FIT2014 exam 2015 soln - sample exam, practice, mock exam, uni exam

Linear algebra and applications (Sunway University)

Faculty of Information Technology

Monash University

FIT2014 Theory of Computation
FINAL EXAM
SAMPLE SOLUTIONS

2nd Semester 2015

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 **(3 marks)**

You are choosing a main course at a restaurant, from a menu containing three items: bibimbap, haggis and manakish.

Suppose we have propositions B, H and M, with the following meanings.

B: You choose bibimbap.
H: You choose haggis.
M: You choose manakish.

Use B, H and M to write a proposition, in Conjunctive Normal Form, that is True precisely when you choose *either* the bibimbap *or* both the other items.

Allowing either *or both* of these two options:

$(B \vee H) \wedge (B \vee M)$

Allowing *exactly one* of these two options:

$(B \vee H) \wedge (B \vee M) \wedge (\overline{B} \vee \overline{H} \vee \overline{M})$

Question 2 **(5 marks)**

Suppose you have predicates `belongsToP` and `belongsToNP` with the following meanings, where variable X represents an arbitrary language:

`belongsToP`(X): the language X belongs to the class P.
`belongsToNP`(X): the language X belongs to the class NP.

(a) In the space below, write a statement in predicate logic with the meaning:

P is a proper subset of NP (i.e., P $\subseteq$ NP and P $\neq$ NP).

To do this, you may only use: the above two predicates; quantifiers; logical connectives; equality. (In particular, you may not use $\subseteq$ or $\subset$ or $\neq$, etc, and in fact they would not help.)

$(\forall X : \texttt{belongsToP}(X) \Rightarrow \texttt{belongsToNP}(X)) \wedge (\exists X : \texttt{belongsToNP}(X) \wedge \neg \texttt{belongsToP}(X))$

Some other equivalent correct answers:
$(\forall X\ \texttt{belongsToP}(X) \Rightarrow \texttt{belongsToNP}(X)) \wedge (\neg \forall X\ \neg \texttt{belongsToNP}(X) \vee \texttt{belongsToP}(X))$
$(\forall X\ \texttt{belongsToP}(X) \Rightarrow \texttt{belongsToNP}(X)) \wedge (\neg \forall X\ \texttt{belongsToNP}(X) \Rightarrow \texttt{belongsToP}(X))$

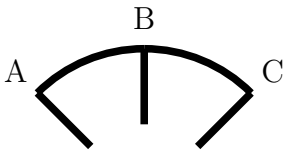
(b) For the statement "P is a proper subset of NP", which one of the following holds? Circle (A), (B), (C) or (D) to indicate your answer.

(A) The statement is True.
(B) The statement is False.
(C) It is not known whether it's True or False, but it's generally believed to be True.
(D) It is not known whether it's True or False, but it's generally believed to be False.

Official use only
8

Question 3 **(7 marks)**

A knob is used to select one of three positions A, B, C, as shown in the following diagram.



The position of the knob is recorded every second for some nonzero period of time.

The knob can never be turned from A to C, or from C to A, without spending at least one second in position B. The initial position of the knob can be any of A, B, C.

Consider the language of all possible strings of knob positions recorded in this way.

(a) Give a regular expression for this language.

If empty string forbidden:

$AA^* \cup CC^* \cup (A^* \cup C^*)B(A^* \cup C^*)(B(A^* \cup C^*))^*$

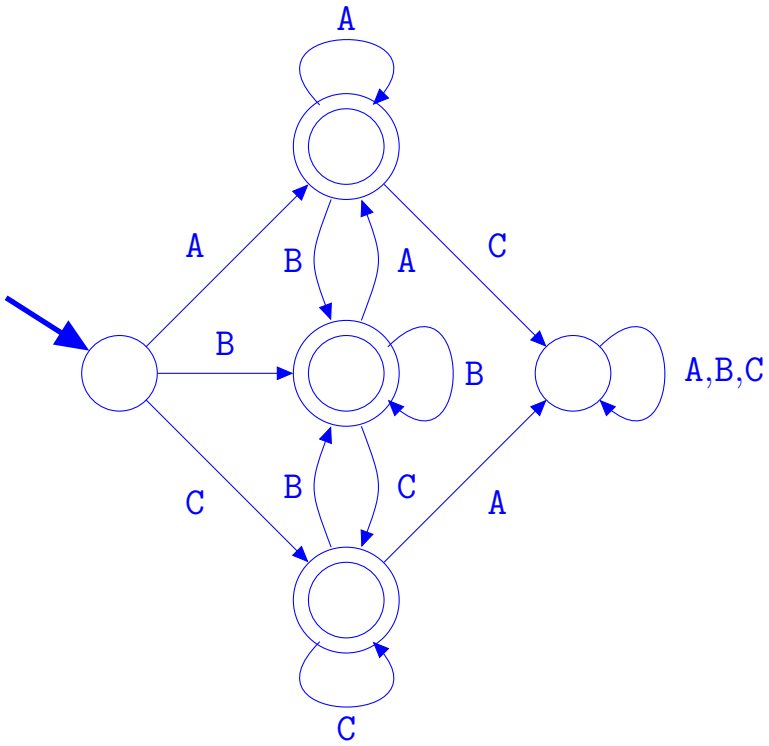
Many alternatives are possible, e.g.: $((AA^* \cup CC^*)(B(A^* \cup C^*))^*) \cup B(A^* \cup C^*)(B(A^* \cup C^*))^*$, ...

