Monash University

[illegible] Information Technology

[illegible] Theory of Computation
FINAL EXAM
SOLUTIONS

2nd semester, 2014

Instructions:

10 minutes reading time.

3 hours writing time.

No books, calculators or devices.

Total marks on the exam = 120.

Answers in blue.

Comments in green.

Working Space

Question 1 **(4 marks)**

Use De Morgan's Laws to prove that $\neg((P \wedge Q) \wedge (\neg P \wedge \neg Q))$ is a tautology.

You may use other principles of propositional logic too, if you wish. But your proof must make significant use of De Morgan's Laws.

$$
\begin{aligned}
\neg((P \wedge Q) \wedge (\neg P \wedge \neg Q)) &= \neg(P \wedge Q) \vee \neg(\neg P \wedge \neg Q) \\
&= (\neg P \vee \neg Q) \vee (\neg\neg P \vee \neg\neg Q) \\
&= (\neg P \vee \neg Q) \vee (P \vee Q) \\
&= \neg P \vee \neg Q \vee P \vee Q \\
&= \neg P \vee P \vee \neg Q \vee Q \\
&= \text{True} \vee \text{True} \\
&= \text{True.}
\end{aligned}
$$

Official use only
4

Question 2 **(6 marks)**

Suppose you have predicates algorithm and TuringMachine with the following meanings, where variable X represents an arbitrary function:

algorithm(X): there is an algorithm for X.
TuringMachin [illegible] uring machine for computing X.

(a) Write a universa [illegible] cate logic with the meaning:

"Every fun [illegible] thm can be computed by a Turing machine."

$$\forall X : \texttt{algorithm}(X) \Rightarrow \texttt{TuringMachine}(X)$$

Equivalent answer:

$$\forall X : \neg\texttt{algorithm}(X) \vee \texttt{TuringMachine}(X)$$

(b) By what name is this assertion usually known?

Church-Turing thesis

(c) What reasons are there for believing that this assertion is true?

- So far, algorithms that people try to program have turned out to be programmable.
- No counterexample. (I.e., there is no algorithm which does not seem to be programmable in principle.)
- Different approaches to computability end up in agreement.
  Recursive functions, and lambda calculus, both arrive at the same class of computable functions as those captured by Turing machines.

Question 3 **(3 marks)**

Give a regular expression for the language of all strings over alphabet {a,b} whose first and last letters are different.

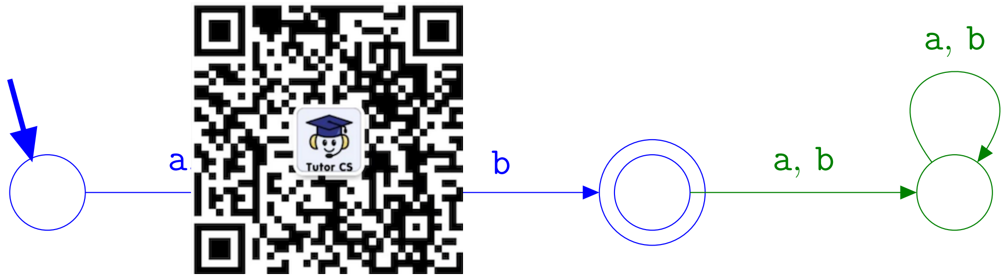
$(\texttt{a}(\texttt{a}\cup\texttt{b})^*\texttt{b}) \cup (\texttt{b}(\texttt{a}\cup\texttt{b})^*\texttt{a})$

It's ok to use [ab] instead of (a∪b).

Official use only
9

Question 4 **(4 marks)**

Let L be the language of nonempty strings over {a,b} that can start and finish with any letter but all other letters must be a. Draw a Finite Automaton to recognise L.



Question 5 **(3 marks)**

When converting a Nondeterministic Finite Automaton (NFA) to an equivalent Finite Automaton (FA), how do you determine which states in the NFA are combined to form the single Start state in the FA?

Every state that is reachable *from* the starting state along a sequence of empty (i.e., ε) transitions.

No need to set it out as an algorithm; a clear description of which states are so combined is enough. But it's also ok to set it out as an algorithm, in which case the solution would look something like this:

Mark the Start state.
While there is an empty transition *from* a marked state *to* an unmarked state:
    Mark this unmarked state.
Output: the set of marked states.

