The University of Nottingham

SCHOOL OF COMPUTER SCIENCE
AND INFORMATION TECHNOLOGY

A LEVEL 1 MODULE, SPRING SEMESTER 2004-2005

MACHINES AND THEIR LANGUAGES

Time allowed TWO hours

Candidates must NOT start writing their answers until told to do so

Answer QUESTION ONE and any THREE other questions

Marks available for sections of questions are shown in brackets in the right-hand margin.

No calculators are permitted in this examination.

Dictionaries are not allowed with one exception. Those whose first language is not English may use a dictionary to translate between that language and English provided that neither language is the subject of this examination.

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn examination paper over until instructed to do so

Question 1 (Compulsory)
The following questions are multiple choice. There is at least one correct choice but there may be several. To get all the marks you have to list all the correct answers and none of the wrong ones.

Note: The answers below provide some explanations, mainly for the incorrect alternatives. This is just for clarification. The answer should just be a list of alternatives.

(a) Which of the following statements are correct?

(i) An alphabet is a finite set of symbols.

(ii) A word is a possibly infinite sequence of symbols over a given alphabet.

(iii) A language is the set of all possible words over a given alphabet.

(iv) A regular language is always finite.

(v) A finite language is always regular.

(5)

Correct: i, v

Incorrect:

ii A word must be finite.

iii A language is a subset *of the possible words.*

iv Σ^* *is an example of an infinite regular language for any non-empty alphabet* Σ*.*

(b) Which of the following statements are correct?

(i) The empty word ϵ belongs to all languages.

(ii) The empty word ϵ is the only word in the empty language $\emptyset$.

(iii) $\epsilon \in \{a\}^*$

(iv) $\epsilon \in \emptyset^*$

(v) If L is a language containing at least one non-empty word, then L^* is an infinite language.

(5)

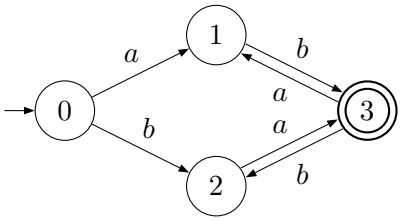
Correct: iii, iv, v

Incorrect:

i A language is an arbitrary subset of the possible words over a given alphabet. This may or may not include the empty word.

ii The empty language contains no words, not even the empty one.

(c) Consider the following finite automaton A over $\Sigma = \{a, b\}$:



Which of the following statements about A are correct?

(i) The automaton A is a Deterministic Finite Automaton (DFA).

(ii) $\epsilon \in L(A)$

(iii) $abba \in L(A)$

(iv) All words accepted by A contain equally many a's and b's.

(v) The automaton A accepts all non-empty words over Σ that contain equally many a's and b's.

(5)

Correct: iii, iv

Incorrect:

i NFA since the transition function is not total for the states 1 and 2.

ii State 0 is not an accepting state (and there is no way to get to an accepting state on the empty input).

v No, it does not accept aabb, for example. (The automaton accepts $(\mathbf{ab} + \mathbf{ba})(\mathbf{ab} + \mathbf{ba})^*$*.)*

(d) Consider the following set W of words:

$$W = \{\epsilon,\ ab,\ cab,\ abab\}$$

Which of the following regular expressions denote a language that contains *all* words in W? (But not necessarily *only* the words in W: it is OK if the language denoted by the regular expression contains *more* words.)

(i) $(\epsilon + \mathbf{ab} + \mathbf{c})(\epsilon + \mathbf{ab})$

(ii) $(\epsilon + \mathbf{ab} + \mathbf{c})(\emptyset + \mathbf{ab})$

(iii) $(\epsilon + \mathbf{ab} + \mathbf{c})^*$

(iv) $(\mathbf{ab} + \mathbf{c})^*$

(v) $(\mathbf{ab}\emptyset + \mathbf{c})^*$

(vi) $(\mathbf{ab})^* + \mathbf{c}^*$

(5)

Correct: i, iii, iv

Incorrect:

- *ii The empty word ϵ is not in the language. To see this, recall that concatenation with the empty* language *yields the empty language and simplify:*

$$\begin{aligned}(\epsilon + \mathbf{ab} + \mathbf{c})(\emptyset + \mathbf{ab}) &= (\epsilon + \mathbf{ab} + \mathbf{c})\emptyset + (\epsilon + \mathbf{ab} + \mathbf{c})\mathbf{ab} \\ &= (\epsilon + \mathbf{ab} + \mathbf{c})\mathbf{ab} \\ &= \mathbf{ab} + \mathbf{cab} + \mathbf{abab}\end{aligned}$$

- *v Concatenation with the empty language yields the empty language, thus the r.e. can be simplified as follows:*

$$(\mathbf{ab}\emptyset + \mathbf{c})^* = (\emptyset + \mathbf{c})^* = \mathbf{c}^*$$

vi The word cab is not in the language.

(Six rather than five alternatives is intentional.)

(e) Consider the following Context-Free Grammar (CFG) G:

$$\begin{array}{rcl} S & \rightarrow & X \mid YC \\ X & \rightarrow & aXc \mid B \\ Y & \rightarrow & aYb \mid \epsilon \\ B & \rightarrow & bB \mid \epsilon \\ C & \rightarrow & cC \mid \epsilon \end{array}$$

S, X, Y, B, C are nonterminal symbols, S is the start symbol, and a, b, c are terminal symbols.

Which of the following statements about the language $L(G)$ generated by G are correct?

(i) $aabbcc \in L(G)$
(ii) $\{a^i b^j c^k \mid i, j, k \in \mathbb{N}\} \subseteq L(G)$
(iii) $\{a^i b^i c^j \mid i, j \in \mathbb{N}\} \subseteq L(G)$
(iv) $\{a^i b^i c^i \mid i \in \mathbb{N}\} \subseteq L(G)$
(v) L(G) $= \{a^i b^i c^i \mid i \in \mathbb{N}\}$

(5)

The language denoted by the CFG is

$$L(G) = \{a^i b^j c^i\} \cup \{a^i b^i c^j\}$$

Thus, correct: i, iii, iv

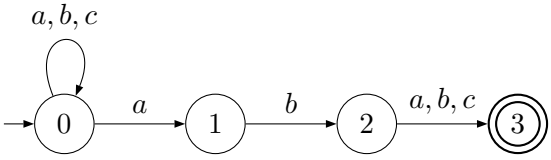
Incorrect:

ii E.g. abbccc is not in L(G).

v E.g. aabbccc is *in L(G).*

Question 2

(a) Given the following NFA N over the alphabet $\Sigma = \{a, b, c\}$, construct a DFA $D(N)$ that accepts the same language as N by applying the *subset construction*:

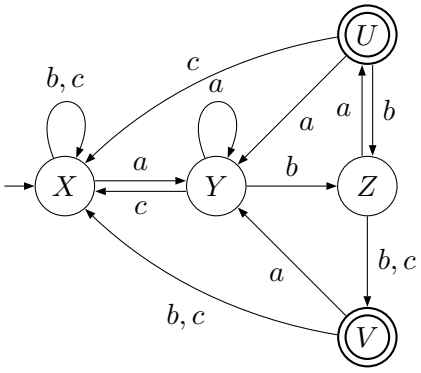


To save work, consider only the *reachable* part of $D(N)$. Clearly show your calculations, e.g. in a state-transition table. Do not forget to indicate the initial state and the final states of the resulting DFA $D(N)$. (12)

The DFA states are sets of NFA states. Any DFA state containing an accepting NFA state is accepting. → marks an initial state, ∗ a final one.

	$\delta_{D(N)}$	a	b	c
→	$\{0\} = X$	$\{0,1\}$	$\{0\}$	$\{0\}$
	$\{0,1\} = Y$	$\{0,1\}$	$\{0,2\}$	$\{0\}$
	$\{0,2\} = Z$	$\{0,1,3\}$	$\{0,3\}$	$\{0,3\}$
∗	$\{0,1,3\} = U$	$\{0,1\}$	$\{0,2\}$	$\{0\}$
∗	$\{0,3\} = V$	$\{0,1\}$	$\{0\}$	$\{0\}$

Now we can draw the transition diagram:



(b) Construct a finite automaton (DFA or NFA) that accepts the language of *correct* binary (base 2) additions (and no other strings) according to the following. We consider addition of two binary numbers of equal but arbitrary length. For example:

$$\begin{array}{r} 1001 \\ +0011 \\ \hline 1100 \end{array}$$

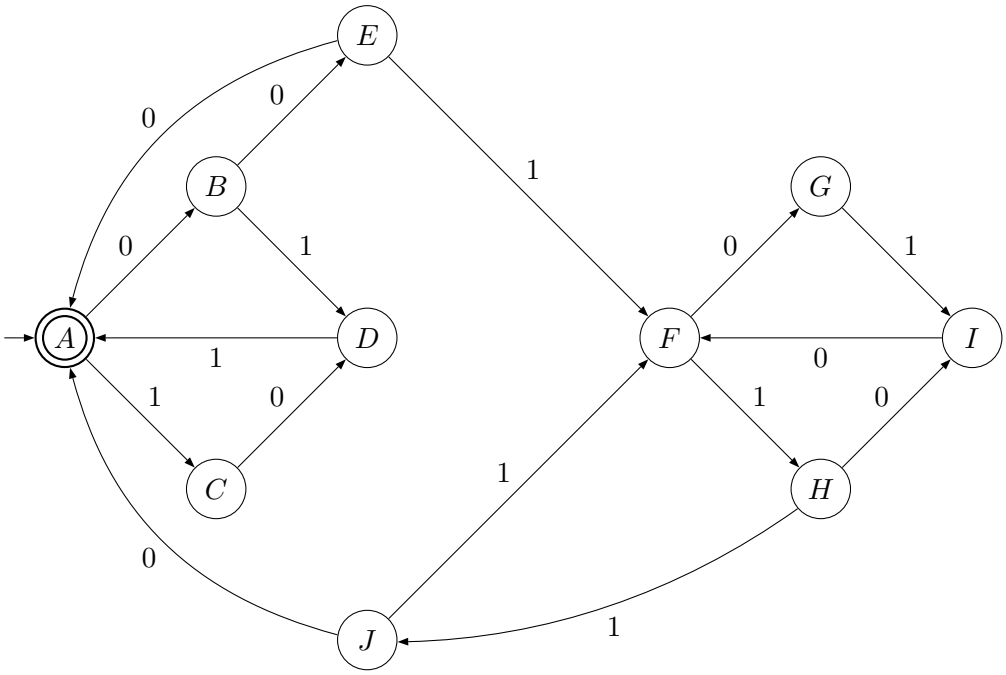
We represent an addition by the string of binary digits (0 or 1) obtained by reading the digits top-down, column by column, from left to right. The example above is thus represented by the string 101001010110. We assume that the summands have been padded with 0's to the left to make them as long as the result, if necessary. For example, the addition

$$\begin{array}{r} 01 \\ +11 \\ \hline 100 \end{array}$$

is represented by the string 001010110. To simplify the problem slightly, the empty string ϵ is considered representing a correct binary addition (two zero-length summands yield a zero-length result). (13)

Key idea: When we encounter a digit from the sum, it will be clear whether or not the addition in the next *column must result in a carry if the addition is to be correct. If not, we just go back to the start state (which is accepting). Otherwise, we go to an "expect carry" state which is like the start state, except that it is not accepting and that what constitutes a correct sequence of digits is adjusted to reflect the expected carry.*

The following transition diagram represents a 10-state NFA that implements this idea. A is the initial and only accepting state. F is the "expect carry state".



Question 3

(a) Classify the following languages as regular, context-free, or neither. Justify your answer by providing, where possible and reasonable, a regular expression or a context-free grammar denoting the language in question. Otherwise justify by giving a short (informal but convincing) argument.

(i) All words over $\Sigma = \{a, b, c\}$ in which every a is immediately followed by a b.

(ii) All words over $\Sigma = \{a, b, c\}$ in which every a is eventually followed by a b.

(iii) $\{a^i b^i c^j d^j \mid i, j \in \mathbb{N}\}$

(iv) $\{a^i b^j c^i d^j \mid i, j \in \mathbb{N}\}$

(v) All legal sequences of moves in the game of Chess.

