FIT2014 Exam 2017 Questions and Solution

Theory Of Computation (Monash University)

Faculty of Information Technology

Monash University

FIT2014 Theory of Computation
FINAL EXAM
SAMPLE SOLUTIONS

2nd Semester 2017

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 A, K and L, with the following meanings.

A: alphaGo is the equal-best Go player in the world.
K: Kē Jié is the equal-best Go player in the world.
L: Lee Se-dol is the equal-best Go player in the world.

Use A, K and L to write a proposition that is True if and only if **exactly two** of alphaGo, Kē Jié and Lee Se-dol are the equal-best Go players in the world.

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

$(A \vee K) \wedge (A \vee L) \wedge (K \vee L) \wedge (\neg A \vee \neg K \vee \neg L)$

Question 2 **(3 marks)**

Suppose you have predicates `computableByGoedel`, `computableByChurch`, and `computableByTuring` with the following meanings, where variable $F : \{\mathtt{a}, \mathtt{b}\}^* \to \mathbb{N} \cup \{0\}$ represents an arbitrary function from finite strings over $\{\mathtt{a}, \mathtt{b}\}$ to nonnegative integers:

`computableByGoedel`(F):	the function F is a *recursive function*, according to Kurt Gödel's definition.
`computableByChurch`(F):	the function F has a *lambda expression*, in the sense of Alonzo Church's Lambda Calculus.
`computableByTuring`(F):	the function F is *computable*.

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

> Every function that satisfies any one of the three definitions of computability — recursive functions (Gödel), lambda calculus (Church), or Turing computability — also satisfies each of the others.

To do this, you may only use: the above three 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.)

$$\begin{aligned} \forall F : \quad & (\texttt{computableByGoedel}(F) \lor \texttt{computableByChurch}(F) \lor \texttt{computableByTuring}(F)) \\ \implies \quad & (\texttt{computableByGoedel}(F) \land \texttt{computableByChurch}(F) \land \texttt{computableByTuring}(F)) \end{aligned}$$

Some alternatives:

$$\begin{aligned} \forall F : \quad & (\texttt{computableByGoedel}(F) \implies \texttt{computableByChurch}(F)) \\ \land \; & (\texttt{computableByChurch}(F) \implies \texttt{computableByTuring}(F)) \\ \land \; & (\texttt{computableByTuring}(F) \implies \texttt{computableByGoedel}(F)) \end{aligned}$$

It's ok if *all* the directions of implication are reversed here.

An alternative, unnecessarily verbose but still correct:

$$\begin{aligned} \forall F : \quad & (\texttt{computableByGoedel}(F) \implies (\texttt{computableByChurch}(F) \land \texttt{computableByTuring}(F))) \\ \land \; & (\texttt{computableByChurch}(F) \implies (\texttt{computableByTuring}(F) \land \texttt{computableByGoedel}(F))) \\ \land \; & (\texttt{computableByTuring}(F) \implies (\texttt{computableByGoedel}(F) \land \texttt{computableByChurch}(F))) \end{aligned}$$

Any or all of the second conjuncts, following the implications, can be omitted; if all are omitted, we have the second solution given above.

Another approach is to take *at least two of the three pairs* and link them by two-way implications (biconditionals). For example:

$$\begin{aligned} \forall F: \quad & (\texttt{computableByGoedel}(F) \iff \texttt{computableByChurch}(F)) \\ & \wedge (\texttt{computableByChurch}(F) \iff \texttt{computableByTuring}(F)) \end{aligned}$$

Official use only
7

Question 3 **(6 marks)**

Let E_n be the following Boolean expression in variables $x_1, x_2, \ldots, x_n$:

$$((\cdots((((\neg x_1 \vee x_2) \wedge \neg x_2) \vee x_3) \wedge \neg x_3) \cdots) \vee x_n) \wedge \neg x_n$$

For example, $E_1 = \neg x_1$, and $E_2 = (\neg x_1 \vee x_2) \wedge \neg x_2$. In general, $E_n = (E_{n-1} \vee x_n) \wedge \neg x_n$.

Prove by induction on n that, for all $n \geq 1$, the expression E_n is satisfiable.

Inductive basis:

If $n = 1$, we have $E_1 = \neg x_1$, which is satisfiable because $x_1 =$ False satisfies it.