Official use only

7

Question 6 **(4 marks)**

The following table describes a Finite Automaton with three states.

Find another FA that is equivalent to this one and has only two states.

Write your two-[illegible]nd table below.

state	a	b
1	2	3
2	1	3
3	1	2

YOUR ANSWER:

	state	a	b
Start	1,2	1,2	3
Final	3	1,2	1,2

The state here called "1,2", formed from merging states 1 & 2 in the original table, could be called any other name so long as the name is used consistently throughout the table and is different to the name used for the Final state.

For example, the following is fine:

	state	a	b
Start	1	1	2
Final	2	1	1

Official use only
4

Question 7 **(4 marks)**

Explain why the class of regular languages over the alphabet {a,b} is closed under interchange of a and b.

Definition:

If L is any lang[illegible]bet {a,b}, then the *interchange of* a *and* b forms a new language as foll[illegible]ng in L, and replace every a by b and every b by a, simultaneously. So, [illegible]ing abb becomes baa.

Any regular language has a regular expression (such that the strings in the language are precisely those matched by the regular expression). Interchanging a and b in the regular expression gives another regular expression, and this defines a new language obtained from the original one by interchanging a and b. So this new language must be regular.

Alternative answer, using finite automata instead of regular expressions:

Any regular language has a Finite Automaton that recognises it. Interchanging a and b thoughout such an FA gives a new FA that recognises the language formed from the original one by interchanging a and b. Since this language is recognised by an FA, it must be regular, by Kleene's Theorem.

Official use only
4

Question 8 **(7 marks)**

Consider the following Context-Free Grammar. Its non-terminals are S and L.

$$
\begin{aligned}
S &\rightarrow L && (1)\\
L &\rightarrow \mathtt{ha}L && (2)\\
L &\rightarrow \mathtt{a}L && (3)\\
L &\rightarrow \mathtt{h} && (4)
\end{aligned}
$$

Give

(a) a derivation, and

(b) a parse tree,

for the string `ahahaah` .

Label each step in the derivation on its right by the rule used. Use the spaces below for your answers.

(a)

$$
\begin{aligned}
S &\Rightarrow L && (1)\\
&\Rightarrow \mathtt{a}L && (3)\\
&\Rightarrow \mathtt{aha}L && (2)\\
&\Rightarrow \mathtt{ahaha}L && (2)\\
&\Rightarrow \mathtt{ahahaa}L && (3)\\
&\Rightarrow \mathtt{ahahaah} && (4)
\end{aligned}
$$

(b)

Official use only
7

Working Space

Question 9 **(3 marks)**

Give a regular grammar for the language of positive integers represented in binary with leading bit 1. (The *leading* bit is the most significant bit, i.e., the leftmost bit.)

