Last name (please print)	
First name (please print)	
Student Number	

WESTERN UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CS3331 - Foundations of Computer Science – Fall 2015 – Final Exam
Instructor: Dr. Lucian Ilie
Friday, Dec. 11, 2015, 7:00pm - 10:00pm
P&AB106 (Absi - Meht), P&AB148 (Meye - Zhu)

This exam consists of 6 questions worth a total of 100 marks. **No other materials are allowed, such as cheat-sheets (or any other sheets), books, or electronic devices.** All answers are to be written in this booklet. For each question, if use the back of the page or the scrap work sheets at the end to write your answers, please indicate this clearly on the page that contains the question. The exam is 120 minutes long and comprises 30% of your final mark.

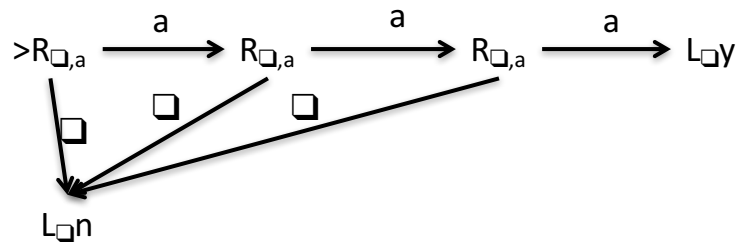
(1) 10pt	
(2) 10pt	
(3) 30pt	
(4) 10pt	
(5) 30pt	
(6) 10pt	
Grade	

1. (10pt) Construct a deterministic Turing machine M that decides the language

$$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ contains at least three } \mathtt{a}\text{'s}\} \ .$$

M starts with the initial configuration $(s, \sqcup w)$ and halts with the configuration $(q, \sqcup w)$, in the appropriate state $q \in \{y, n\}$. Describe M using the macro language (e.g.: $\mathtt{a}R_{\sqcup}\mathtt{b}L_{\#}$)

Solution:



2. (10pt) The set SD is closed under intersection. Is the following a correct proof of this fact? Explain your answer.

   Assume $L_1, L_2 \in$ SD, semidecided by two Turing machines M_1 and M_2, respectively. We can build a machine M to decide $L_1 \cap L_2$ as follows:
   1. on input w
   2. run M_1 on w
   3. if M_1 accepts, then
   4. run M_2 on w
   5. if M_2 accepts, then accept

   Solution:

   Yes, it is. If $w \in L_1 \cap L_2$, then M_1 halts in step 2 and accepts in step 3, then M_2 halts in step 4 and accepts in step 5, so M accepts and $w \in L(M)$.

   If $w \in L(M)$, then M must have accepted in step 5. But the only way to get there is when both M_1 and M_2 accept. So $w \in L_1 \cap L_2$.

3. (30pt) Can you give an example of a language L such that:

   (a) $L \in$ D and $\neg L \in$ SD $-$ D?

   (b) $L, \neg L \in$ SD $-$ D?

   (c) $L \in$ SD, $\neg L \notin$ SD?

   (d) $L, \neg L \notin$ SD?

   Prove your answers. ($\neg L$ is the complement of L.)

   Solution:

   (a) No. If $L \in$ D, then $\neg L \in$ D.

   (b) No. If $L, \neg L \in SD$, then $L, \neg L \in D$.

   (c) H, the halting language, is known to be in SD $-$ D. By the above result, it must be that $\neg H \notin$ SD.

   (d) Consider $L = \{< M > |\ L(M) \text{ is regular}\}$. We show that $L, \neg L \notin$ SD.

   (i) $L \notin$ SD. Reduction from $\neg H$.
   $R(< M, w >) =$

   1. construct $M_{\#}$:
      1.1 on input x, save x for later
      1.2 erase tape
      1.3 write w on tape
      1.4 run M on w
      1.5 put x back on the tape
      1.6 if $x \in A^n B^n$, then accept, else reject
   2. return $< M_{\#} >$

   Assume *Oracle* that semidecides L.

   $< M, w > \in \neg H$: M does not halt on w, so M never makes it to 1.5 and hence $M_{\#}$ accepts $\emptyset$, which is regular. *Oracle* accepts.

   $< M, w > \notin \neg H$: M halts on w, so $M_{\#}$ accepts $A^n B^n$, which is not regular. *Oracle* does not accept.

   (ii) $\neg L = \{< M > |\ L(M) \text{ is not regular}\} \notin$ SD. Reduction from $\neg H$.
   $R(< M, w >) =$

   1. construct $M_{\#}$:
      1.1 on input x, if $x \in A^n B^n$, then accept
      1.2 erase tape
      1.3 write w on tape
      1.4 run M on w
      1.5 accept
   2. return $< M_{\#} >$

   Assume *Oracle* that semidecides $\neg L$.

   $< M, w > \in \neg H$: M does not halt on w, so M never makes it to 1.5 and hence $M_{\#}$ accepts $A^n B^n$, which is not regular. *Oracle* accepts.

   $< M, w > \notin \neg H$: M halts on w, so $M_{\#}$ accepts everything, which is regular. *Oracle* does not accept.

4. (10pt) Assume a language L and a Turing machine M that lexicographically enumerates L. Construct a Turing machine M' that decides $\neg L^R$ (the complement of the reversal of L). (Only clear English description is required.)