Inductive step:

Suppose $n \geq 2$. Assume E_{n-1} is satisfiable (the Inductive Hypothesis). So it has a satisfying truth assignment to its variables, $x_1, x_2, \ldots, x_{n-1}$. Consider $E_n = (E_{n-1} \vee x_n) \wedge \neg x_n$. (Note, this uses $n \geq 2$.) We have a satisfying truth assignment that makes E_{n-1} true. Under this assignment, $E_{n-1} \vee x_n$ is also true, so in this case we have $E_n = (E_{n-1} \vee x_n) \wedge \neg x_n =$ True $\wedge \neg x_n = \neg x_n$. This can be satisfied by putting $x_n =$ False. So E_n is satisfiable.

Therefore E_n is satisfiable for all n, by Mathematical Induction.

Official use only
6

Question 4 **(4 marks)**

Write down all strings of length ≤ 6 that match the following regular expression:

$$\varepsilon \cup (\mathtt{ab})^*\mathtt{b}(\mathtt{aaa} \cup \mathtt{bb})^*$$

ε, b, abb, bbb, baaa, abbaaa, abbbb, ababb, bbbbb, baaabb, bbbaaa

---

Official use only
4

Question 5 **(4 marks)**

Let R be any regular expression.

(a) Give, in terms of R, a regular expression for the language of all strings that can be divided into two substrings, each of which matches R.

RR

(b) Prove that the language EVEN(R) of all strings that can be divided into *any even number* of substrings, each of which matches R, is regular.

> For example, if R is $\texttt{a} \cup \texttt{bb}$, then the string $w = \texttt{aabba}$ can be divided into four substrings $\texttt{a}$,$\texttt{a}$,$\texttt{bb}$,$\texttt{a}$ which each match R. So this w belongs to EVEN(R). The empty string is also in EVEN(R), noting that zero is an even number. But $\texttt{aabb}$ and $\texttt{bbb}$ do not belong to EVEN(R).

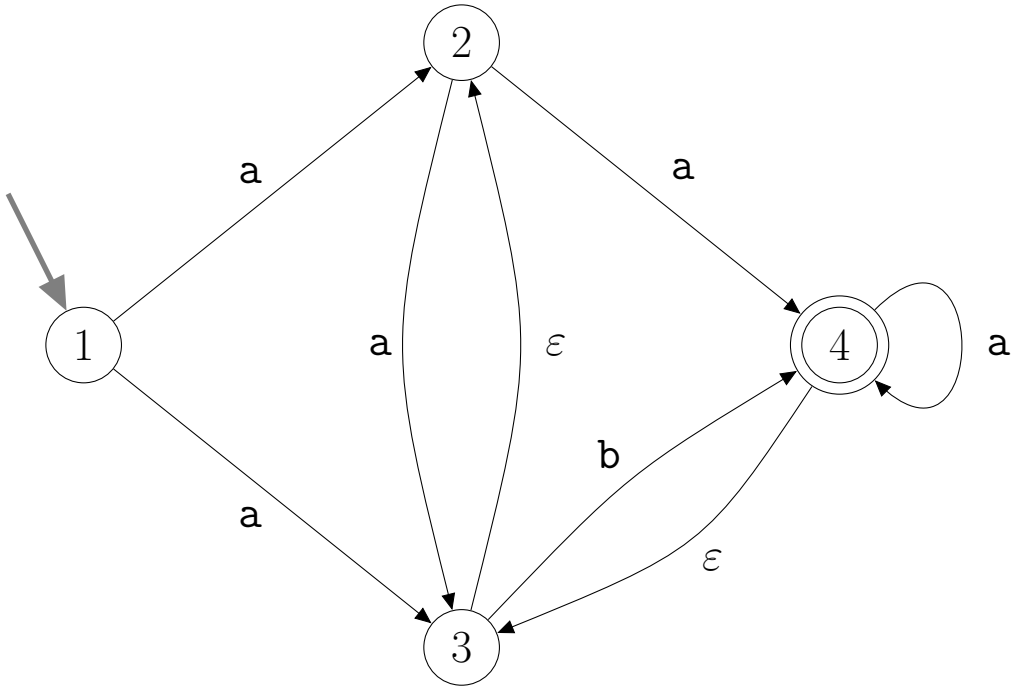
Suppose that a string s can be divided into an even number of substrings, each of which matches R. Let the number of such substrings be $2k$. Then s can be divided into k pairs of substrings, with each pair consisting of two strings that match R. Each pair must therefore match RR, by (a). So s can be divided into k strings that match RR. Since k can be any number (including 0), our string s must match $(RR)^*$. Conversely, any string that matches $(RR)^*$ must consist of some number of substrings that each match RR, and therefore it must consist of an even number of substrings that each match R.