If empty string allowed:

$((A^* \cup C^*)B)^*(A^* \cup C^*)$

Many alternatives are possible, e.g.: $((A^*B)^*(C^*B)^*)^*(A^* \cup C^*)$, $(A^* \cup C^*)(BA^* \cup BC^*)^*$, ...

(b) Give a Finite Automaton for this language.

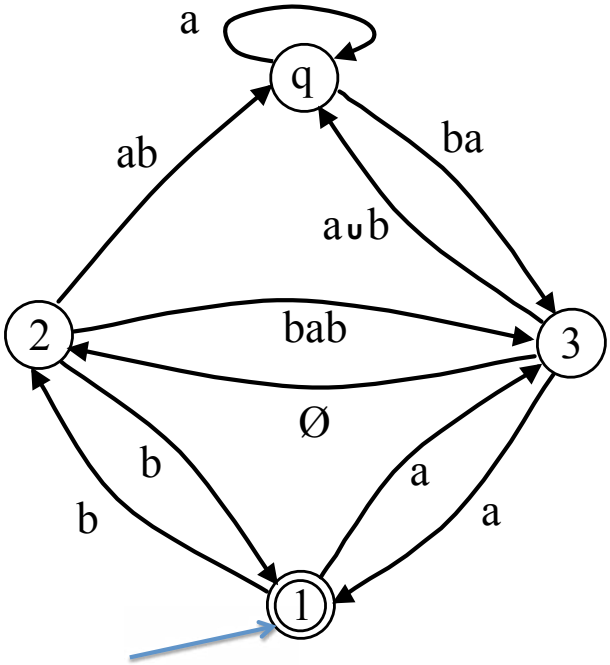


Official use only
7

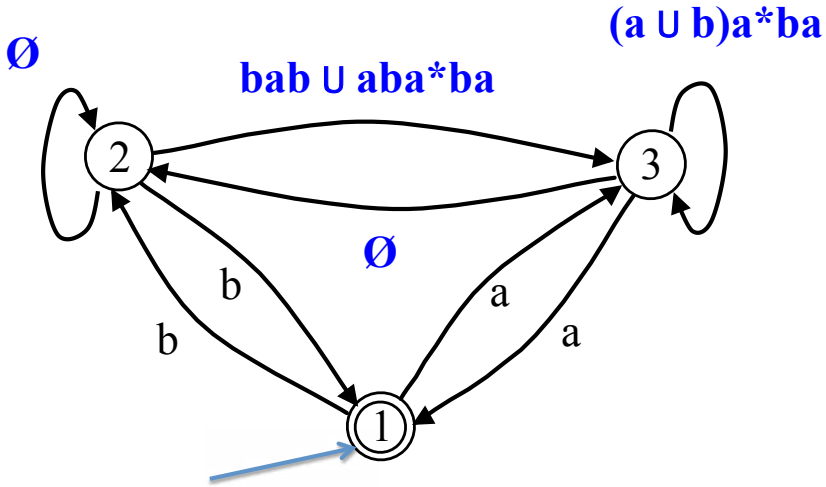
Question 4 **(5 marks)**

Consider the following Generalised Nondeterministic Finite Automaton (GNFA). Construct an equivalent GNFA with the top state, q, removed.

Show your answer by writing in the four missing transition labels, on the dotted lines, in the second diagram below.



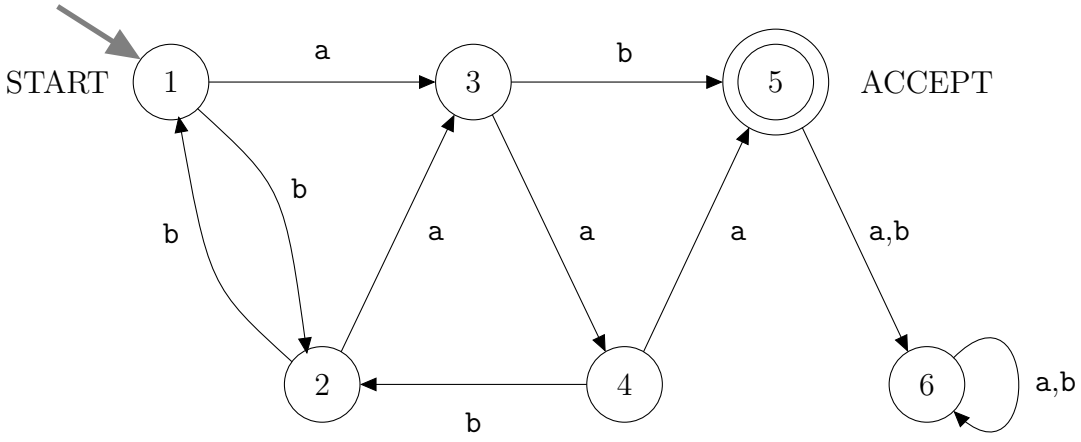
YOUR ANSWER:



Official use only
5

Question 5 **(6 marks)**

Consider the following Finite Automaton.



(a) Which, if any, of the following strings are accepted by this FA? Circle the ones that are accepted.

ab aba aabb aabab

(b) Consider the string aabab. Show how to split it up into three parts xyz such that, for all $i \geq 0$, the string xy^iz is also accepted by the FA.

Do this by drawing vertical lines between the parts of the string given on the next line:

a | a b a | b

This question is about the ideas behind the Pumping Lemma for Regular Languages.

Official use only
6

Question 6 **(4 marks)**

The following table describes part of a Finite Automaton with three states. The second row, for state 2, has not been filled in.

What could the state-2 row be, if you are given that the number of states of this FA is **not minimum**?

Write all possible state-2 rows in the second table below. Use as many rows as you need, one for each possible state-2 row. You may not need all the rows available.

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

YOUR ANSWER:

state	a	b
2	1	3
2	2	3
2		
2		

By allowing State 2 to be a Final State, some extra possibilities arise, shown in the third and fourth rows below.

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

Official use only
4

Question 7 **(6 marks)**

This question uses the following **Definitions** and **Facts**.

Definitions:

- If L is any language, the *reversal* of L is the set of all reversals of strings in L. In other words, you obtain the reversal of L by taking every string in L and writing it backwards. So, for example, the string `abb` becomes `bba`.
- If L is any language over the alphabet {`a`,`b`}, then the *interchange of* `a` *and* `b` forms a new language as follows: take every string in L, and replace every `a` by `b` and every `b` by `a`, simultaneously. So, for example, the string `abb` becomes `baa`.

Facts:

- The class of regular languages is closed under reversal.
- The class of regular languages is closed under interchange of `a` and `b`.
- The language AB := $\{\ \mathtt{a}^n\mathtt{b}^n\ :\ n \in \mathbb{N}\ \}$ is not regular.

Your task:

Using these facts, and any other closure properties of regular languages you like, *prove by contradiction* that the language

$$\{\ \mathtt{a}^m\mathtt{b}^n\ :\ m \leq n\ \}$$

is not regular.

Assume, by way of contradiction, that the language $\{\ \mathtt{a}^m\mathtt{b}^n\ :\ m \leq n\ \}$ is regular. Then its reversal, namely $\{\ \mathtt{b}^n\mathtt{a}^m\ :\ m \leq n\ \}$, is regular, since the class of regular languages is closed under reversal (using the first fact). Then we interchange `a` and `b`, giving the language $\{\ \mathtt{a}^n\mathtt{b}^m\ :\ m \leq n\ \}$, which must be regular, since the class of regular languages is closed under interchange of `a` and `b` (using the second fact). Take the intersection of the language we started with, and this last language. The former's `b`-part is at least as long as its `a`-part, while the latter's `a`-part is at least as long as its `b`-part. Their intersection consists of strings where the two parts have the same size:

$$\{\ \mathtt{a}^m\mathtt{b}^n\ :\ m \leq n\ \} \cap \{\ \mathtt{a}^n\mathtt{b}^m\ :\ m \leq n\ \} = \{\ \mathtt{a}^n\mathtt{b}^n\ :\ n \in \mathbb{N}\ \} = \text{AB},$$

which must be regular, since the class of regular languages is closed under intersection (as discussed in lectures). But we know that AB is not regular (third fact). So we have a contradiction. So the original assumption, that the language given in the question is regular, is wrong. So that language is not regular.

Question 8 **(3 marks)**

Explain how to derive, for any Finite Automaton, a regular grammar for the language recognised by the FA.

For each state, create a new non-terminal symbol to represent that state. The initial state is represented by the start symbol S.
For each transition, create a production rule as follows. If the transition has label x and goes from state P to state Q, the production rule is $P \to xQ$. For each final state, create a production rule whose right-hand side is the empty string. So, for a final state represented by non-terminal symbol P, we create the rule $P \to \varepsilon$.