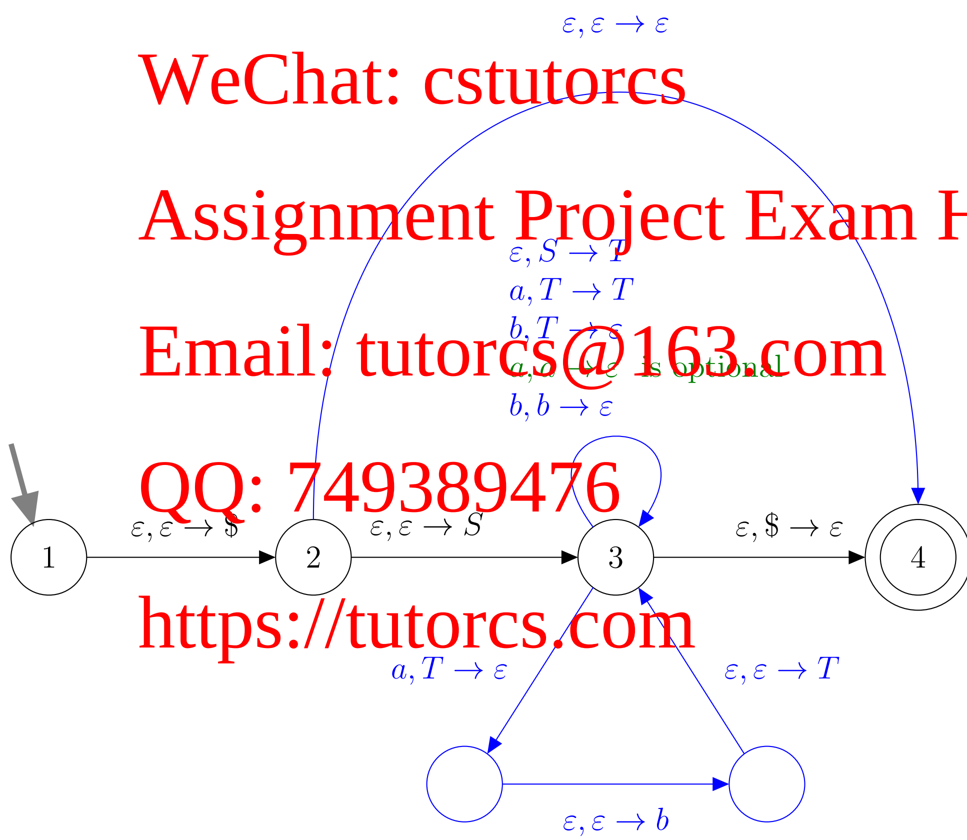
$$
\begin{aligned}
S &\rightarrow 1L\\
L &\rightarrow 0L\\
L &\rightarrow 1L\\
L &\rightarrow \varepsilon
\end{aligned}
$$

Question 10 **(6 marks)**

Given the following Context-Free Grammar:

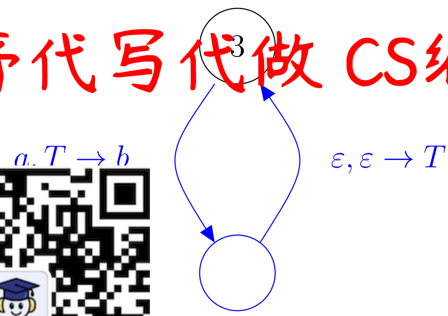
$$
\begin{aligned}
S &\rightarrow \varepsilon && (1)\\
S &\rightarrow T && (2)\\
T &\rightarrow aTb && (3)\\
T &\rightarrow aT && (4)\\
T &\rightarrow b && (5)
\end{aligned}
$$

complete the followi[illegible] a Pushdown Automaton for the language generated by the grammar.



An alternative to the arc $2 \rightarrow 4$ labelled $\varepsilon, \varepsilon \rightarrow \varepsilon$ is to omit that arc and, instead, have an extra label $\varepsilon, S \rightarrow \varepsilon$ on the loop at state 3.

An alternative to the 3-cycle underneath node 3 would be the following 2-cycle, which has the same effect and is also correct.



Official use only
9

Question 11 **(10 marks)**

This question uses the same grammar as the previous question.

Let L be the language generated by this grammar.

(a) Prove by in[illegible]ring of the form $\mathtt{a}^m\mathtt{b}^n$, where $m \geq n-1$ and $n \geq 1$, belongs to L.

We prove this b[illegible]ength of the string, i.e., on $m+n$.

Inductive Basis:

If the string has length 1, then it just consists of the string $\mathtt{b}$, which is in L by the derivation $S \Rightarrow T \Rightarrow \mathtt{b}$.

Inductive step:

Assume that any string of the form $\mathtt{a}^m\mathtt{b}^n$ with length $< k$, and $m \geq n-1$ and $n \geq 1$, belongs to L, where $k \geq 2$. This is the **Inductive Hypothesis**.

Now suppose we have a string $\mathtt{a}^m\mathtt{b}^n$ of length $k \geq 2$. FIrstly, observe that $m \geq 1$, since if $m = 0$ then $n = 1$ (by $m \geq n-1$ and $n \geq 1$) which implies the string length, k, is only 1 (whereas here we have $k \geq 2$). So we know that the string starts with $\mathtt{a}$ and ends with $\mathtt{b}$.