This shows that EVEN(R) is described by the regular expression $(RR)^*$, so it must be regular.

Official use only
4

Question 6 **(8 marks)**

(a) Convert the following Nondeterministic Finite Automaton (NFA) into an equivalent Deterministic Finite Automaton (FA).



Your FA must be presented by filling in some rows in the following table. You may not need all the rows available.

state	a	b
Start {1}	{2,3}	∅
{2,3}	{2,3,4}	{2,3,4}
Final {2,3,4}	{2,3,4}	{2,3,4}
∅	∅	∅

It's ok if the states are renumbered, as long as the FA is correct. For example:

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

(b) Give a regular expression for the language accepted by the above NFA.

(You should not need to apply a general automaton-to-regexp conversion algorithm. Just think about what the automaton does. The equivalent FA should help.)

$\mathtt{a}(\mathtt{a} \cup \mathtt{b})(\mathtt{a} \cup \mathtt{b})^*$ or, equivalently, $\mathtt{a}(\mathtt{a} \cup \mathtt{b})^*(\mathtt{a} \cup \mathtt{b})$

Question 7 **(5 marks)**

Give an algorithm which takes, as input, a Finite Automaton represented as a table, and finds another Finite Automaton that accepts the same language as the first one and has the minimum number of states among all FAs that accept that language.

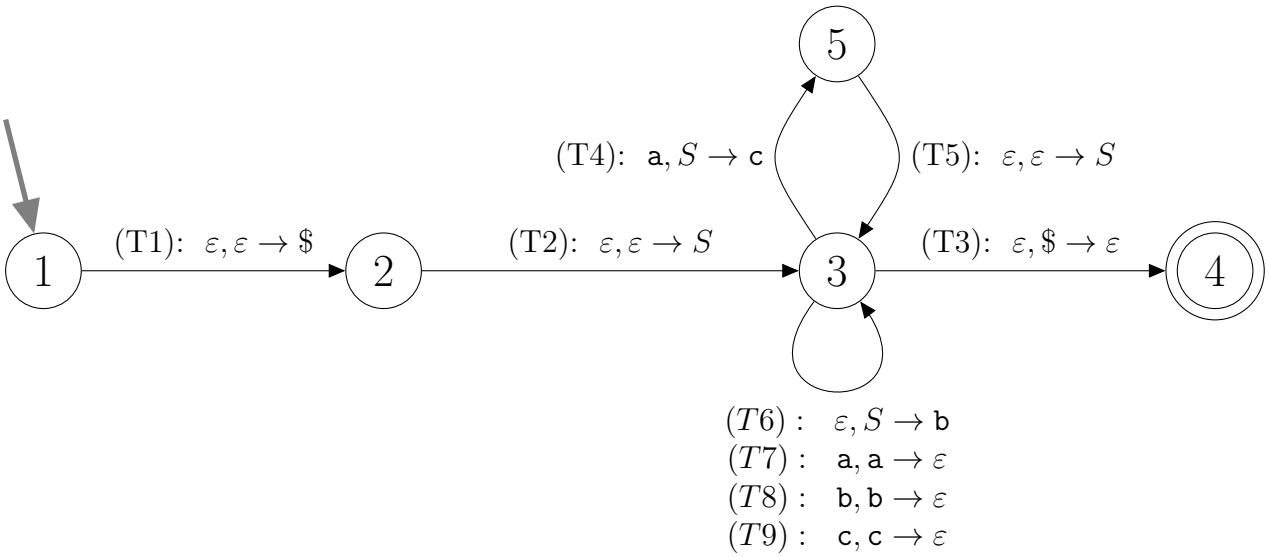
A pseudocode description is fine.
Do not try to write your algorithm as a Turing machine.

Input: FA represented as table.
Give all non-Final states the first colour.
Give all Final states the second colour.
Apply these colours throughout the table (i.e., not just in the states column).
While (there exist two states of the same colour whose rows have different colour patterns)
{
    Pick one of these states.
    Identify all other states whose rows have the same colour pattern as it.
    Give each of these states the same new colour.
        I.e., they each get a new colour, and they all get the same colour.
    Apply this new colour to that state throughout the table.
}
For each colour:
{
    Merge all states of that colour into a single state.
    Throughout table, replace the names of these states by the name of the merged state.
}
Output: the new FA, as described by the new table.

Official use only
13

Question 8 **(12 marks)**

Consider the following Pushdown Automaton (PDA), with input alphabet {a,b,c} and extra stack symbols S and $.



(a) Show that the single-letter string b is accepted by this PDA, by giving the sequence of transitions that leads to acceptance of b.