(10)

(i) Regular: $(\mathbf{ab} + \mathbf{b} + \mathbf{c})^*$

(ii) Regular: $(\mathbf{a}(\mathbf{a} + \mathbf{c})^*\mathbf{b} + \mathbf{b} + \mathbf{c})^*$

(iii) Context free:

$$\begin{array}{lcl} S & \rightarrow & AB \\ A & \rightarrow & aAb \mid \epsilon \\ B & \rightarrow & cBd \mid \epsilon \end{array}$$

(iv) Not context free (and thus not regular). A production for guaranteeing the balance between the a's and the c's will necessarily look something like $A \to aAc \mid B$. But once the B is reached, there is no way to enforce that the number of b's matches the number of d's that follow the string derived from A. The situation is similar if one sets out to initially maintain the balance between the b's and the d's.

Another argument: a PDA has only one stack and thus cannot count two different things simultaneously and independently.

(v) Regular since the number of chess states (ways of placing pieces on the board) is finite and since the legal moves is uniquely determined by the current state.

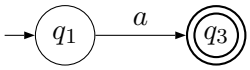
(b) Systematically construct an NFA accepting the language denoted by the following regular expression by following the graphical construction described in the lectures/lecture notes:

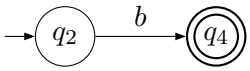
$$(\mathbf{a} + \mathbf{b})^*(\mathbf{c} + \mathbf{d})$$

The alphabet is $\Sigma = \{a, b, c, d\}$.

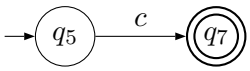
Your answer should clearly show what you are doing. In particular, in addition to the final NFA, the answer should include at least two intermediate stages of the construction. However, states only have to be named in the final NFA. Also, feel free to tidy up the final NFA by removing "dead ends", but be sure to explain what you are doing.

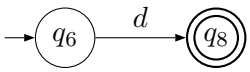
(5) *Work structurally, from the smallest constituent subexpressions towards to overall regular expression. Since they are simple enough, we start with NFAs for* $(\mathbf{a} + \mathbf{b})$*:*



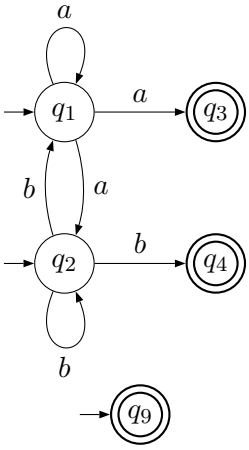


and $(\mathbf{c} + \mathbf{d})$*:*

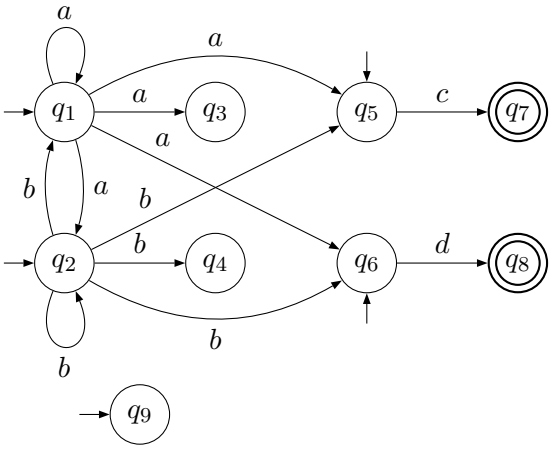




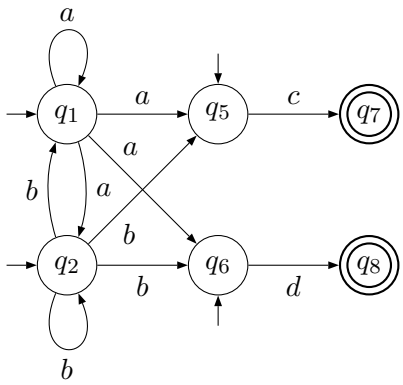
Then form an NFA for $(\mathbf{a} + \mathbf{b})^*$*:*



Concatenate the NFAs for $(\mathbf{a} + \mathbf{b})^*$ *and* $(\mathbf{c} + \mathbf{d})$*, keeping in mind that there is a start state that is also final in the first NFA (since* ϵ *belong to the language), meaning that the* initial states *of the second NFA are kept as initial states. Don't forget to change the accepting states of first the NFA to non-accepting ones:*