If the string has only one $\mathtt{b}$ (i.e., $n = 1$), then it has the form $\mathtt{a}^m\mathtt{b}$ (where $m \geq 1$, since the length k is ≥ 2). Consider the shorter string $\mathtt{a}^{m-1}\mathtt{b}$. Since its length is $k-1$, the Inductive Hypothesis can be used. This tells us that this shorter string belongs to L. So there is a derivation

$$S \Rightarrow T \Rightarrow \cdots \Rightarrow \mathtt{a}^{m-1}\mathtt{b}.$$

(Observe that the first step of any derivation of a nonempty string in this grammar must be $S \Rightarrow T$.) Removing the first step, we have a derviation of $\mathtt{a}^{m-1}\mathtt{b}$ from T, i.e.,

$$T \Rightarrow \cdots \Rightarrow \mathtt{a}^{m-1}\mathtt{b}.$$

Prefixing each string in this derivation gives a derivation of $\mathtt{a}^m\mathtt{b}$ from $\mathtt{a}T$:

$$\mathtt{a}T \Rightarrow \cdots \Rightarrow \mathtt{a}^m\mathtt{b}.$$

But we also have a derivation of $\mathtt{a}T$ from S, namely $S \Rightarrow T \Rightarrow \mathtt{a}T$ (using rule (2) then (4)). Putting these together gives a derivation of $\mathtt{a}^m\mathtt{b}$ from S:

$$S \Rightarrow T \Rightarrow \mathtt{a}T \Rightarrow \cdots \Rightarrow \mathtt{a}^m\mathtt{b}.$$

So our original string of length k does indeed belong to L.

It remains to deal with the case where our string has more than one $\mathtt{b}$, i.e., $n \geq 2$.

Since $m \geq n-1$ and $n \geq 2$, we have $m \geq 1$. The string $\mathtt{a}^{m-1}\mathtt{b}^{n-1}$ has length $k-2$, which is $< k$. Also, $m-1 \geq (n-1)-1$ and $n-1 \geq 1$. So the inductive hypothesis applies. We deduce that this string belongs to L, so there must be some derivation

$$S \Rightarrow \cdots \Rightarrow \mathtt{a}^{m-1}\mathtt{b}^{n-1}.$$

This includes a deri[illegible]

$$[illegible] \cdots \Rightarrow \mathtt{a}^{m-1}\mathtt{b}^{n-1}.$$

Adding a at the start [illegible] f each string in this derivation gives a new derivation,

$$\mathtt{a}\,T\,\mathtt{b} \Rightarrow \cdots \Rightarrow \mathtt{a}^{m}\mathtt{b}^{n}.$$

We also have the derivation $S \Rightarrow T \Rightarrow \mathtt{a}\,T\,\mathtt{b}$ (using rule (2) then (3)). Putting these derviations together gives a derivation of $\mathtt{a}^{m}\mathtt{b}^{n}$ from S:

$$S \Rightarrow T \Rightarrow \mathtt{a}\,T\,\mathtt{b} \Rightarrow \cdots \Rightarrow \mathtt{a}^{m}\mathtt{b}^{n}.$$

So the string $\mathtt{a}^{m}\mathtt{b}^{n}$ belongs to L.

We have now established this conclusion for any string of the form $\mathtt{a}^{m}\mathtt{b}^{n}$ of length k, with $m \geq n-1$ and $n \geq 1$, belongs to L.

By the Principle of Mathematical Induction, it follows that any string of this form, of any length, belongs to L.

(b) Prove or disprove: some string in L has a derivation, using this grammar, that is neither a leftmost derivation nor a rightmost derivation.

This is not true. At any stage of any derivation in this grammar, there is only one non-terminal. So there is no choice of which nonterminal in the string is to be used for the next production rule. In particular, whichever production rule we use, we will always be applying it to the leftmost nonterminal (since it is the only nonterminal) — and, also to the rightmost nonterminal. So, in fact, every derivation is a leftmost derivation, and also a rightmost derivation.

(There might sometimes be a choice of which production rule to use for L, since there are several different production rules for L. But this is not relevant to the issue at hand.)

Official use only
10

Working Space

Question 12 **(7 marks)**

Recall that the **Fibonacci numbers**, F_n, are defined recursively as follows:

$$F_1 = 1,$$

$$\ldots_{-1} + F_{n-2} \qquad \text{for } n \geq 3.$$

The first few numbe… re

$$\ldots 3, 5, 8, 13, 21, 34, \ldots$$

Note that the Fibon… …ter the first term) an *increasing* sequence of positive integers.

