FIT2014 2020 Mock EExam soln

Linear algebra and applications (Sunway University)

Faculty of Information Technology

Monash University

FIT2014 Theory of Computation
Mock e-Exam
SAMPLE SOLUTIONS

2nd Semester 2020

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)**

Suppose we have propositions B, C and T, with the following meanings.

B: Charles Babbage was the first computer scientist.
C: Alonzo Church was the first computer scientist.
T: Alan Turing was the first computer scientist.

Use B, C and T to write a proposition that is True if and only if the first computer scientist was *one, and only one,* of Charles Babbage, Alonzo Church and Alan Turing.

For full marks, your proposition should be in Conjunctive Normal Form.

$(B \vee C \vee T) \wedge (\neg B \vee \neg C) \wedge (\neg B \vee \neg T) \wedge (\neg C \vee \neg T)$

Question 2 **(4 marks)**

Suppose you have predicates `automatic`, `generalPurpose`, `programmable` and `storedProgram` with the following meanings, where variable X represents an arbitrary device:

`automatic`(X):	the device X does not require human intervention, once its instructions are entered.
`computer`(X):	the device X is a computer.
`generalPurpose`(X):	the device X can compute any computable function.
`programmable`(X):	the device X can be programmed.
`storedProgram`(X):	the device X stores its instructions in its memory, in order to execute them.

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

A device is a computer if and only if it is automatic, programmable, stored program, and can compute any computable function.

To do this, you may only use: the above five predicates; quantifiers; logical connectives. (In particular, you may not use set theory symbols such as $\subseteq$, $\subset$, $\cap$, $\cup$, etc, and in fact they would not help.)

$\forall X :\ \text{computer}(X) \iff \text{automatic}(X) \wedge \text{programmable}(X) \wedge \text{storedProgram}(X) \wedge \text{generalPurpose}(X)$

Acceptable alternatives include:

$\forall X :\ (\text{computer}(X) \impliedby \text{automatic}(X) \wedge \text{programmable}(X) \wedge \text{storedProgram}(X) \wedge \text{generalPurpose}(X))$
$\wedge\ (\text{computer}(X) \implies \text{automatic}(X) \wedge \text{programmable}(X) \wedge \text{storedProgram}(X) \wedge \text{generalPurpose}(X))$

(b) What type of Turing machine best models the idea of a computer, according to the above definition?

A universal Turing machine.

Official use only
8

Question 3 **(6 marks)**

A traffic light displays a sequence of colours Red (R), Green (G), and Amber (A), in that order, then going back to Red and repeating the cycle again, and so on, for some period of time. The first colour displayed is always Red, but it can finish at any time, on any colour.

Consider the language of all finite nonempty strings over the alphabet {R,G,A} that represent a valid sequence of traffic light colours. No two consecutive letters can be the same. For example, the language includes the strings R, RG, RGA, RGAR, RGARG, RGARGA, ...

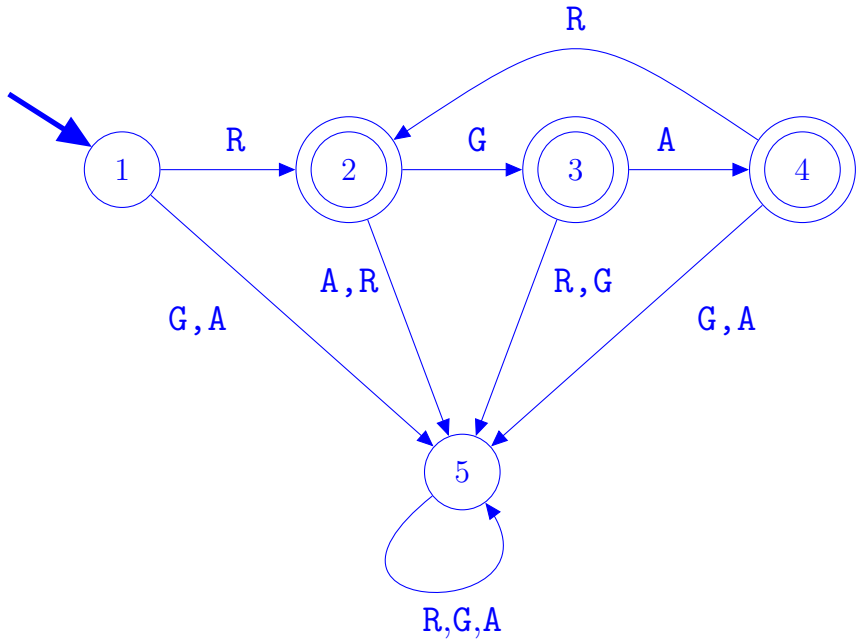
(a) Give a regular expression for this language.

$$\mathtt{R}(\mathtt{GAR})^*(\mathtt{GA} \cup \mathtt{G} \cup \varepsilon)$$

Alternative answer (longer, but still correct):

$$\mathtt{R}\left((\mathtt{GAR})^* \cup \mathtt{G}(\mathtt{ARG})^* \cup \mathtt{GA}(\mathtt{RGA})^*\right)$$

(b) Give a Finite Automaton for this language.

