Clayton School of Information Technology

Monash University

FIT2014 Theory of Computation
FINAL EXAM
SOLUTIONS

2nd semester, 2013

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)**

You are hunting a mouse in your wardrobe.

Suppose we have three propositions, C, H and S, with the following meanings:

C: The mouse is in your coat.
H: The mouse is in your hat.
S: The mouse is in your shoe.

Use C, H and S to write a proposition, in Conjunctive Normal Form, that is True precisely when the mouse is in one of these three locations: your coat, your hat or your shoe.

$$(C \vee H \vee S) \wedge (\neg C \vee \neg H) \wedge (\neg C \vee \neg S) \wedge (\neg H \vee \neg S)$$

It's ok to have it in 3-CNF, which would have 7 clauses, provided done correctly. But the question does not stipulate 3-CNF.

Question 2 **(4 marks)**

Suppose you have the predicates `computer` and `utm` with the following meanings:

`computer`(X):	X is a computer.
`utm`(X):	X can simulate any Turing machine.

(a) Write a universal statement in predicate logic with the meaning:

"Everything that can simulate any Turing machine is a computer."

$$\forall X \ (\, \texttt{utm}(X) \Rightarrow \texttt{computer}(X) \,)$$

Alternative:

$$\forall X \ (\, \neg\texttt{utm}(X) \vee \texttt{computer}(X) \,)$$

(b) What additional fact would you need to know, to be able to use this statement (and nothing else) to prove that the object `myPhone` is a computer? (Express this fact as an atomic sentence in predicate logic.)

$\texttt{utm}(\texttt{myPhone})$

No need for full stop at the end, as this is predicate logic, not a Prolog fact.

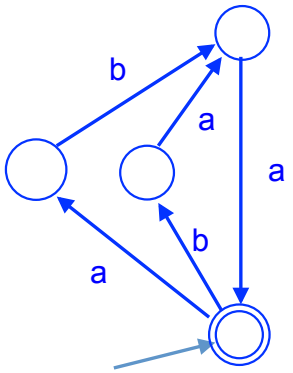
8

Question 3 **(4 marks)**

(a) Write down all strings of at most 8 letters, over alphabet {a,b}, that match the regular expression $(((\texttt{ab}) \cup (\texttt{ba}))\texttt{a})^*$.

ε, aba, baa, abaaba, ababaa, baaaba, baabaa.

(b) Give an NFA with at most 7 states that recognises the language described by this regular expression.



Equivalent answers with extra empty transitions are ok.

Question 4 **(3 marks)**

A language over alphabet {a,b} is said to be *cofinite* if it contains all strings over that alphabet *except* for some finite number of strings. Prove that every cofinite language is regular.

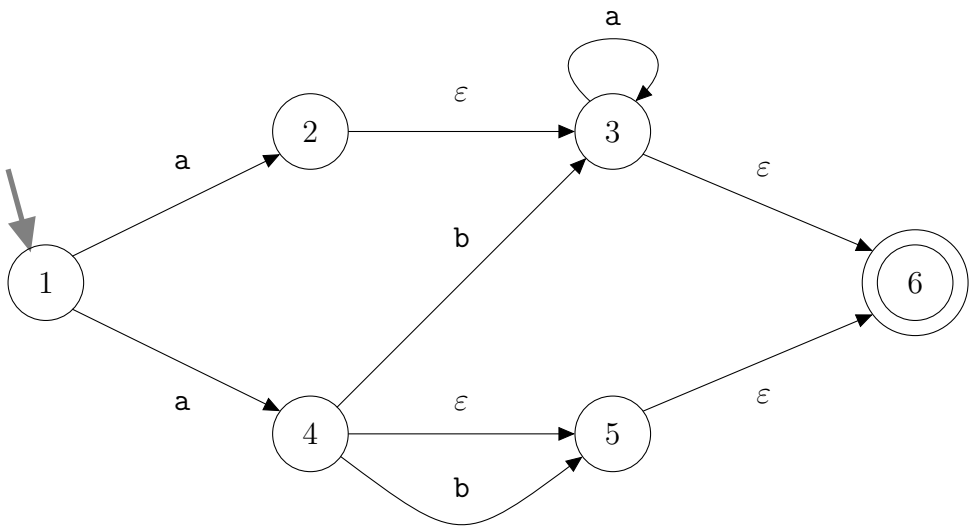
A cofinite language has a finite complement.
Any finite language is regular.
Regular languages are closed under complement.
So any cofinite language (being the complement of a regular language) is regular.

Alternative solution:

Let L be any cofinite language.
Its complement, $\overline{L}$, is finite.
Therefore $\overline{L}$ is regular, since every finite language is regular.
Therefore, $\overline{L}$ is recognised by a Finite Automaton, by Kleene's Theorem.
If we change that FA so that every Final state becomes Non-Final, and every Non-Final state becomes Final, then we obtain an FA that accepts precisely those strings *not* in $\overline{L}$ — in other words, this FA recognises L.
Therefore L is regular, by Kleene's Theorem.