Use the names of the transitions, i.e., $T1, T2, \ldots$, etc.

T1, T2, T6, T8, T3

(b) Prove the following statement by induction on n:

For all $n \geq 0$:
If the PDA is in State 3, the top symbol on the stack is S, and the remaining input begins with $\mathtt{a}^n\mathtt{bc}^n$, **then** after reading $\mathtt{a}^n\mathtt{bc}^n$ the PDA is again in State 3 and the stack is the same except that the S on the top has been removed.

Inductive basis: $n = 0$: the remaining input is `b`, and if S is on top of the stack, the transitions T6, T8 leave us still in state 3, having read the input, and S has been removed from the top of the stack. No other change has been made to the stack.

Inductive step:

Suppose the statement is true for n (the *Inductive Hypothesis*). Suppose the PDA is in state 3, the remaining input starts with $\mathtt{a}^{n+1}\mathtt{bc}^{n+1}$, and that S is on top of the stack. Then the combined effect of transitions T4, T5 is to read the first `a` of input and replace the S on top of the stack by `c` and then S, i.e., the only change to the stack is that `c` is underneath S; the portion of the stack below this `c` is unchanged, and S is still on top of the stack.

So the situation is now that the input begins with $\mathtt{a}^n\mathtt{bc}^{n+1}$, which is $\mathtt{a}^n\mathtt{bc}^n\mathtt{c}$, which begins with $\mathtt{a}^n\mathtt{bc}^n$; also, we are again in state 3, and S is on top of the stack. This is the "if"-part of the given statement, for n. By the Inductive Hypothesis, after reading $\mathtt{a}^n\mathtt{bc}^n$, the PDA is again in State 3 and the stack is the same except that the S on the top has been removed. The next thing on the stack is the `c` we put there before (using T4 and T5). The next unread letter is `c`. So we can apply T9, which reads that letter and pops the `c` off the stack. We are back at state 3 again, and the stack is the same as it was when we began the inductive step except that the S that was then on the top of it is not there any more. This is exactly our desired conclusion, so we have established the given statement for $n + 1$.

By the Principle of Mathematical Induction, the statement holds for all $n \geq 0$.

(c) Hence prove that, for all $n \geq 0$, the string $\texttt{a}^n\texttt{bc}^n$ is accepted by this PDA.

When this string is given as input to this PDA, before reading anything it pushes \$ and then S onto the stack. We are then in state 3, with S on top of the stack, and the string to be read begins with (and in fact equals) $\texttt{a}^n\texttt{bc}^n$. By part (b), the PDA reads this string and then is in state 3 with S removed from the top of the stack, and otherwise the stack is unchanged. So all that's left on the stack is the single symbol \$. The PDA now does transition T3. At this stage, the entire input has been read, and the PDA is in its Final state, so the input string is accepted.

Official use only
12

Question 9 **(13 marks)**

Let GOAL be the language generated by the following Context-Free Grammar:

$$S \rightarrow \texttt{goo}X\texttt{al} \quad (1)$$
$$X \rightarrow \texttt{oo}X\texttt{a} \quad (2)$$
$$X \rightarrow \varepsilon \quad (3)$$

GOAL consists of all strings of the form $\texttt{g(oo)}^n\texttt{a}^n\texttt{l}$, where $n \geq 1$.

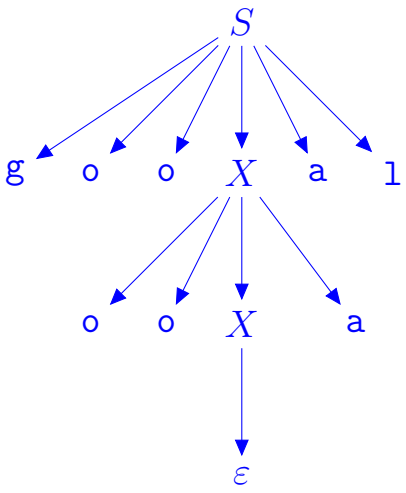
The first few strings in this language, in order of increasing length, are:

gooal, gooooaal, goooooooaaal, ...

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

$$\begin{aligned} S &\Rightarrow \texttt{goo}X\texttt{al} && (1) \\ &\Rightarrow \texttt{gooo}X\texttt{aal} && (2) \\ &\Rightarrow \texttt{gooo}\varepsilon\texttt{aal} && (3) \\ &= \texttt{gooooaal} \end{aligned}$$

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



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

Assume, by way of contradiction, that GOAL is regular. Then, by Kleene's Theorem, there is a Finite Automaton that accepts it. Let N be the number of states of this FA.