   Solution: First, it is clear that $w \in \neg L^R$ iff $w \notin L^R$ iff $w^R \notin L$. The machine M', on input w, constructs first w^R and then checks whether $w^R \notin L$ as follows: M' runs M until one of these happens:

   (a) M enumerates w^R – M' rejects ($w^R \in L$)

   (b) M enumerates a string w' that is lexicographically larger than w^R – M' accepts ($w^R \notin L$)

   (c) M halts – M' accepts ($w^R \notin L$)

5. (30pt) For each of the following languages, prove whether it is in D, SD − D, or ¬SD. Explain first intuitively why you think it is in D, SD − D, or ¬SD, then prove your assertion rigurously.

   (a) $L_1 = \{< M > |\ L(M) \text{ is finite}\}$.

   (b) $L_2 = \{< M > |\ L(M) \neq \emptyset\}$.

   (c) $L_3 = \{< M_1, M_2 > |\ L(M_1) - L(M_2) \text{ is infinite}\}$.

   (d) $L_4 = \{< M, w > |\ M \text{ accepts } w \text{ and rejects } w^R\}$.

   Solution:

   (a) $L_1 \in \neg$SD. Reduction from $\neg H$.
   $R(< M, w >) =$

   1. construct $M_\#$:
      1.1 erase tape
      1.2 write w on tape
      1.3 run M on w
      1.4 accept
   2. return $< M_\# >$

   Assume *Oracle* that semidecides L_1.

   $< M, w > \in \neg H$: M does not halt on w, so M never makes it to 1.4 and hence $M_\#$ accepts $\emptyset$, which is finite. *Oracle* accepts.

   $< M, w > \notin \neg H$: M halts on w, so $M_\#$ accepts everything, which is infinite. *Oracle* does not accept.

   (b) $L_2 \in$ SD − D. It is in SD because we can run M in dovetailing mode on all strings and accept as soon as one string is accepted.
   $L_2 \notin$ D. Reduction from H.
   $R(< M, w >) =$

   1. construct $M_\#$:
      1.1 erase tape
      1.2 write w on tape
      1.3 run M on w
      1.4 accept
   2. return $< M_\# >$

   Assume *Oracle* that semidecides L_2.

   $< M, w > \in H$: M halts on w, so $M_\#$ accepts everything, which is non-empty. *Oracle* accepts.

   $< M, w > \notin H$: M does not halt on w, so M never makes it to 1.4 and hence $M_\#$ accepts $\emptyset$. *Oracle* does not accept.

   (c) $L_3 \in \neg$SD. Reduction from $\neg H$.
   $R(< M, w >) =$

   1. construct $M_\#$:
      1.1 erase tape
      1.2 write w on tape
      1.3 run M on w
      1.4 accept

2. Construct $M_\flat$
   2.1 accept
3. return $< M_\flat, M_\# >$

Assume *Oracle* that semidecides L_3. Note that always $L(M_\flat) = \Sigma^*$.

$< M, w > \in \neg H$: M does not halt on w, so M never makes it to 1.4 and hence $M_\#$ accepts $\emptyset$. Thus $L(M_\flat) - L(M_\#) = \Sigma^*$ is infinite. *Oracle* accepts.

$< M, w > \notin \neg H$: M halts on w, so $M_\#$ accepts Σ^*. Thus $L(M_\flat) - L(M_\#) = \emptyset$ is finite. *Oracle* does not accept.

(d) $L_4 \in$ SD $-$ D. $L_4 \in$ SD because we can run M in parallel (or not) on both w and w^R and accept iff w is accepted and w^R is rejected.

$L_4 \notin$ D. The following "obvious" reduction from H has a problem.

$R(< M, w >) =$

1. construct $M_\#$:
   1.1 on input x, if $x = w^R$, reject
   1.2 erase tape
   1.3 write w on tape
   1.4 run M on w
   1.5 accept
2. return $< M_\#, w >$

Assume *Oracle* that semidecides L_4.

$< M, w > \in H$: M halts on w, so $M_\#$ accepts w. Since $M_\#$ always rejects w^R, *Oracle* accepts.

$< M, w > \notin H$: M does not halt on w, so M never makes it to 1.5 and hence $M_\#$ does not accept anything, in particular, it does not accept w. *Oracle* rejects.

The problem is that it does not work when $w = w^R$. In this case, if M halts on w, then $M_\#$ rejects both w and w^R. Therefore, we need to use a string that is not a palindrome.

$R(< M, w >) =$

1. construct $M_\#$:
   1.1 on input x, if $x =$ `ba`, reject
   1.2 erase tape
   1.3 write w on tape
   1.4 run M on w
   1.5 if $x =$ `ab`, then accept, else reject
2. return $< M_\#,$ `ab` $>$

Assume *Oracle* that semidecides L_4.

$< M, w > \in H$: M halts on w, so $M_\#$ accepts `ab`. Since $M_\#$ rejects `ba`, *Oracle* accepts.

$< M, w > \notin H$: M does not halt on w, so M never makes it to 1.5 and hence $M_\#$ does not accept anything, in particular, it does not accept `ab`. *Oracle* rejects.

6. (10pt)

   (a) Does the following instance of PCP have any solutions? Prove your answer.

a	bbb	aab	b
bab	bb	ab	a

   (b) If you can determine, as you did at (a), whether a PCP instance has solutions or not, how is it possible that the Post Correspondence Problem is undecidable?

   (c) Can you give an example of an instance of PCP that has only one solution? Prove your answer.

Solution:

(a) Yes, here is a solution:

bbb	a	bbb
bb	bab	bb

(b) It may be possible to decide particular instances of PCP. The Post Correspondence Problem being undecidable means that there is no algorithm that answers correctly provided with *any* instance of PCP as input.

(c) No. Any concatenation of solutions is also a solution, so any PCP instance has either zero or infinitely many solutions.

(scrap work)

(scrap work)

(scrap work)

(scrap work)

(scrap work)

(scrap work)