a. if empty-> reject, else -> accept.

b) L = {$a^nb^na^nb^n$| $n \neq 0$}

TM M = on input string ω:

1. determine if input is a member of a+b+a+b+ and if not -> reject. return to the left end tape.
2. Marked a and scan untill unmarked b. If empty-> reject, else -> marked b. Continue until an unmarked a -> marked it. If no unmarked a after b -> reject, else -> marked a. Continue until unmarked b -> marked. If no unmarked b after a -> reject, else -> marked b. return to the left end tape.
3. Continue to the right untill first unmarked symbol. If a-> go previous step. If not -> reject. If no unmarked symbol-> accept.

1.2

a) $f(x) = x^2$

TM M = on input string ω:

1. determine if input is a member of 1+ and if not -> reject. Return to left end tape.
2. change 1 to 1′ and move to first empty position. write 0 and return to left end tape.
3. Continue to right untill first 1 -> go previous step. If no 1 -> return t left end tape.
4. change 1’ to 1″ and move untill first 0. change 0 to 0′ and write 1 at the end of the tape. return to the furthest 0 and go to 2nd step. If no 0 -> go next step.
5. from end of the tape go to furthest 0′ and change to 0 then go to the next symbol.
6. If symbol under the head is 0′ -> change to 0 then go to the next symbol.
7. go to furthest 1′ and go to 4th step. If no 1 ′ -> accepts.

b) $f(x) = \lceil \log_2(x + 1) \rceil$

TM M = on input string ω:

1. determine if input is member of 1+ and if not -> reject. write 0 at end tape. return to left end tape.

2. Seen the 1 and move to the right untill first 0/1. If 0-> change to 1 and go to the first unmarked 1 and continue from beginning of step; if no -> go next step. If 1-> change to 0 and go next. If empty->change it to 0 and do the same for 0.
3. go to first 0/1 on the tape. If not-> accept. marked found symbol and insert 1 at the end of the tape. Start from this step.

1.3

these machines only recognize regular languages.
we can simulate any DFA on a Turing machine with stay put instead of left. The only non-trivial modification is to add transitions from state in F to qaccept upon reading a blank, and from states outside F to qreject upon reading a blank.
Next, we start with a Turing machine M = (Q, Σ, Γ, δ, q0, qaccept, qreject) with stay put instead of left, and show how we can construct a DFA (Q′, Σ′, δ′, q′0, F) that recognizes the same language. The intuition here is that M cannot move left and cannot read anything it has written on the tape as soon as it moves right, and therefore it has essentially only one-way access to its input, much like a DFA.

First, we modify M as follows;

note that these changes do not affect the language it recognizes.
Add a new symbol so that M never writes blanks on the tape; instead, M writes the new symbol when it's going to write blanks, and we extend the transition function so that upon reading this new symbol, it behaves as though it read a blank. When M transitions into qreject or qaccept, the reading head moves right (and never stays put).

Set Q′ = Q, Σ′ = Σ, q′0 = q0, and consider the transition function:

δ′(q, σ) =

- q, if q ∈ {qaccept, qreject}
- qreject, if M starting at state q and reading σ keeps staying put.
- q′, where q′ is the state the M enters when it first moves right upon starting at state q and reading σ. (for q ∈ Q and σ ∈ Σ).

Observe that there are finitely many state-alphabet pairs, M either ends up either staying put and looping, or eventually moves right, and thus δ′ is well-defined. Finally, we define F to be the set containing qaccept and all states q ∈ Q, q 6 = qaccept, qreject such that M starting at q and reading blanks, eventually enters qaccept.

1.4

first, we show how to simulate a Turing Machine with a Queue Automaton (QA). Our queue will contain the contents of the Turing Machine tape at all times, with the symbol currently being read at the head of the queue. We will use "$" to mark the beginning of the tape.

Head/tail of queue, where a ∈ Σ is the symbol currently under the head of the Turing Machine, x ∈ Σ∗ is the contents of the tape to the right of the head, and y ∈ Σ∗ is the contents of the tape to the left of the head.