Let w be the string $\texttt{go}^{2N}\texttt{a}^{N}\texttt{l}$. This has length $> N$. Therefore, by the Pumping Lemma for Regular Languages, w can be divided into substrings x, y, z such that $y \neq \varepsilon$, $|xy| \leq N$, and for all $i \geq 0$ the string $xy^iz \in$ GOAL.

The constraint $|xy| \leq N$ tells us that y is within the first N letters of w. This means that y *either* contains the g at the very start of w *or* lies within the substring $\texttt{o}^{2N}$. If it contains g, then repeating it to form $xyyz$ gives a string with more than one g, which therefore cannot belong to GOAL (since every string in GOAL just has one g, the one at the beginning). If y lies within the substring $\texttt{o}^{2N}$, then repeating it to form $xyyz$ increases the length of the o-substring: it now has length $> 2N$ (using $y \neq \varepsilon$). But the length of the a-substring has not changed: it's still N. So the length of the o-substring is no longer exactly twice the length of the a-substring. This breaks the rules of the language GOAL, so $xyyz$ cannot belong to GOAL. So, regardless of where y is located in w, the string $xyyz \notin$ GOAL.

This violates the conclusion of the Pumping Lemma. So we have a contradiction. Therefore our initial assumption, that GOAL is regular, was wrong. So GOAL is not regular.

Deletion of y works just as well as repeating it, since the Pumping Lemma says that $xy^iz \in$ GOAL for all $i \geq \mathbf{0}$; the $i = 0$ case tells us that $xz \notin$ GOAL.

Official use only
13

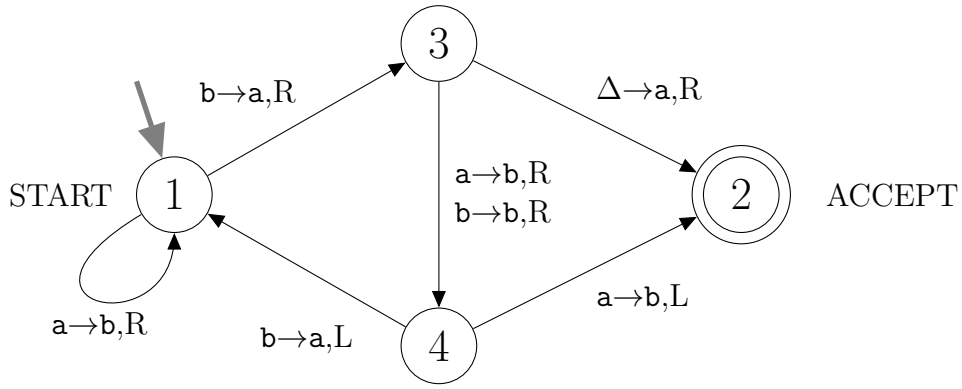
Question 10 **(2 marks)**

The Cocke-Younger-Kasami (CYK) algorithm is less efficient than most commonly used parsing algorithms. What is one significant advantage it has over those algorithms?

It's more general: it can parse *any* context-free language.

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

Question 12 **(5 marks)**

(a) Name three variations on Turing machines that give the same class of computable functions.

Any three of:

- two-way infinite tape (i.e., infinite on the left (negative) side, as well as on the right (positive) side);
- more than one tape;
- allowing the tape head to stay still, as well as to move left or right;
- allowing the tape head to move some number of steps to the left or right, instead of just one step, with a fixed bound on the number of steps;
- forcing the tape head to stay still, or move to the right, if ever it tries to move off the left end of the tape (instead of crashing in these circumstances);
- using a two-dimensional grid instead of a (one-dimensional) tape;
- allowing the actions to depend on the contents of the neighbouring tape cells as well as the current tape cell;
- giving access to a source of random symbols (i.e., probabilistic Turing machine). [We haven't studied this.]

(b) Give one way of modifying the definition of Turing machines so that they can still recognise all regular languages but can no longer recognise all decidable languages.

For full marks, your modification should be as simple as possible. It should involve altering just one part of the definition of Turing machines. Replacement of an entire machine by something else is not acceptable.

No proof is required for this question.

Any of the following is correct:

- restrict the tape head so that it only moves to the right;
- prevent the tape head from moving to the left;
- when the tape head moves to the left, the cell it was on becomes blank.

The following answers are *incorrect*:

- prevent the tape head from moving to the *right*;
- any of the modifications in part (a), which don't change the class of languages recognised.