Official use only
9

Question 9 **(12 marks)**

Consider the following Context-Free Grammar:

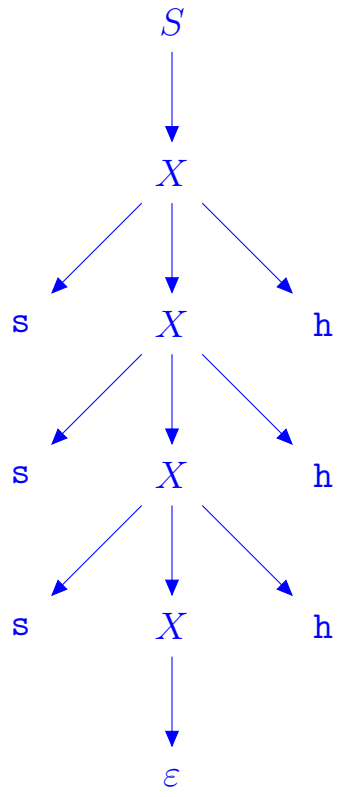
$$\begin{aligned} S &\rightarrow X && (1)\\ X &\rightarrow \texttt{s}X\texttt{h} && (2)\\ X &\rightarrow \varepsilon && (3) \end{aligned}$$

(a) Give a derivation for the string `ssshhh`.

Each step in your derivation must be labelled, on its right, by the number of the rule used.

$$\begin{aligned} S &\Rightarrow X && (1)\\ &\Rightarrow \texttt{s}X\texttt{h} && (2)\\ &\Rightarrow \texttt{ss}X\texttt{hh} && (2)\\ &\Rightarrow \texttt{sss}X\texttt{hhh} && (2)\\ &\Rightarrow \texttt{ssshhh} && (3) \end{aligned}$$

(b) Give a parse tree for the same string, `ssshhh`.



(c) Prove by induction on n, that for all $n \geq 0$, the string $\mathtt{s}^n\mathtt{h}^n$ has a derivation in this grammar of $n+2$ steps.

Inductive basis: when $n = 0$, the string is empty. The empty string can be derived from this grammar in two steps: $S \Rightarrow X \Rightarrow \varepsilon$, which is $n+2$ steps with $n = 0$. So the statement is true for $n = 0$.

Now suppose $n \geq 1$, and assume that the statement is true for $n-1$, i.e., any string $\mathtt{s}^{n-1}\mathtt{h}^{n-1}$ has a derivation of length $(n-1)+2 = n+1$ steps. (This is our Inductive Hypothesis.)

This derivation looks like

$$\underbrace{S \Rightarrow X \Rightarrow \cdots \Rightarrow \mathtt{s}^{n-1}\mathtt{h}^{n-1}}_{n+1 \text{ steps}}.$$

Observe here that the first rule *must* be rule (1), since that is the only one that uses S.

Now consider the string $\mathtt{s}^n\mathtt{h}^n$. Since $n \geq 1$, we can write this as:

$$\mathtt{s}^n\mathtt{h}^n = \mathtt{s}\mathtt{s}^{n-1}\mathtt{h}^{n-1}\mathtt{h}$$

Take the derivation $S \Rightarrow X \Rightarrow \cdots \Rightarrow \mathtt{s}^{n-1}\mathtt{h}^{n-1}$ given above, and do the following. First, omit S and the first production, at the very start. Then we have a derivation from X to $\mathtt{s}^{n-1}\mathtt{h}^{n-1}$ in n steps. Then, for each string in this derivation, put an extra $\mathtt{s}$ at the start and an extra $\mathtt{h}$ at the end. This gives us the partial derivation:

$$\underbrace{\mathtt{s}X\mathtt{h} \Rightarrow \cdots \Rightarrow \mathtt{s}\mathtt{s}^{n-1}\mathtt{h}^{n-1}\mathtt{h}}_{n \text{ steps}}$$

(The fact that it's still a valid derivation follows from the context-free property.)

We now prepend this derivation with the two-step derivation of $\mathtt{s}X\mathtt{h}$ from S, which is $S \Rightarrow X \Rightarrow \mathtt{s}X\mathtt{h}$, to give the $n+2$-step derivation of $\mathtt{s}^n\mathtt{h}^n$:

$$\underbrace{S \Rightarrow X \Rightarrow \mathtt{s}X\mathtt{h} \Rightarrow \cdots \Rightarrow \mathtt{s}\mathtt{s}^{n-1}\mathtt{h}^{n-1}\mathtt{h}}_{n+2 \text{ steps}} = \mathtt{s}^n\mathtt{h}^n.$$

This completes the inductive step.