Question 5 **(7 marks)**

Given the following NFA, convert it to a Finite Automaton that recognises the same language.



Represent the FA by filling in the table below.

	state	a	b
Start	1	{2,3,4,5,6}	∅
Final	{2,3,4,5,6}	{3,6}	{3,5,6}
Final	{3,6}	{3,6}	∅
Final	{3,5,6}	{3,6}	∅
	∅	∅	∅

It's ok if the two states labelled {3,6} and {3,5,6} are merged. In fact, if the above FA is simplified using the state-minimisation algorithm, then those two states would be merged into one.

7

Question 6 **(3 marks)**

Suppose you have, at your disposal, algorithms to convert NFAs to FAs, FAs to Regular Expressions and Regular Expressions to NFAs.

Explain how you would design and implement a lexical analyser to recognise tokens that match a particular regular expression. (In this explanation, do not give code, but do explain where the code comes from.)

Use one of the given algorithms to convert the regular expression to an NFA to recognise it.
Then use another given algorithm to convert the NFA to an equivalent FA.
Then convert the FA to code.

Question 7 **(6 marks)**

Use the Pumping Lemma to prove that the language

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

is not regular.

Suppose that the given language (call it L) is regular.
Then there is a FA that recognises it. Let N be the number of states of this FA.

Let w be any string $\mathtt{a}^m\mathtt{b}^n$ in L with $m > N$.
This certainly ensures $|w| > N$, which we need in order to apply the Pumping Lemma.

Then the Pumping Lemma for Regular Languages tells us that $w = xyz$ where substrings x, y, z satisfy: $y \neq \varepsilon$, $|xy| \leq N$, and $xy^iz \in L$ for all i.

It is not possible for y to include any of the letter `b`, since in that case xy would include every `a`, and there are m of them, so we would have $|xy| > m$. This, together with $m > N$ (by construction of w), would contradict $|xy| \leq N$ (which we got from the Pumping Lemma).

So y falls entirely within the `aa...a` at the start. This means that pumping increases the number of `a`s (using the fact that $y \neq \varepsilon$) but not the number of `b`s. Sufficient pumping will yield more `a`s than `b`s, giving a non-member of L, which contradicts the conclusion of the Pumping Lemma. Hence our assumption, that L is regular, is false.

One variation might be to use $w = \mathtt{a}^m\mathtt{b}^{m+1}$, but still have $m > N$. This means that, as soon as you pump *once*, to get xy^2z, then you have at least as many `a`s as `b`s, hence a non-member of L, hence a contradiction.

Working Space

Question 8 **(2 marks)**

Give a Context-Free Grammar for the language

$$\{\ \mathtt{a}^m\mathtt{b}^n\ :\ m > 2n,\ n \geq 0\ \}.$$

$$\begin{aligned} S &\to \mathtt{a}X \\ X &\to \mathtt{aa}X\mathtt{b} \\ X &\to \mathtt{a}X \\ X &\to \varepsilon \end{aligned}$$

The following would be fine if, in addition, the empty string were included in this language. But it does not satisfy the condition.

$$\begin{aligned} S &\to \mathtt{aa}S\mathtt{b} \\ S &\to \mathtt{a}S \\ S &\to \varepsilon \end{aligned}$$

The following would be fine if the condition were modified to require $n > 0$ instead of $n \geq 0$.

$$\begin{aligned} S &\to \mathtt{aa}X\mathtt{b} \\ X &\to \mathtt{aa}X\mathtt{b} \\ X &\to \mathtt{a}X \\ X &\to \varepsilon \end{aligned}$$

Question 9 **(6 marks)**

Consider the following Context-Free Grammar

$$S \rightarrow \mathtt{a}\,S \quad (1)$$
$$S \rightarrow S\,\mathtt{b} \quad (2)$$
$$S \rightarrow \varepsilon \quad (3)$$

Prove by induction that every string of the form $\mathtt{a}^m\mathtt{b}^n$, where $m \geq 0$ and $n \geq 0$, can be generated by this grammar.

We prove this by induction on the length of the string (i.e., on $m + n$).

Inductive basis: if the length is 0, then the string is empty, and is generated by (3).