The language FIBONACCI is defined to be the set of all strings of the letter a whose length is a Fibonacci number. So,

$$\text{FIBONACCI} = \{ \, \mathtt{a}^{F_n} \; : \; n \in \mathbb{N} \, \}.$$

(a) Prove that the difference, $F_n - F_{n-1}$, between two consecutive Fibonacci numbers increases as n increases, i.e., $F_n - F_{n-1} > F_{n-1} - F_{n-2}$ for all $n \geq 5$.

$F_n - F_{n-1} = F_{n-2}$, and the Fibonacci numbers are increasing, so their differences are increasing too.

(b) Using (a), prove that the language FIBONACCI is not context-free.

Suppose (by way of contradiction) that FIBONACCI is context-free.

Then it has a CFG, in Chomsky Normal Form, with some number k of nonterminal symbols.

Let w be any word in the language FIBONACCI that has length $> 2^{k-1}$. Observe that it must be of the form $\mathtt{a}^{F_n}$ for some $F_n > 2^{k-1}$.

By the Pumping Lemma for CFLs, the word w can be partitioned into strings u, v, x, y, z (i.e., $w = uvxyz$) such that:

- v, y are not both empty,
- $|vxy| \leq 2^k$ …which we won't need …, and
- uv^ixy^iz is in FIBONACCI for all $i \geq 0$.

For convenience, write p for the combined length of v and y. So $p = |v| + |y|$. The first and second points above tell us that $1 \leq p \leq 2^k$.

Since $w = uvxyz$ is just a string of F_n a's, the string uv^2xy^2z is just a string of $F_n + p$

a's (since we have just taken w and repeated both v and y). More generally, the string uv^ixy^iz is just a string of $F_n + (i-1)p$ a's. The third point above means that a string of $F_n + (i-1)p$ a's must belong to the language FIBONACCI, for all i. So we have an infinite sequence of strings in FIBONACCI in which each string is exactly p letters longer than its predecessor, and thi[illegible]ve since $p \geq 1$.

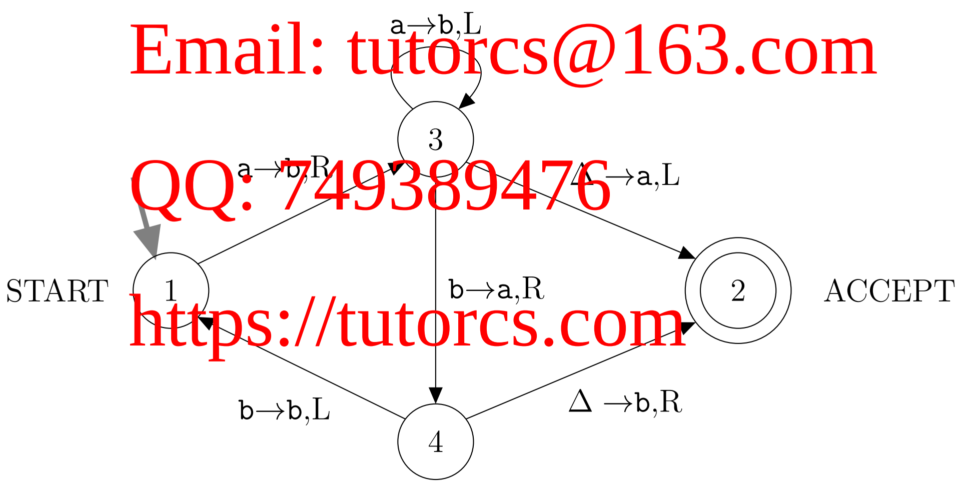
But we saw in part [illegible]ce between successive Fibonacci numbers is increasing. So, whatever p [illegible] sufficiently large m, the difference $F_m - F_{m-1} > p$. This means that so[illegible] $F_n + (i-1)p$ cannot be Fibonacci numbers after all, since some of th[illegible] ever-increasing gaps between consecutive Fibonacci numbers. This mea[illegible]ugh i) some of the strings of $F_n + (i-1)p$ a's cannot be in FIBONACCI after all. This is a contradiction.