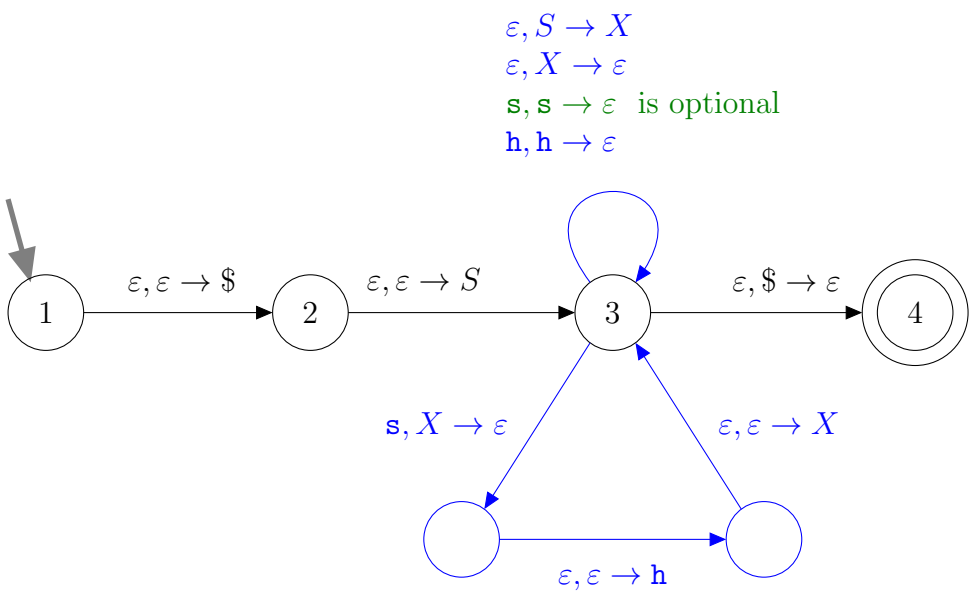
The statement is therefore true for all $n \geq 0$, by the Principle of Mathematical Induction.

Question 10 **(6 marks)**

This question uses the same Context-Free Grammar as the previous question. Here it is again for convenience:

$$S \rightarrow X \quad (1)$$
$$X \rightarrow \texttt{s}X\texttt{h} \quad (2)$$
$$X \rightarrow \varepsilon \quad (3)$$

Complete the following diagram to give a Pushdown Automaton for the language generated by this grammar.



Instead of the triangle below state 3, you could have a 2-cycle with two transitions $\texttt{s}, X \rightarrow \texttt{h}$ then $\varepsilon, \varepsilon \rightarrow X$. (This effectively combines the first two arcs of the triangle.)

Official use only
6

Question 11 **(5 marks)**

State (without proof) the Pumping Lemma for Context-Free Languages, and briefly describe its main purpose.

Let L be a Context-Free Language with k non-terminal symbols in Chomsky Normal Form. For any string w of length $\geq 2^{k-1}$, there is a partition of w into five substrings, $w = uvxyz$, such that:

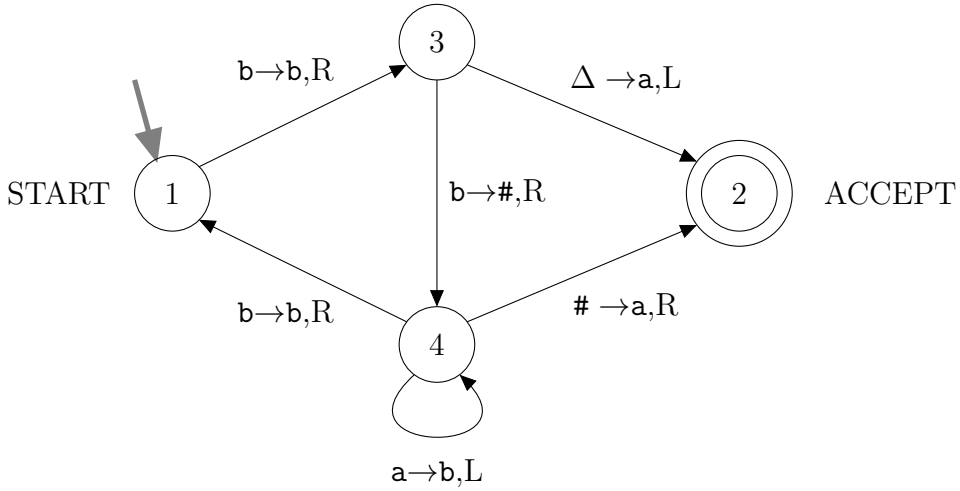
- at least one of v, y is nonempty,
- $|vxy| \leq 2^k$, and
- for all $i \geq 0$, the string uv^ixy^iz belongs to L.

Its main purpose is to help show that certain languages are *not* context-free.

Official use only
5

Question 12 **(6 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).

You should not need all the lines provided.