Inductive step: Suppose the result holds for all strings of length $< \ell$, where $\ell > 0$.
Now suppose we have a string $w = \mathtt{a}^m\mathtt{b}^n$ of length ℓ.
Either the string starts with $\mathtt{a}$, or ends with $\mathtt{b}$, or both. (This uses $\ell > 0$.)
If it starts with $\mathtt{a}$, then it can be written $w = \mathtt{a}w'$, where $|w'| = |w| - 1 = \ell - 1 < \ell$. The fact that w' has length $< \ell$ means we can use the inductive hypothesis to conclude that w' can be generated by the grammar. So we have a derivation

$$S \Rightarrow \cdots \Rightarrow w'.$$

By prefixing each string in this derivation by $\mathtt{a}$, we can form another derivation:

$$\mathtt{a}S \Rightarrow \cdots \Rightarrow \mathtt{a}w' = w.$$

Prepending this derivation with the production $S \Rightarrow \mathtt{a}S$ gives a derivation of w:

$$S \Rightarrow \mathtt{a}S \Rightarrow \cdots \Rightarrow \mathtt{a}w' = w.$$

So w is generated by the grammar.
A similar argument applies if the string w ends with $\mathtt{b}$. This completes the inductive step.

The result follows, by the Principle of Mathematical Induction.

Working Space

Question 10 **(5 marks)**

Consider the following Context-Free Grammar.

$$
\begin{aligned}
S &\rightarrow \texttt{a}B\texttt{a} && (1)\\
B &\rightarrow BB && (2)\\
B &\rightarrow Q && (3)\\
B &\rightarrow R && (4)\\
Q &\rightarrow \texttt{q} && (5)\\
R &\rightarrow \texttt{r} && (6)
\end{aligned}
$$

Give

(a) a derivation, and

(b) a parse tree,

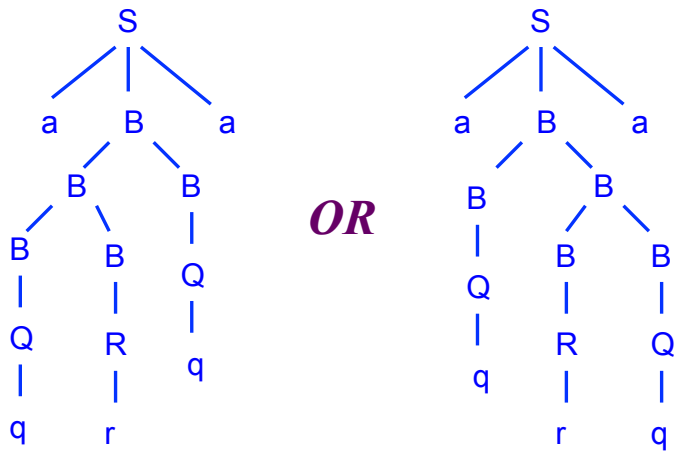
for the string `aqrqa`, labelling each step in the derivation on its right by the number of the rule used. Use the spaces below for your answers.

(a) Derivation:

$$
\begin{aligned}
\texttt{S} &\Rightarrow \texttt{aBa} && \text{by (1)}\\
&\Rightarrow \texttt{aBBa} && \text{by (2)}\\
&\Rightarrow \texttt{aBBBa} && \text{by (2)}\\
&\Rightarrow \texttt{aQBBa} && \text{by (3)}\\
&\Rightarrow \texttt{aQRBa} && \text{by (4)}\\
&\Rightarrow \texttt{aQRQa} && \text{by (3)}\\
&\Rightarrow \texttt{aqRQa} && \text{by (5)}\\
&\Rightarrow \texttt{aqrQa} && \text{by (6)}\\
&\Rightarrow \texttt{aqrqa} && \text{by (5).}
\end{aligned}
$$

Some variation in the order is possible.
Doesn't have to be leftmost or rightmost derivation.

(b) Parse tree:



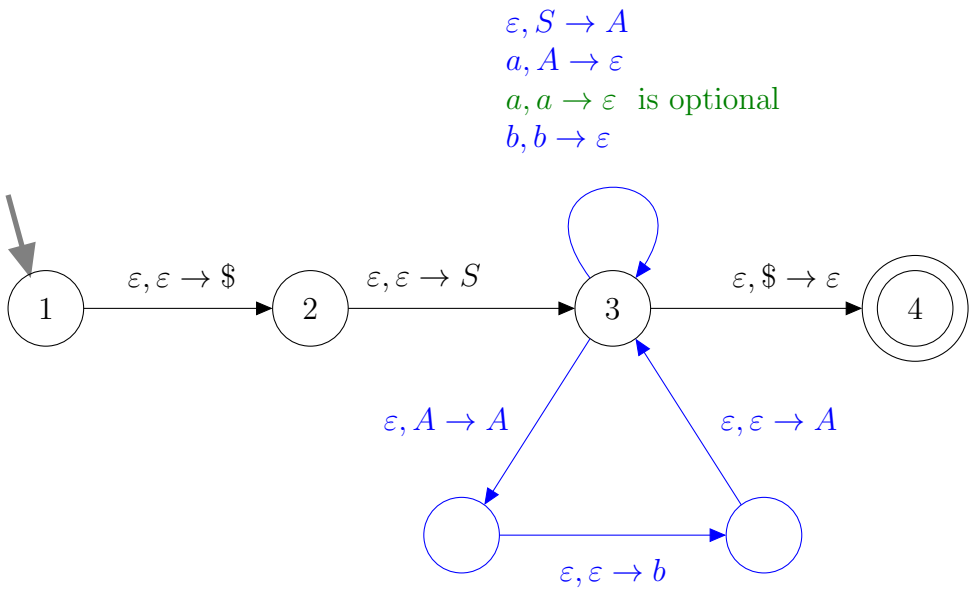
5

Question 11 **(5 marks)**

Given the following Context-Free Grammar:

$$S \rightarrow A \quad (1)$$
$$A \rightarrow AbA \quad (2)$$
$$A \rightarrow a \quad (3)$$