So our original assumption, that FIBONACCI is context free, must be false. Hence FIBONACCI is not context-free.

Official use only
7

Question 13 **(9 marks)**

Consider the following Turing machine.



Trace the execution of this Turing machine, writing your answer in the spaces provided on the next page.

The lines show the configuration of the Turing machine at the start of each step. For each line, fill in the state and the contents of the tape. On the tape, you should indicate the currently-scanned character by underlining it, and you should show the first blank character as Δ (but there is no need to show subsequent blank characters).

To get you started, the first line has been filled in already.

	State	Tape					
At start of step 1:	1	$\underline{a}$	a	b	$\triangle$		
At start of step 2:	[illegible]	b	$\underline{a}$	b	$\triangle$		
At start of step 3:	[illegible]	$\underline{b}$	b	b	$\triangle$		
At start of step 4:	4	a	$\underline{b}$	b	$\triangle$		
At start of step 5:	1	$\underline{a}$	b	b	$\triangle$		
At start of step 6:	3	b	$\underline{b}$	b	$\triangle$		
At start of step 7:	4	b	a	$\underline{b}$	$\triangle$		
At start of step 8:	1	b	$\underline{a}$	b	$\triangle$		
At start of step 9:	3	b	b	$\underline{b}$	$\triangle$		
At start of step 10:	4	b	b	a	$\underline{\triangle}$		
At start of step 11:	2	b	b	a	b	$\underline{\triangle}$	
At start of step 12:							

Official use only
9

Question 14 **(5 marks)**

For each of the following decision problems, indicate whether or not it is decidable.

You may assume that, when Turing machines are encoded as strings, this is done using the Code-Word Language (CWL).

Decision Problem	your answer (tick **one** box in each row): Decidable	Undecidable
Input: a Turing ma[illegible]sitive integer k. Question: Does M [illegible] string of at most k letters?	☐	✓
Input: a Turing machine M, and a positive integer k. Question: Does the encoding of M have at least k letters?	✓	☐
Input: a Turing machine M, and a positive integer k. Question: When M is given an encoding of itself as input, does the computation go for at least k steps?	✓	☐
Input: a Turing machine M. Question: Does M eventually print an encoding of M on the tape and then halt?	☐	✓
Input: a Turing machine M, and a string w. Question: Does M reject w?	☐	✓

Official use only
5

Working Space

Question 15 **(12 marks)**

The Venn diagram on the right shows several classes of languages. For each language (a)–(l) in the list below, indicate which classes it belongs to, and which it doesn't belong to, by placing its corresponding letter in the correct region of the diagram.

If a language do[illegible] of these classes, then place its letter above the top of the diagram.

You may assum[illegible] machines are encoded as strings, this is done using the Code-Word Lan[illegible]

(a) The set of all [illegible] adjacency matrices.

(b) The set of all 2-colourable graphs. (A graph is *2-colourable* if each vertex can be coloured Black or White in such a way that adjacent vertices receive different colours.)

(c) The set of all 3-colourable graphs. (Definition is like 2-colourable, except now the available colours are Red, White and Black.)

(d) The set of all satisfiable Boolean expressions in Conjunctive Normal Form with at least two literals in each clause.

(e) EQUAL, the set of all strings over alphabet {`a`,`b`} with an equal number of `a`'s and `b`'s.

(f) The set of all strings of parentheses such that the parentheses are correctly matched.

(g) The set of all strings in which every letter is next to an identical letter.

(h) The set of all Finite Automata that define the empty language.

(i) The set of all Turing Machines with polynomial time complexity.

(j) The set of all Turing machines which, when given themselves as input, eventually either reject or loop forever.

(k) The set of all Turing machines whose language of accepted strings is regular.

(l) The set of all Turing machines that have ever been written.

Official use only
12

Question 16 **(7 marks)**

Prove that the following problem is undecidable.

Input: a Turing machine M.

Question: Is there a string x which, if given as input to M, causes it to eventually erase everything on the ta[illegible]very letter by Blank), after which it never writes any other symbol?

You may use th[illegible]ng problem is undecidable.