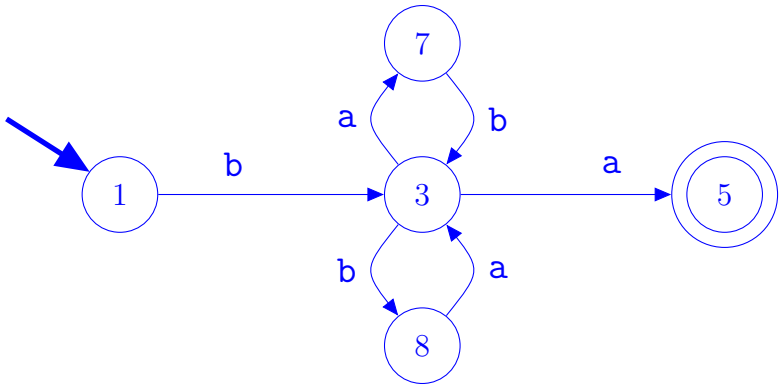


Official use only
6

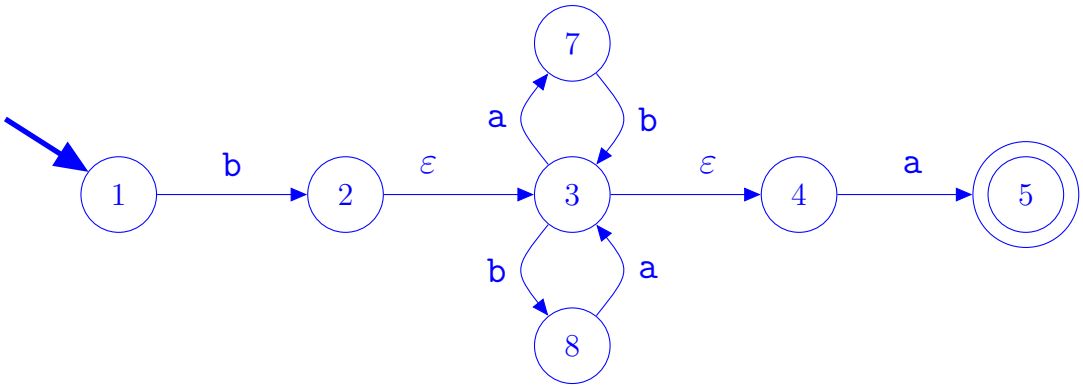
Question 4 **(5 marks)**

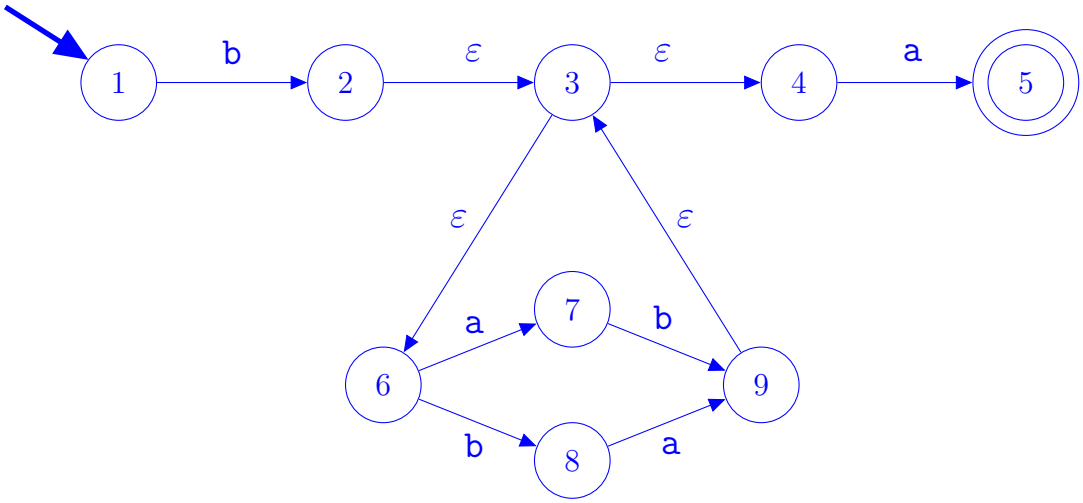
Draw a Nondeterministic Finite Automaton (NFA) that recognises the language of strings that match the following regular expression:

$$\mathsf{b}(\mathsf{ab} \cup \mathsf{ba})^*\mathsf{a}$$



Some alternatives, less elegant but still correct:

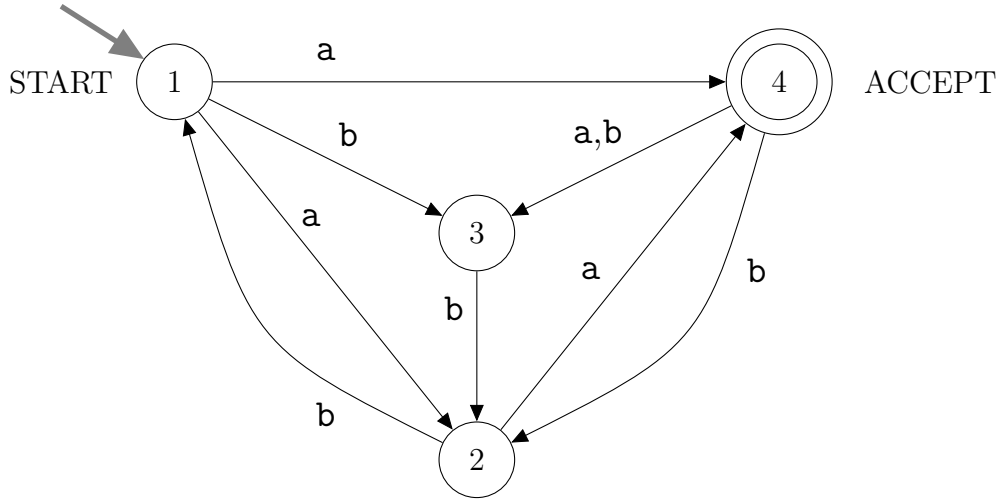




Official use only
5

Question 5 **(7 marks)**

Consider the following Nondeterministic Finite Automaton (NFA).



Let L be the language of strings accepted by this NFA.

(a) What are the possible states that this NFA could be in, after reading the input string `abba`?

2, 4

(b) Prove, by induction on n, that for all positive integers n, the string $(\texttt{abba})^n$ is accepted by this NFA. (This string is obtained by n repetitions of `abba`.)

Base case ($n = 1$): We saw in (a) that `abba` can end up in State 4, which is the Final state. Therefore `abba` is accepted.

Inductive step: suppose $n \geq 2$, and that $(\texttt{abba})^{n-1}$ is accepted.
Since $(\texttt{abba})^{n-1}$ is accepted (by Inductive Hypothesis), there is some path it can take through the NFA that ends at the Final State. We also see that, if we are in the Final State, and we then read `abba`, then we can reach the Final State again, by following the path $4 \to 3 \to 2 \to 1 \to 4$. We can put these two paths together, to give a path for the string $(\texttt{abba})^{n-1}\texttt{abba}$ that goes from the Initial State to the Final State. Therefore this string is accepted by our NFA. But this string is just $(\texttt{abba})^n$. So $(\texttt{abba})^n$ is accepted.

Therefore, by the Principle of Mathematical Induction, the string $(\texttt{abba})^n$ is accepted for all $n \geq 1$.

Official use only
7

Question 6 **(4 marks)**

Consider the five-state Finite Automaton represented by the following table.

	state	a	b
Start	1	5	5
	2	3	5
	3	4	5
	4	2	5
Final	5	2	3

Convert this into an equivalent FA with the minimum possible number of states.

Write your answer in the following table. You may not need all the rows available.

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

Official use only
4

Working Space

Question 7 **(9 marks)**

The language DOG consists of all strings of the form

$$\texttt{gr}^n(\texttt{woof})^n$$

where n is any positive integer. For example, the strings `grwoof` and `grrwoofwoof` both belong to DOG, but `grrwoof` does not.

(a) Use the Pumping Lemma for Regular Languages to prove that DOG is not regular.

Assume, by way of contradition, that DOG is regular.
Then there is a FA that recognises it. Let N be the number of states in such an FA.
Let w be the string $\texttt{gr}^N(\texttt{woof})^N$.
By the Pumping Lemma, w can be divided up into three parts, $w = xyz$, such that y is nonempty, $|xy| \leq N$, and $xy^iz \in$ DOG for all $i \geq 0$.
The requirement that $|xy| \leq N$ forces y to fall within the first part, $\texttt{gr}^N$, of w.
Consider the string $xyyz$. If y contains `g`, then $xyyz$ has two `g`s, so $xyyz \notin$ DOG, since every string in DOG has exactly one `g`.
If y contains no `g`, then it's all-`r`, so repetition of `y` creates at least one extra `r` (since y is nonempty), so the number of `r`s is greater than the number of `woof`s, which violates the definition of DOG. So $xyyz \notin$ DOG, which contradicts the conclusion of the Pumping Lemma.
So our initial assumption, that DOG is regular, must be incorrect.
So DOG is not regular.