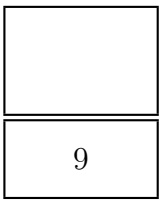
complete the following diagram to give a Pushdown Automaton for the language generated by the grammar.



Question 12 **(4 marks)**

State two important results that can be proved using the Chomsky Normal Form for Context-Free Grammars.

1. The CYK algorithm, which determines, for any CFG and any string, whether the string is generated by the CFG.
2. The Pumping Lemma for CFLs: for any context-free language L and any sufficiently long string $w \in L$, there exist u, v, x, y, z such that $w = uvxyz$, v and y are not both empty, $|vxy|$ is small enough, and for all i, $uv^ixy^iz \in L$.

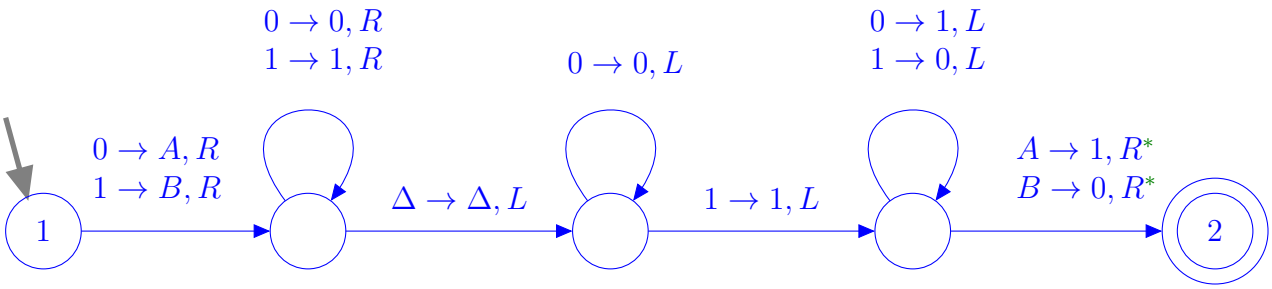


Question 13 **(6 marks)**

The **2's complement** of a binary string is formed as follows. Flip each bit (i.e., change 0 to 1, and 1 to 0) until you get to the last 1. Keep that last 1, and all 0s after it, unchanged.

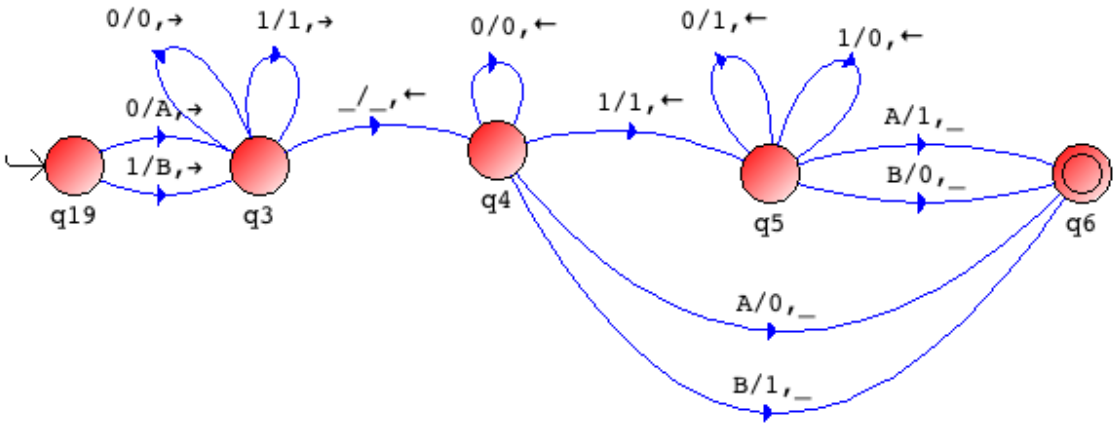
For example, if the string is 0110100, then its 2's complement is 1001100.

Draw a Turing machine to compute the 2's complement of any binary string.