Official use only
5

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: two Turing machines M_1 and M_2. Question: Does M_1 eventually halt, when given M_2 as input?	☐ Decidable	☑ Undecidable
Input: two Turing machines M_1 and M_2. Question: Does M_1 have the same number of states as M_2?	☑ Decidable	☐ Undecidable
Input: two Turing machines M_1 and M_2. Question: Is M_1 equivalent to M_2 (i.e., do M_1 and M_2 have the same sets of accepted strings and the same sets of rejected strings)?	☐ Decidable	☑ Undecidable
Input: two Turing machines M_1 and M_2. Question: If each machine is given itself as input, does M_1 finish before M_2?	☐ Decidable	☑ Undecidable

Official use only
4

Question 14 **(12 marks)**

The Venn diagram on the next page 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 binary strings.

(b) The set of all strings in which every `a` occurs before every `b`.

(c) The set of all strings in which `a` occurs more times than `b`.

(d) The set of all strings in which every `a` occurs before every `b` and ALSO `a` occurs more times than `b`.

(e) The set of adjacency matrices of 2-colourable graphs.

(f) The set of adjacency matrices of 4-colourable graphs.

(g) The set of all Boolean expressions in Conjunctive Normal Form (CNF), whether satisfiable or unsatisfiable. (Assume the variables are $x_1, x_2, \ldots, x_n$, and variable x_i is represented by its index i as a binary positive integer.)

(h) The set of all encodings of Turing machines that accept regular languages.

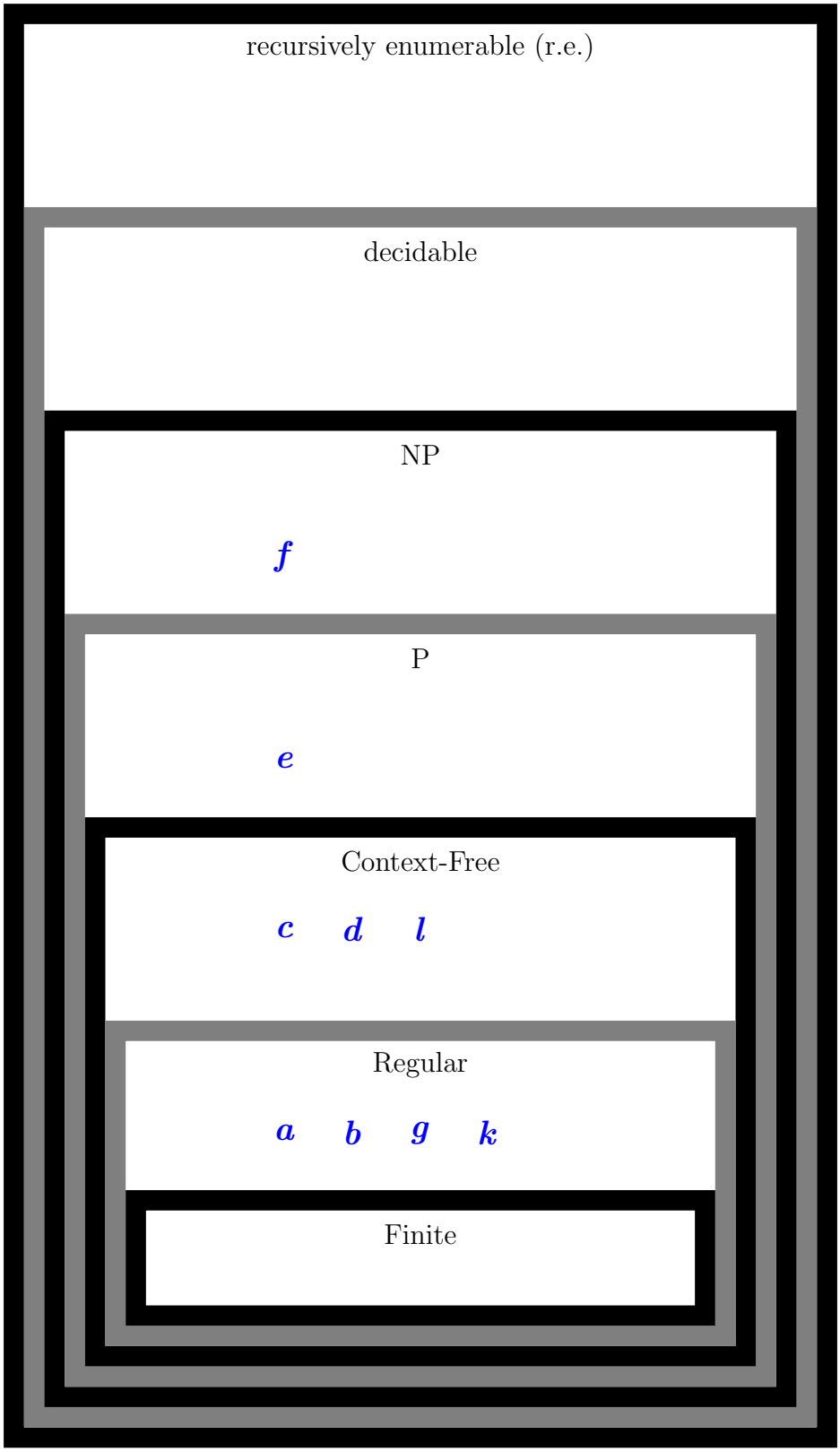
(i) The set of all encodings of Turing machines that accept non-context-free languages.