We will describe a queue “primitive” that we will use in simulating the TM. Cyclically shift right. We perform this operation is three phases. We will keep a running example to illustrate. Suppose we start out with queue contents ab$c

1. In Phase 1 we place a # marker at the end of the queue, and then replace each queue symbol x with a new symbol (w, x), where w is the symbol immediately to x’s left in the queue. For this we use a special set of separate states that “remember” the last symbol shifted. That is, for each w ∈ Σ, after having shifted symbol w, we are in a special state qw. Then when we encounter the next symbol x, we enqueue not x, but the new symbol (w, x). To start the process we enqueue # and move to state q#. We then repeat the following: From a given state qw, dequeue a symbol x, enqueue the new symbol (w, x), and move to state qx. When we dequeue # and enqueue the final new symbol we complete Phase 1. In our example the queue will now look like this: (#, a)(a, b)(b, $)($, c)(c, #)
2. In Phase 2, we repeat the following: dequeue (w, x), enqueue (w, x) until we dequeue (w, x) where x = #. We then enqueue #, enqueue w, and dequeue whatever symbol is at the head of the queue. In our example the queue will now look like this: (a, b)(b, $)($, c)#c
3. In Phase 3, we repeatedly dequeue (w, x) and enqueue w, until we dequeue #, at which point we complete Phase 3. In our example the queue will now look like this: cab$ which is our original queue cyclically shifted right one step.

Now to simulate a given step of the Turing Machine, we dequeue the symbol a at the head of the queue and then:

- if the TM transition that would be taken on reading a writes symbol b and moves right, then our QA enqueues b at the tail of the queue.
- if the TM transition that would be taken on reading a writes symbol b and movesl eft, then our QA enqueues b and then performs two cyclic shifts to the right.

At all points, the following sequence of transitions are available to the QA: dequeue $, enqueue −$ and perform 2 cyclic shifts to the right. This amounts to adding a blank “−” at the end of the tape. As with 2-NPDAs, we initialize the QA by copying the contents of the input into the queue, followed by a $.

Finally, we need to show how to simulate a QA with TM. Since 2-tape TM is equivalent to a single tape TM, as shown in lecture, it is sufficient to show how to simulate a QA with a 2-tape TM. The first tape will simply contain the input string, and the second tape will contain the queue. The tape alphabet of the QA contains special symbol $ that denotes the beginning of the queue. At first, one $ is written on the tape corresponding to the queue. When a symbol is pushed onto the queue, the head finds the first blank space on the tape and writes the symbol there. When a symbol is popped, the tape finds the first symbol on the tape other than $, reads the symbol and substitutes the symbol with $. The queue shifts on the tape to the right as the TM pops, but this is acceptable since the tape is infinite.

2

2.1

a) L = {$a^n b^n c^n$ | n ≥ 1}

The tape alphabet for an LBA, though finite, might be considerably bigger than the input alphabet.

So we can store more information on a tape than just an input string or related sentential form.

let Γ = {a, b, c, a, b, c}, and Σ = {a, b, c}.

TM M = on input string ω:

1. Scan string to ensure it has form $a^k b^m c^n$ for k, m, n ≥ 1. Along the way, mark leftmost a, b, c.
2. Scan string again to see if it's the rightmost a, b, c that are marked. If yes for all three-> ACCEPT. If yes for some but not all of a, b, c-> REJECT.
3. **Scan** string again moving the 'markers' one position to the right. aaabbbccc becomes aaabbbccc. Then Go to previous. All this can be done by a (deterministic) LBA.

b) L = {ωω | ω ∈ {a, b} ∗}

we should have all possible two tuples (sym, pos) in the tape alphabet. With this tape alphabet, we can enumerate input string, find the middle of the string and then decide if it is a repeated string or not.

TM M = on input string ω:

1) write (s, 0) to the tape.
2) read the position p and move to the next symbol. If it is not end of the input -> read the symbol s and write (s, p + 1) to the tape and go to the step 4. If it is end of the input-> from the end of the tape, read the symbol s and write (s, 0) to the tape.
3) read the position p and move to the previous symbol. If current postion p ′ is equal to p -> go to the next symbol and change it to (s, 0). If it is not equal to p-> repeat.

4) like step 3 except. If it is end of the input -> return to the left-hand end of the tape.

5) seen head, go to the next same memory position. If no -> reject, else marked and go to the first unseen symbol and continue from this step.