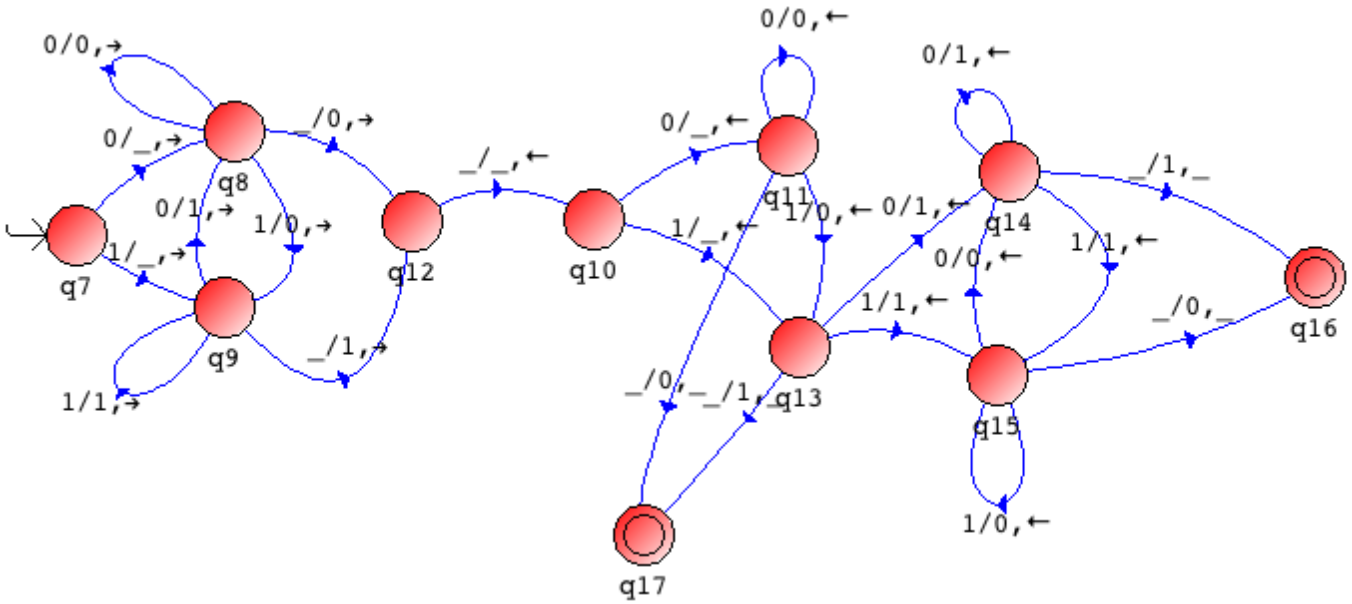


* Allow any direction here.

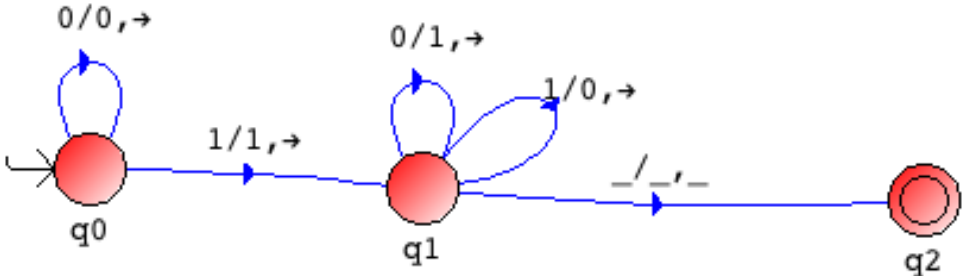
Here is the above in Tuatara, with some extra necessary arcs:



The following monstrous solution has the virtue of not introducing any extra characters into the tape alphabet, although that's not expected in this question:



Here is a solution working from the other end:



Question 14 **(4 marks)**

(a) State the Church-Turing thesis.

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

(b) Give two reasons why the Church-Turing thesis is widely accepted.

Any two of the following:

- different approaches to defining computability end up being equivalent
- long experience, that algorithms can be implemented as programs, and therefore on Turing machines
- no known counterexamples, i.e., no algorithms which seem to be unimplementable

Question 15 **(4 marks)**

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

Decision Problem	your answer (tick **one** box in each row)	
Input: a Turing machine M. Question: Does M eventually halt, if the input is the number 17?	☐ Decidable	☑ Undecidable
Input: a Turing machine M, and a string w. Question: If M is run with input w, does it halt in at most 17 steps?	☑ Decidable	☐ Undecidable
Input: a Turing machine M. Question: Does M have at least 17 states?	☑ Decidable	☐ Undecidable
Input: a Turing machine M. Question: Is the language recognised by M finite?	☐ Decidable	☑ Undecidable

8

Question 16 **(10 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, 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.

(a) $\{\mathtt{a}^n\mathtt{b}^n : n > 0\}$

(b) $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n > 0\}$

(c) $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n : n > 0\}$

(d) The set of all graphs containing a Hamiltonian circuit.