(j) The set of all encodings of Turing machines that loop forever for some input.

(k) The set of all arithmetic expressions involving positive integers, in decimal notation, and addition, subtraction, multiplication and division, but with no parentheses.

(l) The set of all arithmetic expressions involving positive integers, in decimal notation, and addition, subtraction and parentheses, but no multiplication or division.

h ***i*** ***j***



Working Space

Question 15 **(8 marks)**

Let L and K be languages. Suppose that K is finite.

Definition: The **symmetric difference** $L\Delta K$ consists of all strings that belong to L or K, *but not both*. In other words, $L\Delta K = (L \cup K) \setminus (L \cap K)$.

(a) Prove that there exists a mapping reduction from L to $L\Delta K$.

Because K is finite, $K \cap L$ and $K \setminus L$ are also finite, and hence both are decidable.

Here is such a mapping reduction.

Let s be a string in $K\Delta L$ and let s' be a string not in $K\Delta L$.

```
Input:   string x.
Use a K-decider to determine whether or not x ∈ K.
If   x ∈ K ∩ L
//   ... ⟹ it's in L, so it must be mapped to a string in KΔL. But x itself is not in KΔL,
//   so we can't just output x. Instead, output some fixed string known to be in KΔL.
{
      output s
}
else if   x ∈ K \ L
//   ... ⟹ it's not in L, so it must be mapped to a string outside KΔL. But x itself is
//   in KΔL, so we can't just output x. Instead, output some fixed string known not to
//   be in KΔL.
{
      output s'
}
else     //   x ∉ K, so x ∈ L ⟺ x ∈ KΔL
{
      output x
}
```

This algorithm, together with the decidability of $K \cap L$ and $K \setminus L$, implies that this function is computable.

If $x \in L$, then either $x \in K$, in which case it is in $K \cap L$ and is mapped to s which is in $K\Delta L$ (first case of **if** statement), or $x \notin K$, in which case x is mapped to itself and is in $K\Delta L$ (third case of **if** statement).

If $x \notin L$, then either $x \in K$, in which case it is in $K \setminus L$ and is mapped to s' which is not in $K\Delta L$ (second case of **if** statement), or $x \notin K$, in which case x is mapped to itself and is not in $K\Delta L$ (third case of **if** statement).

We have given the mapping reduction and shown that it is indeed a mapping reduction from L to $K\Delta L$.

(b) Prove that L is undecidable if and only if $L \Delta K$ is undecidable.

($\Longrightarrow$)

The undecidability of L, together with the mapping reduction from L to $K \Delta L$, implies that $K \Delta L$ is undecidable.

($\Longleftarrow$)

If $K \Delta L$ is undecidable, then apply the forward direction we just proved to $K \Delta L$ instead of to L. (This is totally fine: it can be applied to any undecidable language, whatever that language may be called.) We deduce that $(K \Delta L) \Delta K$ is undecidable. But $K \Delta L \Delta K = L$. Therefore L is undecidable.

Alternative solution, using contrapositive:

If L is decidable, then using the decidability of K we see that $K \Delta L$ is decidable. (A string is in $K \Delta L$ if and only if it belongs to *exactly one* of K and L, and we can use deciders for K and L to determine whether or not this is the case.)

Official use only
8

Question 16 **(5 marks)**

An enumerator for a language is normally allowed to output a given string in the language more than once. A **direct enumerator** is an enumerator which never outputs a string more than once.

Prove that if a language has an enumerator then it has a direct enumerator.

Let L be any language, and let M be an enumerator for L.