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

At start of step 1:	State: 1	Tape:	b	b	a	Δ	
At start of step 2:	State: 3	Tape:	b	b	a	Δ	
At start of step 3:	State: 4	Tape:	b	#	a	Δ	
At start of step 4:	State: 4	Tape:	b	#	b	Δ	
At start of step 5:	State: 2	Tape:	b	a	b	Δ	
At start of step 6:	State:	Tape:					
At start of step 7:	State:	Tape:					
At start of step 8:	State:	Tape:					
At start of step 9:	State:	Tape:					
At start of step 10:	State:	Tape:					
At start of step 11:	State:	Tape:					
At start of step 12:	State:	Tape:					

Official use only
6

Working Space

Question 13 **(4 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)	
Input: a Turing machine M. Question: Does there exist a string w that is accepted by M in at most 7 steps?	☑ Decidable	☐ Undecidable
Input: a Turing machine M. Question: Does there exist a string w that is accepted by M?	☐ Decidable	☑ Undecidable
Input: a string w. Question: Does there exist a Turing machine that accepts w?	☑ Decidable	☐ Undecidable
Input: a Turing machine M, and a string w. Question: Is w the encoding, in CWL, of M?	☑ Decidable	☐ Undecidable

Official use only
4

Question 14 **(10 marks)**

The Venn diagram on the right shows several classes of languages. For each language (a)–(j) 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 does not belong to any of these classes, then place its letter above the top of the diagram.

You may assume that, when Turing machines are encoded as strings, this is done using the Code-Word Language (CWL), with input alphabet {a,b} and tape alphabet {a,b,#,Δ}.

(a) The set of all palindromes of even length.

(b) The set of all positive integers, in binary, whose number of bits is odd.

(c) The set of all strings of correctly matched parentheses.

(d) The set of all encodings of Turing machines that have at least three states.

(e) The set of all encodings of Turing machines that have at most three states.

(f) The set of all encodings of Turing machines that halt for some input.

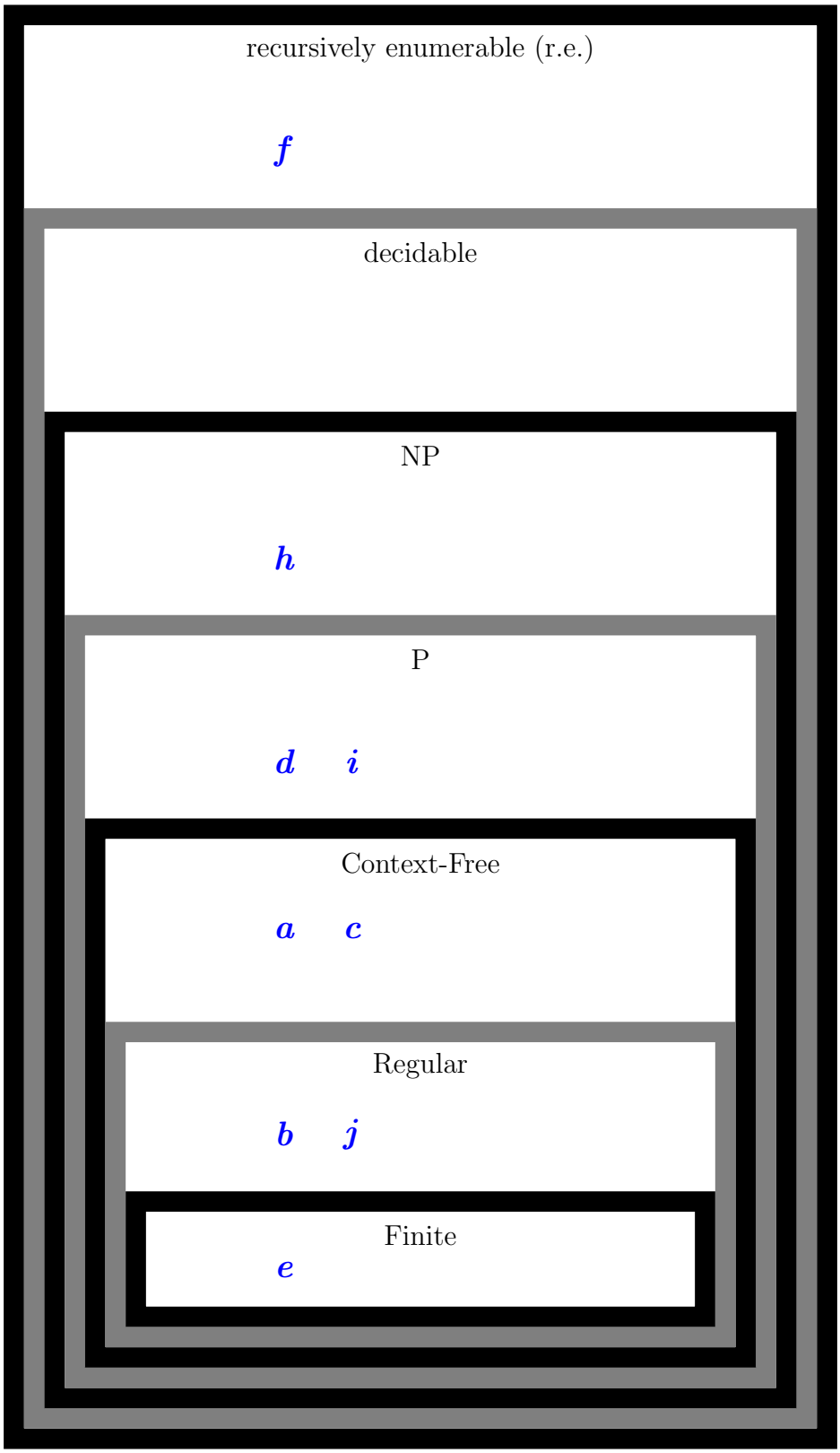
(g) The set of all encodings of Turing machines that loop forever for all inputs.