Other choices of w are possible. You could let w be any string of the form $\texttt{gr}^n(\texttt{woof})^n$ whose total length is $> N$. But then you'd have to have three cases in the proof, according to whether y lies within $\texttt{gr}^n$, or within $(\texttt{woof})^N$, or contains some of each (i.e., it straddles the boundary between the two parts). This is ok if done correctly.

Instead of the string $xyyz$, we could have chosen xy^iz for any $i \geq 2$. The argument would be almost identical. We also could have chosen xz (i.e., $i = 0$). In that case, the argument is slightly different (but not much): if y includes `g`, then xz contains *no* `g`, which violates the requirement that every string in DOG has exactly one `g`; if it does not contain `g`, then it's all-`r`, so the string xz has *fewer* `r`s than `woof`s, which violates the defintion of DOG.

(b) Give a Context-Free Grammar for DOG.

$$\begin{aligned} S &\to \texttt{g}X \\ X &\to \texttt{r}X\texttt{woof} \\ X &\to \texttt{rwoof} \end{aligned}$$

Alternative answer:

$$\begin{aligned} S &\to \texttt{gr}X\texttt{woof} \\ X &\to \texttt{r}X\texttt{woof} \\ X &\to \varepsilon \end{aligned}$$

The following answer is *almost* correct, but it allows the string `g` which is not in the language (since n must be a *positive* integer).

$$\begin{aligned} S &\to \texttt{g}X \\ X &\to \texttt{r}X\texttt{woof} \\ X &\to \varepsilon \end{aligned}$$

The following answer is *incorrect*, since by using S everywhere it allows `g` to appear throughout the generated string.

$$\begin{aligned} S &\to \texttt{g}S \\ S &\to \texttt{r}S\texttt{woof} \\ S &\to \varepsilon \end{aligned}$$

Official use only
9

Question 8 **(13 marks)**

Consider the following Context-Free Grammar:

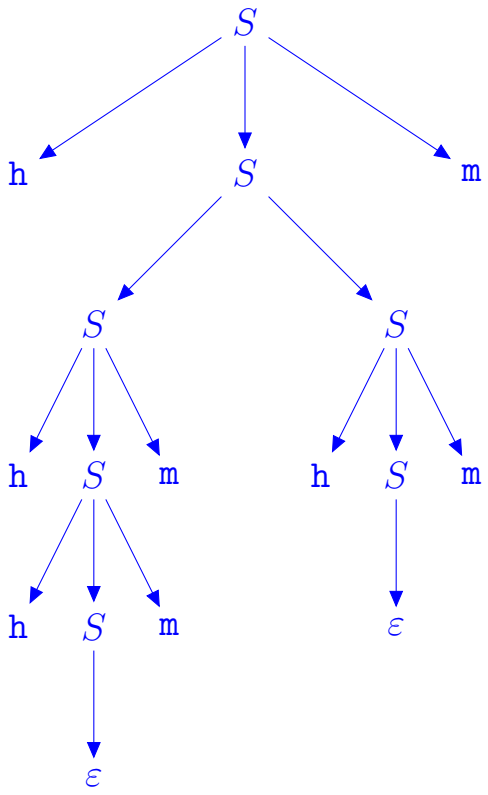
$$S \rightarrow SS \quad (1)$$
$$S \rightarrow \texttt{h}S\texttt{m} \quad (2)$$
$$S \rightarrow \varepsilon \quad (3)$$

(a) Give a derivation for the string `hhhmmhmm`.
Each step in your derivation must be labelled, on its right, by the number of the rule used.

$$\begin{aligned} S &\Rightarrow \texttt{h}S\texttt{m} && (2)\\ &\Rightarrow \texttt{h}SS\texttt{m} && (1)\\ &\Rightarrow \texttt{hh}S\texttt{m}S\texttt{m} && (2)\\ &\Rightarrow \texttt{hhh}S\texttt{mm}S\texttt{m} && (2)\\ &\Rightarrow \texttt{hhh}S\texttt{mmh}S\texttt{mm} && (2)\\ &\Rightarrow \texttt{hhhmmh}S\texttt{mm} && (3)\\ &\Rightarrow \texttt{hhhmmhmm} && (3) \end{aligned}$$

The first two productions must be as given, but some variation of order of the remaining productions is possible. The last production will always use (3), and there will be one other use of that production rule. Production rule (1) is only used once, in the second production.

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



(c) Prove by induction on n, that for all $n \geq 0$, the string $\mathtt{h}^n\mathtt{m}^n$ has a derivation in this grammar of $n+1$ steps.

Inductive basis: when $n = 0$, the string is empty. The empty string can be derived from this grammar in one step: $S \Rightarrow \varepsilon$, which is $n+1$ 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 $\mathtt{h}^{n-1}\mathtt{m}^{n-1}$ has a derivation of length $(n-1)+1 = n$ steps. (This is our Inductive Hypothesis.)