(e) The set of all graphs containing no circuit.

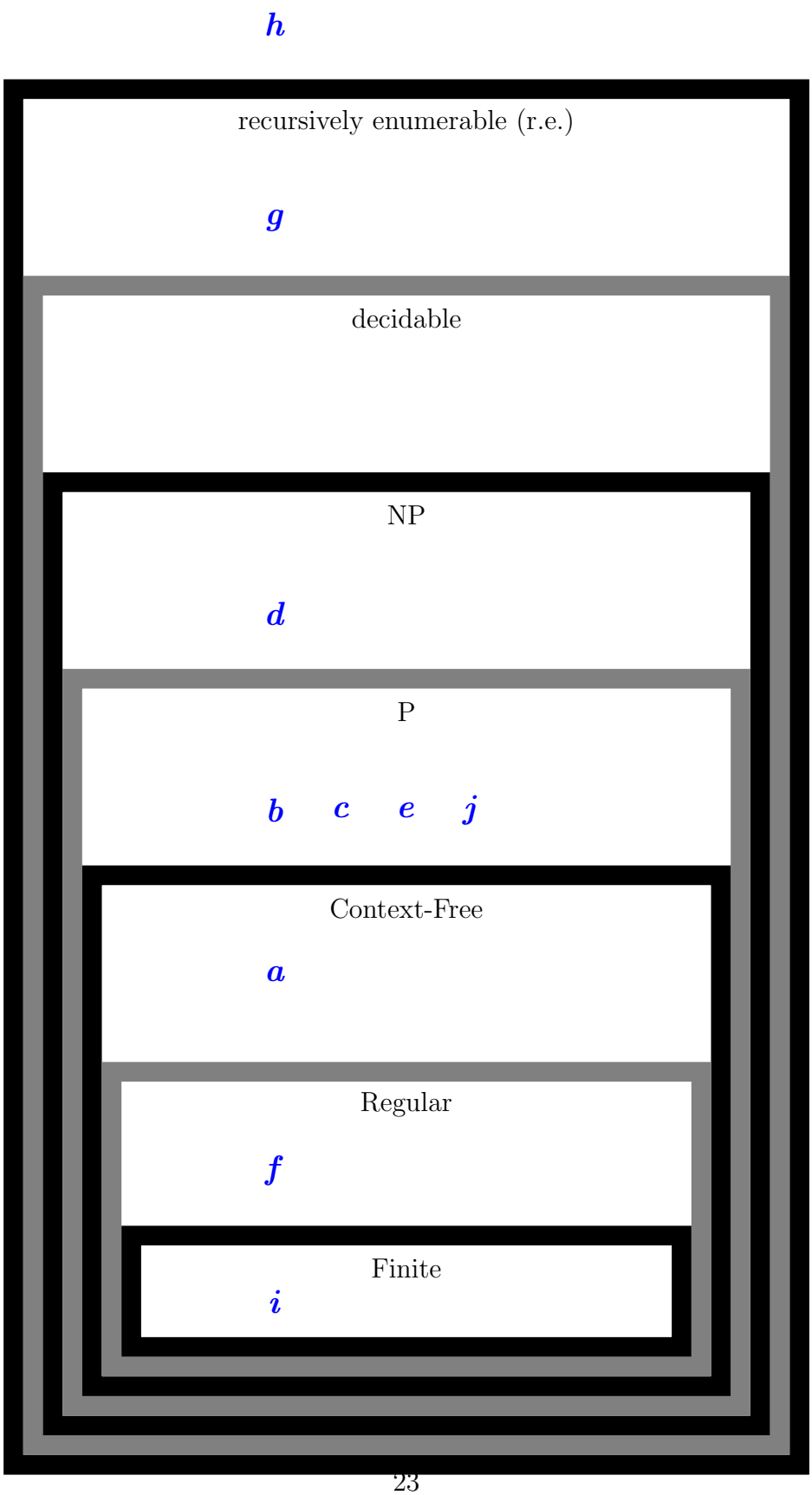
(f) The set of all correctly formed arithmetic expressions that only use positive integers and the symbols + and −.

(g) The set of all (encodings of) Turing machines that eventually halt, when given themselves as input.

(h) The set of all (encodings of) Turing machines that loop forever, when given themselves as input.

(i) The set of all strings that have ever caused any real computer to eventually halt.

(j) The set of all Context-Free Grammars that define an empty language.


h
recursively enumerable (r.e.)
g
decidable
NP
d
P
b c e j
Context-Free
a
Regular
f
Finite
i

Question 17 **(7 marks)**

Prove that the following problem is undecidable.

Input: a Turing machine M, and a string x.
Question: If M is run on input x, does it eventually **accept** x?

You may use the fact that the Halting Problem is undecidable.

Let M, x be any input to the Halting Problem, where we want to know whether or not M eventually halts when its input is x.