(h) The set of all satisfiable Boolean expressions.

(i) The set of all satisfiable Boolean expressions in Conjunctive Normal Form in which every variable appears at most once (so each variable has only one literal).

(j) The set of all *nondecreasing strings* of digits, i.e., strings over the digits 0,1,...,9 such that no digit is ever followed by a lower digit. (So 03348 is ok, but 03328 is not ok.)

g



Question 15 **(6 marks)**

If L is a language and s is a string, then $L\Delta s$ denotes the language formed by *removing* s from L if it is there already, and *adding* s to L otherwise. In other words,

$$L\Delta s := \begin{cases} L \setminus \{s\}, & \text{if } s \in L, \\ L \cup \{s\}, & \text{if } s \notin L. \end{cases}$$

Prove that L is decidable if and only if $L\Delta s$ is decidable.

Suppose L is decidable. Let D be a decider for L. We use it to build the following decider for $L\Delta s$.

> Input: string x.
> If $x = s$ then return the opposite answer to $D(s)$,
> else return $D(x)$.

This works because the only difference between the two languages is for the string s, on which they disagree; but any other string is either in both the languages or in neither of them.
Therefore $L\Delta s$ is decidable.
Exactly the same argument shows that, if $L\Delta s$ is decidable, then so is L. (In fact, if we now make D a decider for $L\Delta s$, then the above decider becomes a decider for L.) Therefore L is decidable if and only if $L\Delta s$ is decidable.

Official use only
6

Question 16 **(5 marks)**

Below is a proof that the Halting Problem is undecidable. But some parts are missing; these are shown as blank spaces, underlined.

Your task is to fill in the underlined spaces to complete the proof. In some cases, options are given in square brackets.

Proof. Assume that the Halting Problem is actually decidable.
Then the Halting Problem has a decider D.
Using D, we can construct a Turing machine E that does the following:

1. Input: encoding of a Turing machine M.
2. Run decider D to determine whether or not M halts when given itself as input.
3. **IF** the answer from D is **Yes**, then

   loop forever

   ---

4. **IF** the answer from D is **No**, then

   stop

Now consider what happens if E is given, as input, an encoding of *itself*.

If E halts on input E, then running D in line 2 gives the answer __Yes__ [Yes/No].

So, using statement __**3**__ [3 or 4], we see that D actually _____ loops forever _____.

On the other hand,

if E does not halt on input E, then running D in line 2 gives the answer __No__ [Yes/No].

So, using statement __**4**__ [3 or 4], we see that D actually _____ stops _____.

This is a contradiction.

So the Halting Problem must be undecidable.

Official use only
5

Question 17 **(6 marks)**

Suppose you have an enumerator M_1 for a language L and another enumerator M_2 for its complement $\overline{L}$.

(a) Explain how to construct a decider for L that uses the enumerators M_1 and M_2.

Decider for L:

Input: string x.
Run M_1 and M_2 in parallel. (This can be done by a loop which keeps track of the entire configuration of both machines and, in each iteration, simulates one step of each machine.) Whenever either machine outputs a string, we check if that string equals x.
If we see x as an output string from M_1, then we know $x \in L$, and we stop and output Yes.
If we see x as an output string from M_2, then we know $x \in \overline{L}$, and we stop and output No.

We must eventually see x as an output string from *exactly one* of the two machines (not both, not neither), since one machine enumerates L and the other enumerates its complement, namely $\overline{L}$. (Every string belongs either to L or to $\overline{L}$.) So the above computation must always halt eventually.

Other algorithms are possible. E.g., for each $i \in \mathbb{N}$, simulate the first i steps of M_1 and then the first i steps of M_2. If either produces x, then output Yes or No according as x was produced by M_1 or M_2. If neither produces x, continue.

(b) What does this tell you about recursively enumerable (r.e.) languages whose complements are also recursively enumerable? (Only one short sentence is required here.)

Such languages must be decidable.