This derivation looks like

$$\underbrace{S \Longrightarrow \cdots \Longrightarrow \mathtt{h}^{n-1}S\mathtt{m}^{n-1} \overset{(3)}{\Longrightarrow} \mathtt{h}^{n-1}\mathtt{m}^{n-1}}_{n \text{ steps}}.$$

Observe here that the last rule *must* be rule (3), since that is the only one without any non-terminal symbols on its right-hand side. So the second-last string must have just a single S. Furthermore, it can be seen from the rules that the only terminal symbol that can appear to the *left* of an S is $\mathtt{h}$, and the only terminal that can appear to the *right* of an S is $\mathtt{m}$. From these observations it follows that the solitary S in the second-last string above must lie exactly in the middle, with an $\mathtt{h}$ on its left and an $\mathtt{m}$ on its right.

Take the derivation $S \Longrightarrow \cdots \Longrightarrow \mathtt{h}^{n-1}S\mathtt{m}^{n-1} \overset{(3)}{\Longrightarrow} \mathtt{h}^{n-1}\mathtt{m}^{n-1}$ given above, and do the following. Instead of applying the rule (3) to S in the string $\mathtt{h}^{n-1}S\mathtt{m}^{n-1}$, we apply the rule (1). Then our productions take us from S to $\mathtt{h}^{n-1}\mathtt{h}S\mathtt{m}\mathtt{m}^{n-1}$ in n steps. This string is just $\mathtt{h}^nS\mathtt{m}^n$. Then apply rule (1). This gives us the derivation:

$$\underbrace{S \Longrightarrow \cdots \Longrightarrow \mathtt{h}^{n-1}S\mathtt{m}^{n-1} \overset{(3)}{\Longrightarrow} \mathtt{h}^nS\mathtt{m}^n \overset{(1)}{\Longrightarrow} \mathtt{h}^n\mathtt{m}^n}_{n+1 \text{ steps}}.$$

This completes the inductive step.

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

An alternative approach to the inductive step is to take the n-step derivation

$$S \Longrightarrow \cdots \Longrightarrow \mathtt{h}^{n-1}\mathtt{m}^{n-1},$$

given by the Inductive Hypothesis, and then put $\mathtt{h}$ in front of every string and $\mathtt{m}$ behind every string. This gives the n-step partial derivation

$$\mathtt{h}S\mathtt{m} \Longrightarrow \cdots \Longrightarrow \mathtt{h}\mathtt{h}^{n-1}\mathtt{m}^{n-1}\mathtt{m} = \mathtt{h}^n\mathtt{m}^n.$$

Then put the production $S \overset{(2)}{\Longrightarrow} \mathtt{h}S\mathtt{m}$ at the start of this partial derivation, giving the $n+1$-step derivation

$$S \overset{(2)}{\Longrightarrow} \mathtt{h}S\mathtt{m} \Longrightarrow \cdots \Longrightarrow \mathtt{h}^n\mathtt{m}^n.$$

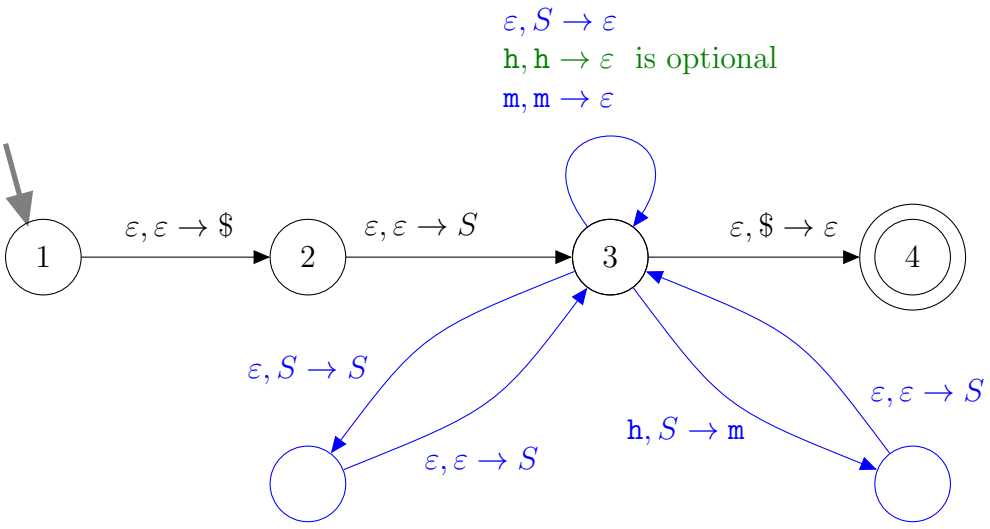
Official use only
13

Question 9 **(6 marks)**

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

$$S \rightarrow SS \quad (1)$$
$$S \rightarrow \texttt{h}S\texttt{m} \quad (2)$$
$$S \rightarrow \varepsilon \quad (3)$$

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



Instead of the 2-cycle at lower right of state 3, you could have a triangle with three transitions $\texttt{h}, S \rightarrow \varepsilon$ then $\varepsilon, \varepsilon \rightarrow \texttt{m}$ then $\varepsilon, \varepsilon \rightarrow S$. (This effectively turns the first arc of the 2-cycle into the first two arcs of the triangle.)