From M, x, we construct the following program, which we call $P_{M,x}$ (since we get a different program for each M, x, since M, x are hard-coded into the program):

$P_{M,x}$:

```
Input: anything.
Simulate the execution of M on input x.
Accept.
```

If M halts for input x, then when $P_{M,x}$ is run, on any input at all, it will first simulate M with input x, and after that simulation halts (which it must, eventually, in this case), it will Accept.

On the other hand, if M does not halt for input x, then $P_{M,x}$ will just get stuck in its simulation of M with input x, so it will never Accept.

So, M halts for input x if and only if $P_{M,x}$ eventually accepts its input. (And this holds no matter what input is given to $P_{M,x}$.)

Therefore, if we have a decider, D, for the problem given in the question, then we can use it to construct a decider H for the halting problem, which works as follows.

Suppose M, x is input to the halting problem. Compute the program $P_{M,x}$. (This is possible, as constructing the program $P_{M,x}$ from M, x is computable.) Pick any input for $P_{M,x}$ that you like. Use D to decide whether or not $P_{M,x}$ accepts that input. If it does, then the answer to the halting problem is Yes; if it doesn't, the answer to the Halting Problem is No. So we have a decider for the halting problem.

This contradicts the undecidability of the Halting Problem. So the assumed decider, D, cannot exist. So the problem given in the question is undecidable.

7

Question 18 **(5 marks)**

Let L be a recursively enumerable (r.e.) language. We saw in lectures that there is an enumerator that enumerates L. Here is an attempt at constructing an "enumerator" for L.

Let M be a Turing machine whose set of accepted strings is L. Construct another Turing machine E which does the following.

For each string $w = \varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{ab}, \ldots$, in turn (i.e., sequentially):
{
    Simulate the execution of M on input w.
    If M accepted w, then output w.
    Continue to the next iteration.
}

(a) What is wrong with this attempt at an enumerator?

It is quite possible that, for some specific string $w \notin L$, the machine M might never halt. Then the above enumerator just gets stuck, "hanging" in the middle of that iteration, and it never gets to examine any string w that comes later in the ordering of the strings. So those later strings can never be output by the enumerator. So the enumerator cannot enumerate all strings in L (unless L is finite).

(b) Indicate briefly what would need to be done to fix it. (You don't need to say *in detail* how to fix it, but just indicate in general terms what to do.)

The simulated execution of M on each string w needs to be split up so that, in each main iteration, you just simulate one step of the execution of the machine on each of several strings w. By enlarging, at each main iteration, the set of strings considered, you can ensure that all the required simulation is done eventually, in parallel.

5

Question 19 **(4 marks)**

Suppose M is a Turing machine with time complexity $O(n^3)$, and that U is a Universal Turing Machine that can simulate any t-step Turing machine computation in at most t^4 steps.

Find an upper bound, in big-O notation and with proof, of the time taken by U to simulate the computation by M on an input of size n.

Let x be an input string, of length n, for M. Then there is a constant c such that, provided n is sufficiently large, the time (number of steps) taken by M for input x is $\leq cn^3$.

These steps can be simulated by U in $\leq (cn^3)^4$ steps, i.e., in $\leq c^4n^{12}$ steps. Since c^4 is a constant, this means that the time taken by the simulation is $O(n^{12})$.

4

Question 20 **(4 marks)**

Prove that the class of regular languages is a subset of the class P.

Let L be a regular language. Then, by Kleene's Theorem, there is a FA which recognises L. We can turn this into a Turing machine (or a computer program) that keeps track of which state the FA is in, at any time, and depending on that state, and what the next letter is, determines what the next state is. This TM (or program) looks at each letter of the input string, in turn, and the work it does for that letter — checking what it is and changing its record of the current state accordingly — can be done in only a few steps (constant time or linear time, depending on how the information is organised and managed). Multiply this by the number of letters in the string, and you get an overall time complexity of quadratic time or better. So it takes polynomial time, so L is in P. Hence every regular language is in P.

Question 21 **(2 marks)**

Suppose that a particular decider has polynomial time complexity. When inputs are sufficiently large, what happens to its running time when the length of the input is doubled?

Choose **one** of the answers (a)–(g) below, by circling the appropriate letter.

If more than one answer holds, you must choose the strongest (i.e., most precise) correct answer.

(a) It is raised to at most a constant power.

(b) It is increased by at most a constant factor.

(c) It is increased by at most a constant amount.

(d) It is at most squared.

(e) It is at most doubled.

(f) It increases by at most 2.

(g) It doubles after two years, according to Moore's Law.

6

Question 22 **(4 + 6 + 7 + 2 + 2 = 21 marks)**

Consider the language NO MONOCHROMATIC TRIANGLE, which consists of all graphs G such that we can assign colours to the edges of the graph so that (i) each edge is either Black or White, and (ii) there is no *triangle* (i.e., cycle of length 3) in the graph which is *monochromatic* (i.e., all its edges have the same colour).