Official use only
6

Working Space

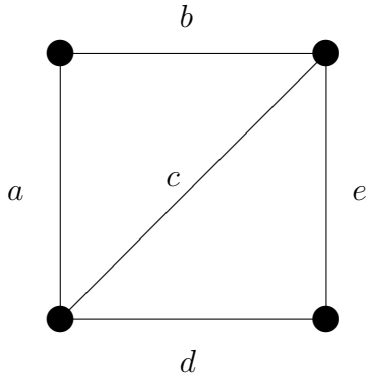
Question 18 **(13 marks)**

A **perfect matching** in a graph G is a subset X of the edge set of G that meets each vertex exactly once. In other words, no two edges in X share a vertex, and each vertex of G is incident with exactly one edge in X.

The PERFECT MATCHING decision problem is as follows.

PERFECT MATCHING
Input: Graph G.
Question: Does G have a perfect matching?

For example, in the following graph, the edge set $\{a, e\}$ is a perfect matching. But $\{a, b, e\}$ is not a perfect matching (since, for example, a and b share a vertex), and $\{a\}$ is not a perfect matching (since some vertices are not incident with the edge in this set).



Let W be the above graph.

(a) Construct a Boolean expression E_W in Conjunctive Normal Form such that the satisfying truth assignments for E_W correspond to perfect matchings in the above graph W.

$$(a \vee b) \wedge (\neg a \vee \neg b) \wedge$$
$$(a \vee c \vee d) \wedge (\neg a \vee \neg c) \wedge (\neg a \vee \neg d) \wedge (\neg c \vee \neg d) \wedge$$
$$(b \vee c \vee e) \wedge (\neg b \vee \neg c) \wedge (\neg b \vee \neg e) \wedge (\neg c \vee \neg e) \wedge$$
$$(d \vee e) \wedge (\neg d \vee \neg e).$$

(b) Give a polynomial-time reduction from PERFECT MATCHING to SATISFIABILITY.

Input: graph G.
For each vertex v:
{
    Let the edges incident with v be $e_1, e_2, \ldots, e_d$.
        We need to create clauses which, when AND-ed together, say:
        Exactly one of these edges is in the matching.
    Create a new clause, $e_1 \vee \cdots \vee e_d$.
        This says: at least one of $e_1, \ldots, e_d$ is in the matching.
    For each pair e_i, e_j of edges incident with v:
        Create a new clause, $\neg e_i \vee \neg e_j$.
            This says: at least one of e_i, e_j is *not* in the matching.
}
Combine all these clauses using $\wedge$, to create the conjunction of all of them.
Output the resulting expression.

Official use only
13

Question 19 **(8 marks)**

Prove that the GRAPH 4-COLOURABILITY problem is NP-complete, by reduction from GRAPH 3-COLOURABILITY. You may assume that GRAPH 3-COLOURABILITY is NP-complete.

Definitions:

For any positive integer k, a **k-colouring** in a graph G is an assignment of "colours" from the set $\{1, 2, \ldots, k\}$ to the vertices of G such that (a) each vertex gets exactly one colour from the set, and (b) adjacent vertices get different colours.

GRAPH 3-COLOURABILITY
Input: Graph G.
Question: Does G have a 3-colouring?

GRAPH 4-COLOURABILITY
Input: Graph G.
Question: Does G have a 4-colouring?

GRAPH 4-COLOURABILITY is in NP, since it has a polynomial-time verifier: the certificate is a 4-colouring, and this can be checked in polynomial time: for each edge, check that its endpoints are differently coloured, and check that the total number of colours used is at most 4.

We now prove that GRAPH 3-COLOURABILITY is polynomial-time reducible to GRAPH 4-COLOURABILITY. The reduction works as follows.

> Input: graph G.
> Create a new vertex v, and join it to every vertex of G. Denote the resulting graph by $G + v$.
> Output $G + v$.

This reduction is polynomial-time computable, since we just have to add one new vertex, and the number of new edges is at most the number of vertices in G.

If G is 3-colourable, then we can give a new colour to v, different to the three colours used on G. This gives a 4-colouring of $G + v$, since each new edge is properly coloured. (For each new edge, one of its endpoints is the new colour, which cannot appear on the other endpoint.) So we have shown that, if G is 3-colourable, then $G + v$ is 4-colourable.

Now suppose $G + v$ has a 4-colouring. We will show that G is 3-colourable.
Consider v. Since it is adjacent to every vertex of G, the colour given to v (in the 4-colouring) cannot appear on any vertex of G. Once that colour is excluded, there are only three colours left, so we must have a 3-colouring of G. So we have shown that, if $G + v$ is 4-colourable,

then G is 3-colourable.

We therefore have: G is 3-colourable if and only if $G + v$ is 4-colourable. This, together with its polynomial-time computability, shows that the reduction given above is a polynomial-time reduction from GRAPH 3-COLOURABILITY to GRAPH 4-COLOURABILITY. Since GRAPH 3-COLOURABILITY is known to be NP-complete, and since we showed at the start that GRAPH 4-COLOURABILITY is in NP, we conclude that GRAPH 4-COLOURABILITY is NP-complete.

Official use only
8

END OF EXAMINATION