Suppose (by way of contradiction) that this problem is decidable. Then there exists a decider, D, for it.

Now, take any input M to the Diagonal Halting Problem.

Construct a new Turing machine M' from M as follows.

M':

1. Input: x
2. Mark the first tape cell, then move the tape head to just past the end of x, i.e., to the first Blank cell.
3. Simulate the running of M on input M. This simulation treats the first Blank cell after x as the start of its tape, just for the simulation. So the simulation does not interfere with x. (The code for doing this simulation of M is hardcoded into M'. Note that M' only does this for the one specific machine M that it was constructed from.)
4. Go back to the start of the tape (i.e., to the very first letter of x, which was specially marked).
5. Move along the tape to the right, one cell at a time, overwriting each cell with Blank. (It doesn't matter if this goes on forever.)

Now, if M halts on input M, then the simulation step in M' (Step 3) will eventually finish. Then M' goes on to Step 4, so it then erases the tape. This happens regardless of what the input x was, i.e., for any x.

On the other hand, if M does not halt on input M, then M' is stuck in Step 3 forever. This simulation does not erase the tape, since x is still sitting unaltered at the start of the tape.

We conclude that M halts on input M if and only if M' eventually erases the entire tape, for some input.

We can therefore use a decider for the given problem to decide the Diagonal Halting Problem. Given M, we construct M' and let x be any nonempty string. We use D to decide whether or not M' permanently erases the tape, and the answer from D immediately tells us whether M halts on input M.

So we have a decider for the Diagonal Halting Problem. But that problem is known to be undecidable. So we have a contradiction. So the assumption that the given problem is decidable must be incorrect. Therefore it is undecidable.

Question 17 **(3 marks)**

Explain how to obtain, from any language that is recursively enumerable but not decidable, another closely related language that is not recursively enumerable. (Proof is not required.)

Take the compl[illegible]ge.

Official use only
10

Question 18 **(15 marks)**

Consider the language 3-EDGE-COLOURABILITY, which consists of all graphs G such that we can assign colours to the edges of the graph so that (i) each edge is Red, White or Black, and (ii) any two edges incident at a common vertex must get different colours.

Let H be the following graph.

(a) Construct a Boolean expression E_H in Conjunctive Normal Form such that the satisfying truth assignments for E_H correspond to solutions to the 3-EDGE-COLOURABILITY problem on the above graph H (i.e., they correspond to colourings of the edges of H, using at most three colours and which give different colours to incident edges).

To do this, use variable names $a_R, a_W, a_B, b_R, b_W, b_B, c_R, c_W, c_B, d_R, d_W, d_B$, where variable e_X is True if and only if edge e gets colour X (where $e \in \{a, b, c, d\}$ and $X \in \{R, W, B\}$).

The solution needs to describe the *conjunction* of all clauses

$$e_R \vee e_W \vee e_B$$

over all edges e (these clauses ensure that each edge gets at least one colour), together with all clauses

$$\neg e_X \vee \neg f_X$$

over all pairs of incident edges e, f and all $X \in \{R, W, B\}$ (these clauses ensure that incident edges get different colours).

The possible pairs of incident edges are: a, b; a, c; a, d; b, c; c, d. (This is all pairs of edges except b, d.)

$$\bigwedge_{e \in \{a,b,c,d\}} (e_R \vee e_W \vee e_B) \quad \wedge \quad \bigwedge_{\substack{e \text{ incident with } f \\ X \in \{R,W,B\}}} (\neg e_X \vee \neg f_X)$$

OR

$$(a_R \vee a_W \vee a_B) \wedge (b_R \vee b_W \vee b_B) \wedge (c_R \vee c_W \vee c_B) \wedge (d_R \vee d_W \vee d_B) \wedge$$
$$(\neg a_R \vee \neg b_R) \wedge (\neg a_W \vee \neg b_W) \wedge (\neg a_B \vee \neg b_B) \wedge$$

$(\neg a_R \vee$ [illegible] $) \wedge (\neg a_B \vee \neg c_B) \wedge$

$(\neg a_R \vee$ [illegible] $) \wedge (\neg a_B \vee \neg d_B) \wedge$

$(\neg b_R \vee$ [illegible] $) \wedge (\neg b_B \vee \neg c_B) \wedge$

$(\neg c_R \vee$ [illegible] $) \wedge (\neg c_B \vee \neg d_B)$

OR something like . . .