(a) Prove that the language NO MONOCHROMATIC TRIANGLE is in NP.

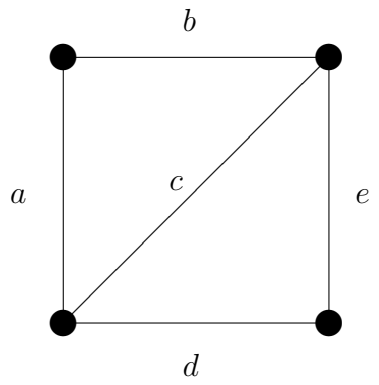
Here is a verifier:

> Input: graph G.
> Certificate: a 2-colouring of the edges of G.
> [This can be described in various ways. E.g., as an assignment of either Black or White to each edge of G, or a list of colours, Black/White, one colour for each edge, or a partition of the set of edges into two parts, etc.]
> Verification:
> For each triangle of G:
> {
> if the three edges of the triangle all have the same colour, Reject and Stop.
> }
> Accept and Stop.
> [If we reach this point, each triangle must have been *non*-monochromatic.]

It should be clear from the construction of the verifier that it will accept if and only if the input graph belongs to NO MONOCHROMATIC TRIANGLE.

It remains to show that this verifier runs in polynomial time. To see this, observe that a graph on n vertices has $O(n^3)$ triangles, so the number of outer-loop iterations is $O(n^3)$. For each such triangle, we must check that the three edges do not all receive the same colour. The exact amount of time taken by this depends on the details of the implementation (what data structure is used for the graph, etc.), but in any case takes polynomial time, and should take constant or linear time. Multiplying this by $O(n^3)$ gives a total time which is polynomial in the input size. So the verifier is a polynomial-time verifier. So NO MONOCHROMATIC TRIANGLE has a polynomial-time verifier, hence it is in NP.

Now, let W be the following graph.



(b) Construct a Boolean expression E_W in Conjunctive Normal Form such that the satisfying truth assignments for E_W correspond to solutions to the NO MONOCHROMATIC TRIANGLE problem on the above graph W (i.e., they correspond to colourings of the edges of W which have no monochromatic triangle).

Using variables a, b, c, d, e, with True meaning Black and False meaning White (or the other way round):

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

An alternative approach is to use variables $B_a, W_a, B_b, W_b, B_c, W_c, B_d, W_d, B_e, W_e$, with the interpretation that, for any edge x in the graph,

$$\begin{aligned} B_x &= \text{True if edge } x \text{ is Black, and False otherwise;} \\ W_x &= \text{True if edge } x \text{ is White, and False otherwise.} \end{aligned}$$

This is more cumbersome. The expression should then be:

$$\begin{aligned} &(B_a \vee B_b \vee B_c) \wedge (W_a \vee W_b \vee W_c) \wedge (B_c \vee B_d \vee B_e) \wedge (W_c \vee W_d \vee W_e) \wedge \\ &(B_a \vee W_a) \wedge (\neg B_a \vee \neg W_a) \wedge (B_b \vee W_b) \wedge (\neg B_b \vee \neg W_b) \wedge (B_c \vee W_c) \wedge (\neg B_c \vee \neg W_c) \wedge \\ &(B_d \vee W_d) \wedge (\neg B_d \vee \neg W_d) \wedge (B_e \vee W_e) \wedge (\neg B_e \vee \neg W_e) \end{aligned}$$

The first four clauses are clearly analogous to the four clauses of the earlier expression. The last ten clauses are to ensure that each edge gets precisely one colour.

(c) Give a polynomial-time reduction from NO MONOCHROMATIC TRIANGLE to SATISFIABILITY.

Input: Graph G.
For each edge of G, create a new variable.
For each triangle T of G:
{
    Let a, b, c be the edges of T.
    Create two new clauses: $a \vee b \vee c$, $\neg a \vee \neg b \vee \neg c$.
    [Some might use the more cumbersome set of clauses,
    but that's fine if done right.]
}
Output: the conjunction of all the clauses created so far.

(d) State the Cook-Levin Theorem.

SATISFIABILITY is NP-complete.

(e) Given the facts stated so far in this question:
What else, if anything, would you need to prove, in order to show that NO MONOCHROMATIC TRIANGLE is NP-complete?

SAT $\leq_P$ NO MONOCHROMATIC TRIANGLE.

Reasoning, though the question doesn't ask for it:
We know SAT is NP-complete, and we know NO MONOCHROMATIC TRIANGLE is in NP. So, if we show that SAT $\leq_P$ NO MONOCHROMATIC TRIANGLE, then we have completed the proof that NO MONOCHROMATIC TRIANGLE is NP-complete.

END OF EXAMINATION