Official use only
6

Question 10 **(6 marks)**

This question is about doing the *last part* of the Cocke-Younger-Kasami (CYK) algorithm, to determine whether the string `abab` can be generated by the following grammar.

$$\begin{aligned} S &\rightarrow AA \mid AB \\ A &\rightarrow BB \mid \texttt{a} \\ B &\rightarrow AB \mid BA \mid \texttt{b} \end{aligned}$$

Suppose you have dealt with all substrings of length ≤ 3, and that, for each of these strings, you have worked out all the nonterminals that can generate it. This information is summarised in the following table.

string	Nonterminals that can generate the string
`a`	A
`b`	B
`ab`	S, B
`ba`	B
`aba`	S, B
`bab`	A

(a) Determine all *pairs* of nonterminals that can generate the string `abab`.

AA, SS, SB, BS, BB

(b) Determine all *single* nonterminals that can generate this same string, `abab`.

S, A

(c) State whether or not `abab` can be generated by the above grammar, by circling the appropriate word YES or NO below:

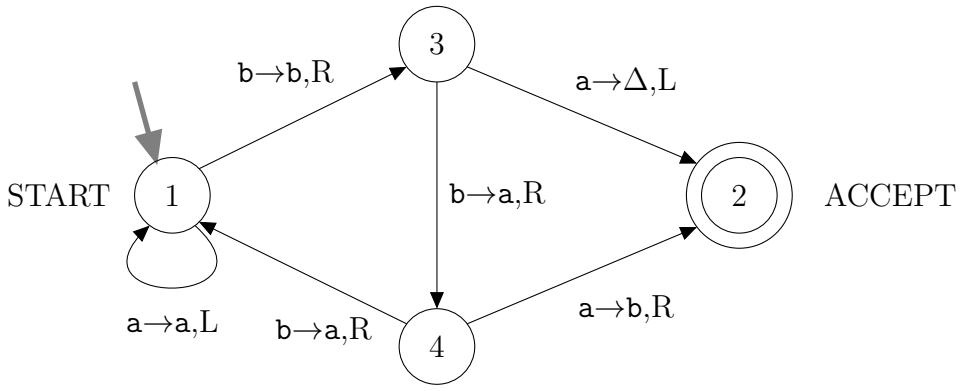
(YES), `abab` does belong to this language.

NO, `abab` does **not** belong to this language.

Official use only
6

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

Official use only
8

Working Space

Question 12 **(3 marks)**

State the Church-Turing thesis, and give two reasons why it is believed to be true.

Any function which can defined by an algorithm can be represented by a Turing Machine.

The reasons can be any two of:

- 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 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 with at most ten states. Question: Does M eventually halt, when given itself as input?	☑ Decidable	☐ Undecidable
Input: a Turing machine M with at most ten transitions. Question: Does M eventually halt, when given itself as input?	☑ Decidable	☐ Undecidable
Input: a Turing machine M with a tape alphabet of at most ten letters. Question: Does M eventually halt, when given itself as input?	☐ Decidable	☑ Undecidable
Input: a Turing machine M that can only move to the Right. Question: Does M eventually halt, when given itself as input?	☑ Decidable	☐ Undecidable

Official use only
7

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 empty language.

(b) The set of all strings in which a and b appear the same number of times.

(c) The set of all adjacency matrices of graphs.
(Such a matrix is represented as a string of n^2 bits, where n is the number of vertices.)

(d) The set of adjacency matrices of 3-colourable graphs.

(e) The set of all Boolean expressions that are not satisfiable.

(f) The set of all encodings of Turing machines.