We construct a direct enumerator M' for L as follows. M' simulates M and also maintains a list of all strings output so far. Whenever M outputs a string, M' looks it up in the list of strings output so far. Note that the list is always finite (although it grows without limit, if L is infinite), so this lookup can be done in a finite amount of time and is computable. If the string appears in that list (so it has already been output by M), then M' just continues without outputting that string. If the string does not appear in the list (so it has never been output before), then M' outputs it, and the string is appended to the list.

Every string in L is eventually output by M', since it must be output by M, and the first time that happens it will be output by M' also. No string not in L can ever be output by M', since to be output by M' it would first have to be output by M and therefore be in L. So M' is indeed an enumerator for L. Furthermore, the use of the list ensures that no string is ever output by M' more than once, so it is a direct enumerator for L.

Official use only
5

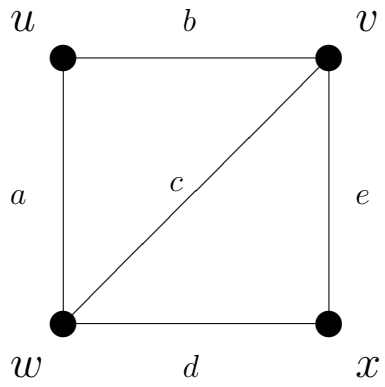
Question 17 **(11 marks)**

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

The VERTEX COVER decision problem is as follows.

VERTEX COVER
Input: Graph G.
Question: Does G have a vertex cover?

For example, in the following graph, the vertex set $\{u, w, x\}$ is a vertex cover, and so is $\{v, w\}$. But $\{u, x\}$ is not a vertex cover, since it does not meet every edge. (Specifically, it misses the diagonal edge c.)



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 vertex covers in the above graph W.

$$(u \vee v) \wedge (u \vee w) \wedge (v \vee w) \wedge (v \vee x) \wedge (w \vee x)$$

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

Input: graph G.
For each vertex of G, create a variable to represent that vertex.
For each edge e of G:
{
    Create a clause consisting of a disjunction of the two variables associated with the endpoints of e.
}
Combine all the clauses using conjunction.
Output the expression so formed.

Alternative solution:

This variant of VERTEX COVER has no constraint on the size of the vertex cover. So, in fact, for any input graph G, the answer is always YES, since every graph has a vertex cover: take the entire vertex set of the graph, for example.[1]

So, for a polynomial-time reduction, it's sufficient to just map any input graph to some fixed satisfiable CNF expression. (Any input string that is not even a graph could be mapped to a fixed **un**satisfiable CNF expression.)

Official use only
11

[1] Normally, the VERTEX COVER problem includes a second input, being a positive integer k, and the problem asks if there is a vertex cover *size at most* k. That version is NP-complete, as shown in the last lecture.

Question 18 **(8 marks)**

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

Definitions:

HAMILTONIAN PATH
Input: Graph G.
Question: Does G have a path that contains every vertex?

LONG PATH
Input: Graph G, with an even number of vertices.
Question: Does G contain a path of length $\geq n/2$?

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

LONG PATH is in NP, because it has a polynomial-time verifier. Given an input graph G on n vertices, the certificate is a path of length $\geq n/2$ in G. The verifier just goes along the path, checking that the successive vertices are indeed adjacent, that no vertex is repeated, and that there are $\geq n/2$ edges altogether. This takes polynomial time.

We now give a polynomial-time reduction from HAMILTONIAN PATH to LONG PATH.

Input: graph G.
Create $n-1$ new vertices, $v_1, v_2, \ldots, v_{n-2}$, that are distinct from all vertices of G and from v_0.
Output this new graph.

The time taken by this algorithm is dominated by the creation of $n-1$ new vertices, so it runs in polynomial time.

Let H be the new graph constructed from G by this algorithm. We show that G has a Hamiltonian path if and only if H has a "long path", i.e., a path of length $\geq |V(H)|/2$.

Suppose G has a Hamiltonian path P. This has length $n-1$. This path is still present in H. The number of vertices of H is given by

$$|V(H)| = n + (n-2) = 2n - 2 = 2(n-1).$$

The length of P is half this amount. So H has a path of length $\geq |V(H)|/2$. So $H \in$ LONG PATH.

Now suppose $H \in$ LONG PATH. So H has a path of length $|V(H)|/2$, which is $2(n-1)/2$, which is $n-1$. This path cannot include the new isolated vertices, so it must be in G. A path of length $n-1$ in G must be a Hamiltonian path. So G has a Hamiltonian path.

Official use only
8

Working Space

END OF EXAMINATION