6) If all symbols are marked-> accept, else-> reject

2.2

Context-sensitive grammars:

L(G) = {$a^{2^\wedge i}$ | i ≥ 1}

SACaB

AaaCB

AaaE

AaEa

AEaa

Aa

- A and B serve as left and right end-markers for sentential forms (derivation of each string)
- C is a marker that moves through the string of a's between A and B, doubling their number using $Ca \rightarrow aaC$
- When C hits right end-marker B, it becomes a D or E by $CB \rightarrow DB$ or $CB \rightarrow E$
- if a D is chosen, that D migrates left using $a D \rightarrow D a$ until left end-marker A is reached
- At that point D becomes C using $AD \rightarrow AC$ and the process starts over
- Finally, E migrates left until it hits left end-marker A using $a E \rightarrow E a$
- The following are derivations in this grammar

$S \Rightarrow ACaB \Rightarrow AaaCB \Rightarrow AaaB \Rightarrow AaEa \Rightarrow AEaa \Rightarrow aa$

$S \Rightarrow ACaB \Rightarrow AaaCB \Rightarrow AaaDB \Rightarrow AaDaB \Rightarrow ADaaB \Rightarrow ACaaB \Rightarrow AaaCaB \Rightarrow AaaaaCB \Rightarrow AaaaaE \Rightarrow AaaaEa \Rightarrow AaaEaa \Rightarrow AaEaaa \Rightarrow AEaaaa \Rightarrow aaaa$

3

3.1

The set of all Turing machines can be counted because we can encode each Turing machine into a finite string with finite symbols.

The total number of turing recognizable languages can be counted since every turing recognizable language L has a corresponding turing machine TM. ($L^t \in P(\Sigma*)$ and $P(\Sigma*)$ is not countable.)

So there are infinite languages over that no Turing machine can recognize according to the theorem.

3.2

$L^{¯}$ and L are turing recognizable -> both turing decidable.

If L is Turing recognizable, there exists a TM M1 that halts and accepts any input string $w \in L$. It is not guaranteed to halt on any input string $w \notin L$.

If $L^{¯}$ is Turing recognizable, there exists a TM M2 that halts and accepts any input string $w \notin L$. Similarly to above, M2 is not guaranteed to halt on any input string $w \in L$.

Now, for every w we have two options: either w∈L or w∉L. Therefore for any input string w, exactly one of these machines will halt and accept. This allows us to construct the decider described in you proof. Create a machine M that simulates M1 and M2. With w as input, if M1 accepts-> accept w. if M2 accepts-> reject. This machine is a decider for L and therefore L is decidable. decidable languages are closed under complement so L¯ is decidable.

3.3

- every context free language L is turing decideable.
- L and L¯ are turing recognizable so both are decidable.
- Theorem 1. A language L is decidable if and only if both L and L are Turing recognizable.

 Proof. If L is decidable then it is Turing recognizable. Moreover since decidable languages are closed under complement (To design a machine for the complement of a language L, we can simulate the machine for L on an input. If it accepts then accept and vice versa.), L is also Turing recognizable. Suppose L is Turing recognizable via a TM M and L is Turing recognizable via a TM M' . Given an input x, simulate x on both the machines M and M' simultaneously (similar to union). If M accepts then accept and if M' accepts then reject. Observe that since the string x either belongs to L or L therefore one of the two machines must accept

-> complement of a context free language -> turing recognizable.

3.4

A,B turing decidable, C turing recognizable

a) B ∪ A , B ∩ A decidable.

TM MI = On input string ω:

- Run TM_A on the input. If it rejects-> reject, else-> Run TM_B on the input. If it accepts -> accept, else-> reject.

TM MC = On input string ω:

- Run TM_A on the input. If it accepts-> accept; else -> Run TM_B on the input. If it accepts-> accept else-> reject.

b) A^R decidable, C^R recognizable

Copy the input string reversed onto the tape and position the head at the beginning of it before the machine begins to read the input. and, the final symbol of the original input should be an empty symbol. run TM_A.

C^R is proof also like A^R.

c) A^- decidable, C^- maybe recognizable

L and $L^{\overline{}}$ are turing recognizable so both are decidable.

If c^{-} recognizable -> c decidable but it may not be

TM M = On input string ω:

- Run TM_A on the input. If it accepts-> reject, else-> accept.

4

4.1

a)

MN = {M | M is a TM and M halts on all inputs.}

MN is undecidable

Must show HALTTM ≤m MN by constructing the following mapping function. Since HALTTM is undecidable then MN is undecidable.

Let z by a string that does not describe a TM.

s = On input string ω:

1. If w is not of the form M,x where M is the description of a TM, then output z. Otherwise, parse w to get M and x.
2. Construct the following TM
   a. N: On input s: Run M on x.
3. Output N.

show that w ∈ HALTTM iff f(w) ∈ MN and that f always halts.

The second part, that f always halts is straightforward since it only ever does simple transformations of the descriptions of TMs. If w ∈ HALTTM, then w = M,x such that M halts on x. Thus, f(w) will be N where N always halts, since N simply runs M on x.