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

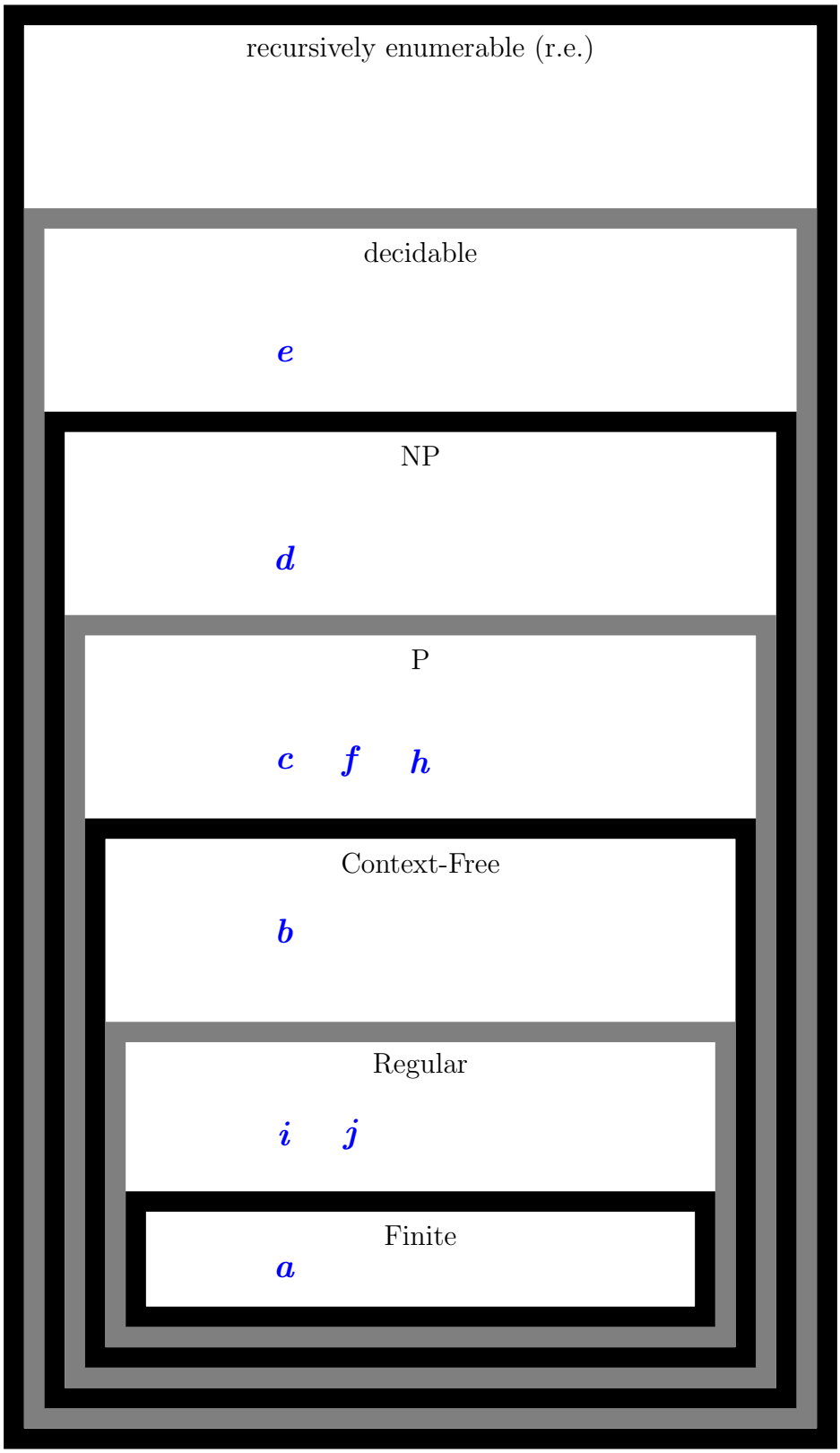
(h) The set of strings of the form $\mathtt{x}^n\mathtt{y}^{2n}\mathtt{z}^{3n}$, where n is any positive integer.

(i) The set of legal positions in One-Dimensional Go.

(j) The set of illegal positions in One-Dimensional Go.

Reminder: a legal position in 1D-Go is a string over the alphabet {b,w,u} such that every letter b is part of a string of consecutive bs with u at one end (or both ends), and every letter w is part of a string of consecutive ws with u at one end (or both ends). An illegal position in 1D-Go is a string over the same alphabet that is not a legal position.

g



Official use only

10

Question 15 **(6 marks)**

Prove by contradiction that, if L_1 is undecidable, $L_2 \subseteq L_1$, and L_2 is decidable, then $L_1 \setminus L_2$ is undecidable.

(Here, $L_1 \setminus L_2$ is the set of strings that belong to L_1 but not L_2.)

Let L_1 and L_2 be languages such that L_1 is undecidable, $L_2 \subseteq L_1$, and L_2 is decidable.
Let C be a decider for L_2.
Assume, by way of contradiction, that $L_1 \setminus L_2$ is decidable. Let D be a decider for it.
Observe that $L_1 = L_2 \cup (L_1 \setminus L_2)$.
Then we can form a decider for L_1 as follows:

> Input: x
> Use C to decide if $x \in L_2$.
> Use D to decide if $x \in L_1 \setminus L_2$.
> If one of C or D answers Yes, then output Yes, since we then know that $x \in L_1$.
> If both of C and D answer No, then output No, since we then know that $x \notin L_1$.