$$(a_R \vee a_W \vee a_B) \vee \cdots \vee (d_R \vee d_W \vee d_B) \vee$$
$$(\neg a_R \vee \neg b_R) \wedge (\neg a_W \vee \neg b_W) \wedge (\neg a_B \vee \neg b_B) \wedge$$
$$(\neg a_R \vee \neg c_R) \wedge \quad \cdots$$
$$\cdots$$
$$\cdots \wedge (\neg c_B \vee \neg d_B)$$

. . . provided it is clear which clauses are included, or how they are constructed.

Normally, in reducing 3-EDGE-COLOURABILITY to SATISFIABILITY, you need extra clauses to ensure that each edge gets *at most* one colour. Such clauses have the form

$$\neg e_X \vee \neg e_Y$$

for each edge e and each pair X, Y of distinct colours from the colour set $\{R, W, B\}$. BUT, in this case, it so happens that the structure of the graph ensures that this constraint is imposed by the other clauses anyway. So these clauses are not required in this case.

(b) Give a polynomial-time reduction from 3-EDGE-COLOURABILITY to SATISFIABILITY.

Input: Graph G.
For each edge e of G, create three new variables, and put them in a clause, $e_R \vee e_W \vee e_B$, and also create the … $\neg e_W$, $\neg e_R \vee \neg e_B$, $\neg e_W \vee \neg e_B$.
For each pair of inci… :
{
    For each colo…
    {
        Create t… $\vee \neg f_X$.
    }
}
Output: the conjunction of all the clauses created so far.

Official use only
15

Question 19 **(8 marks)**

Prove that the HAMILTONIAN CIRCUIT problem is NP-complete, by reduction from HAMILTONIAN PATH. You may assume that HAMILTONIAN PATH is NP-complete.

Definitions:

A **Hamiltonian** [illegible] G is a path that includes every vertex of G. All the vertices on the path [illegible]

A **Hamiltonian** [illegible] G is a circuit that includes every vertex of G. All the vertices on the circuit must be distinct.

HAMILTONIAN PATH
Input: Graph G.
Question: Does G have a Hamiltonian path?

HAMILTONIAN CIRCUIT
Input: Graph G.
Question: Does G have a Hamiltonian circuit?

HAMILTONIAN CIRCUIT belongs to NP:

Given a graph G, let the certificate be a Hamiltonian circuit of G. This can be verified in polynomial time, by checking that the circuit is indeed a circuit and that it visits each vertex exactly once.

Polynomial-time reduction from HAMILTONIAN PATH to HAMILTONIAN CIRCUIT:

Given a graph G (which may or may not have a Hamiltonian path), construct a new graph H by adding a new vertex v and joining it, by n new edges, to every vertex of G. (Here, n denotes the number of vertices of G.)

We show that G has a Hamiltonian path if and only if H has a Hamiltonian circuit.

Suppose G has a Hamiltonian path. Call its end vertices u and w. Then a Hamiltonian circuit of H can be obtained by adding the new vertex v, and the edges uv and wv, to the Hamiltonian path. So H has a Hamiltonian circuit.

Conversely, suppose H has a Hamiltonian circuit C. This circuit must include v, and two edges incident with v. Let u and w be the two vertices of G that are incident with v in C. (So C includes the edges uv and wv as well as the vertex v.) The rest of C must constitute a Hamiltonian path between u and w in G. So G has a Hamiltonian path.

This completes the proof that G has a Hamiltonian path if and only if H has a Hamiltonian circuit.

It remains to observe that the construction of H from G can be done in polynomial time.

Therefore the construction of H from G is a polynomial-time reduction from HAMILTONIAN PATH to HAMILTONIAN CIRCUIT.

Since HAMILTONIAN PATH is NP-complete, and it's polynomial-time reducible to HAMILTONIAN CIRCUIT, and HAMILTONIAN CIRCUIT is in NP, we conclude that HAMILTONIAN CIRCUIT is NP-complete.

Official use only
8

END OF EXAMINATION