If w !∈ HALTTM then either w is not of the form M,x, in which case we output z which will be rejected by MN since it is not of the right form. If w !∈ HALTTM and w is of the form M,x then it means that M must loop on w. In this case, when we output N, it will always loop, and therefore not be in MN

b)

L = {M1, M2 | M1 and M2 are Turing machines and L(M1) ⊆ L(M2)}.

L is undecidable.

By reduction from E_{TM}. Assume L is decidable. Then there exists at TM M_L that decides L.

S = On input M, an encoding of a TM M:

1. Construct a new TM M0 that rejects on all inputs (ie, L(M’) = ∅).

2. Run M_L on input M, M'. If M_L accepts-> accept. ELSE -> reject.

To see that S decides E_{TM}, M_L accepts if L(M) = ∅ and rejects if L(M) != ∅ (L(M') = ∅). Therefore S(M') accepts if L(M) = ∅ and rejects otherwise. Since each step of S is guaranteed not to loop, S is a decider for E_{TM} implying E_{TM} is decidable. However, this is a contradiction because we know that E_{TM} is undecidable. Therefore our assumption that L is decidable must be false.

c) FINITETM = { M | M is a TM and L(M) is finite}

FINITETM is undecidable

By reduction from A_{TM}. Suppose FINITETM is decidable and let R be a TM that decides it. We'll construct a TM S to decide A_{TM}, which will give the desired contradiction since A_{TM} is known to be undecidable.

s = On input string ω:

1. Construct an encoding 〈Mw〉 of a TM Mw such that: Mw=On input x:
    (a) Run M on w. If M accepts w, accept. If M rejects w, reject.
2. Run R on 〈Mw〉.
3. if R accepts, reject. If R rejects, accept.

It remains to be shown that S in fact decides ATM. So suppose 〈M, w〉 is an input to S. First suppose M accepts w. Then Mw accepts all input, i.e., L(Mw) = Σ∗, an infinite language. Hence, R rejects 〈M_w〉 Thus, S accepts 〈M, w〉. Next suppose M does not accept w, i.e., M either rejects or loops on w. In either case, L(M_w) = ∅, a finite language. Hence, R accepts 〈M_w〉. Thus, S rejects 〈M, w〉 Therefore, S decides A_{TM}.

4.2

a)

If P is a nontrivial property of r.e. languages then

LP = {M: P(L(M))} is undecidable.

(1) if P(∅) then LP is not r.e.
(2) if P(∅) then LP is not co-r.e..

Case 1: ∅ ∈/ P

Assume that ∅ does not satisfy P. Find one other Turing machine N such that L(N) does satisfy P. We can now show that $A_{TM} \leq_m$ L(P). To do so, we'll take a candidate solution for A_{TM} (i.e. an encoding of a machine and a word) and transform it into a candidate solution for L(p) (i.e. an encoding of a machine) so that the result is in L(P) if and only if M accepts w.

Since we know that N satisfies p based only on its language, we can create our new machine M' such that L(M') = L(N) if M accepts w and L(M') = ∅ otherwise.

F = On input w:

1. Construct a new machine M’ as follows:
   a. M’ = On input x:
   b. Simulate M on w: If M accepts w, simulate N on x A. If N accepts x, Accept x B. If N rejects x, Reject x ii. If M rejects w, Reject x”
2. Output M’

Case 2: ∅ ∈ L(P)

Since we’re showing that L(P) is undecidable using an indirect proof, we start with the assumption that L(P) is decidable. This means that L(¬P) is also decidable. We can therefore use the same proof as in case 1, showing that L(¬P) is undecidable, which implies that L(P) is undecidable as well

b)

does L(M) have the property S?’ is undecidable

the empty language ∅ is not a member of S. We then construct, given a machine/input description 〈M, w〉 a Turing machine M′ such that L(M′) has the property S if and only if M halts when given w as input. Our assumption that ∅ ∈/ S gives us that this is equivalent to stating that L(M′) =/ ∅ if and only if M halts when given w as input.

It is no loss of generality to assume that ∅ ∈/ S. For if we wanted to show that S was undecidable and ∅ ∈ S, we could simply consider the complementary property S. This property is undecidable if and only if S is. Since S is a non-trivial property of Turing-acceptable languages, there must be some L ∈ S. Since L is Turing-acceptable, there exists a Turing machine recognizing L. We call this machine ML.

Now assume that S is decidable. This will mean that the property language LS is Turing-decidable (recursive). This in turn means that there exists a Turing machine MS which for any machine description 〈M′〉 will determine whether L(M′) ∈ S or not.