This shows that L_1 is decidable.
This is a contradiction, since we are given that L_1 is undecidable.
So our assumption, that $L_1 \setminus L_2$ is decidable, was incorrect.
Therefore $L_1 \setminus L_2$ is undecidable.

Official use only
6

Question 16 **(5 marks)**

Suppose you have an enumerator M for a language L.

Give an algorithm that accepts precisely the strings in L.

Your algorithm may use M.
For strings not in L, the algorithm must either reject or loop forever.

Input: x
Loop:
{
    Run M until it generates another string.
    If that string equals x, then Accept.
}

Official use only
5

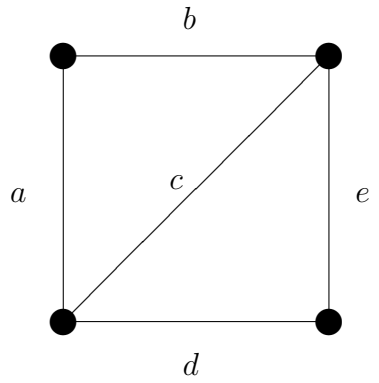
Question 17 **(12 marks)**

An **edge cover** in a graph G is a set X of edges that meets every vertex of G. So, every vertex is incident with at least one edge in X.

The EDGE COVER decision problem is as follows.

EDGE COVER
Input: Graph G.
Question: Does G have an edge cover?

For example, in the following graph, the edge set $\{a, b, c, d\}$ is an edge cover, and so is $\{a, e\}$. But $\{a, b, c\}$ is not an edge cover, since it does not meet every vertex. (Specifically, it misses the bottom right vertex.)



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 edge covers in the above graph W.

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

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

Input: graph G.
For each vertex v:
{
    Let the edges incident with v be $e_1, e_2, \ldots, e_d$.
    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 edge cover.
}
Combine all these clauses using $\wedge$, to create the conjunction of all of them.
Output the resulting expression.

Official use only
12

Question 18 **(8 marks)**

Prove that the problem BIG INDEPENDENT SET problem is NP-complete, by reduction from HUGE INDEPENDENT SET. You may assume that HUGE INDEPENDENT SET is NP-complete.

Definitions:

For any positive integer k, an **independent set** in a graph G is a subset X of the vertex set of G such that no two vertices in X are adjacent.

HUGE INDEPENDENT SET
Input: Graph G, with an even number of vertices.
Question: Does G have an independent set of size $\geq n/\mathbf{2}$?

BIG INDEPENDENT SET
Input: Graph G.
Question: Does G have an independent set of size $\geq n/\mathbf{3}$?

In each of these definitions, n is the number of vertices in the graph G.

BIG INDEPENDENT SET is in NP, since it has a polynomial-time verifier as follows:
Input: graph G
Certificate: a set of vertices of G
Check that the number of vertices in the certificate is $\geq n/3$.
For each pair of vertices in the certificate, check they are not adjacent.
If all these checks pass, then Accept, otherwise Reject.
The work required in checking is counting the number of vertices in the certificate, doing one division and one comparision, and iterating over all pairs of vertices in the certificate. The number of pairs of vertices in the certificate is $O(n^2)$. So the certificate can be verified in polynomial time.

Reduction from HUGE INDEPENDENT SET to BIG INDEPENDENT SET:

Let G be a graph with an even number of vertices (called n).
Add $n/2$ new vertices to G, and join them all up to each other as well as to every vertex of G. Let H be the new graph so formed, and let p be the number of vertices of H. Then $p = n + (n/2) = 3n/2$.
The largest independent set of H is the same as the largest independent set of G, since the new vertices in H can never be added to any independent set in G. So,

$$\begin{aligned} G \ &\in \ \text{HUGE INDEPENDENT SET} \\ &\iff \ G \text{ has an independent set of size } \geq n/2 \\ &\iff \ H \text{ has an independent set of size } \geq n/2 \end{aligned}$$

$\iff$ H has an independent set of size $\geq p/3$

(since $p/3 = (3n/2)/3 = n/2$)

$\iff$ $H \in$ BIG INDEPENDENT SET

It remains to show that the reduction runs in polynomial time. To see this, observe that the reduction first creates $n/2$ new vertices (time $O(n)$), and then adds edges between each pair of them, and between each new vertex and each old vertex. The number of pairs of new vertices is $\begin{pmatrix} n/2 \\ 2 \end{pmatrix}$, which is $O(n^2)$. The number of pairs consisting of one new vertex and one old vertex is $(n/2) \cdot n$, which is also $O(n^2)$. So the total time required to construct H from G is at most polynomial.

Official use only
8

Working Space

END OF EXAMINATION