Finally, remove "dead ends":



(c) Use the Pumping Lemma for regular languages to show that the following language is not regular:

$$\{a^i b^j c^k \mid i, j, k \in \mathbb{N},\ k = \min(i, j)\} \tag{10}$$

Call the given language L. Assume it is regular. Then, according to the pumping lemma for regular expressions, there is a constant n such that any string $w \in L$ which has length at least n ($|w| \geq n$) can be divided into three parts $w = xyz$ as follows:

1. $|xy| \leq n$
2. $|y| > 0$
3. $xy^i z \in L$ *for any natural number* i

Consider a string $w = a^n b^n c^n$. As $\min(n, n) = n$, we clearly have $w \in L$. Moreover, the length of w is $|w| = 3n \geq n$. The pumping lemma for regular languages therefore applies, and our w, as any sufficiently long string in the language, can be divided into three parts $w = xyz$ accordingly.

Since $|xy| \leq n$, it must be the case that $y = a^k$ for $0 < k \leq n$ due to the way w was chosen and condition 2 on the division into parts.

Now, consider condition 3. It should hold for any i. Pick $i = 0$ for example. $xy^0 z = a^{(n-k)} b^n c^n$. Since $k > 0$, $n - k < n$. Therefore $\min(n - k, n) = n - k < n$. But the string $xy^0 z$ has n c's. That is too many, and thus it cannot belong to L. We have a contradiction, and thus our initial assumption that L is regular must be wrong. Thus L is not regular, QED.

Question 4

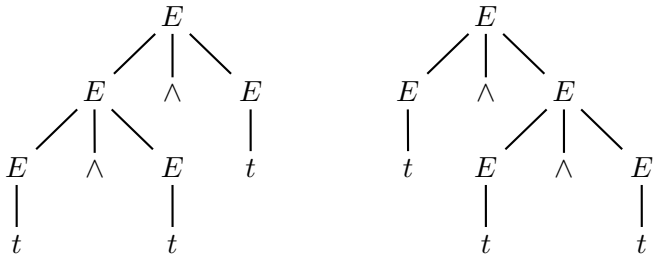
(a) The following is a context-free grammar (CFG) for Boolean expressions:

$$\begin{array}{rcl} E & \rightarrow & E \wedge E \\ & | & E \vee E \\ & | & \neg E \\ & | & (E) \\ & | & t \\ & | & f \end{array}$$

E is a nonterminal and the start symbol, $\wedge$, $\vee$, $\neg$, (,), t, and f are terminals.

Show that this grammar is ambiguous. (5)

A CFG is ambiguous if at least one word in the described language has more than one parse tree. To show that a grammar is ambiguous pick a word in the language that has two parse trees and show these two trees. For the given language, $t \wedge t \wedge t$ is a word that has two parse trees:



An equivalent way is to show that the word in question either has two leftmost *derivations or two* rightmost *derivations. It is* NOT *enough to just show two different derivations, as merely permuting the order in which non-terminals are expanded does not affect the structure of the corresponding parse tree. Here are two different leftmost derivations. The first one, corresponding to the first tree:*

$$\begin{array}{rcl} E & \underset{\text{lm}}{\Rightarrow} & E \wedge E \\ & \underset{\text{lm}}{\Rightarrow} & E \wedge E \wedge E \\ & \underset{\text{lm}}{\Rightarrow} & t \wedge E \wedge E \\ & \underset{\text{lm}}{\Rightarrow} & t \wedge t \wedge E \\ & \underset{\text{lm}}{\Rightarrow} & t \wedge t \wedge t \end{array}$$

The second one, corresponding to the second tree:

$$\begin{aligned} E &\underset{\text{lm}}{\Rightarrow} E \wedge E \\ &\underset{\text{lm}}{\Rightarrow} t \wedge E \\ &\underset{\text{lm}}{\Rightarrow} t \wedge E \wedge E \\ &\underset{\text{lm}}{\Rightarrow} t \wedge t \wedge E \\ &\underset{\text{lm}}{\Rightarrow} t \wedge t \wedge t \end{aligned}$$

(b) Construct an *unambiguous* version of the context-free grammar for Boolean expressions given above by making it reflect the following operator *precedence* conventions:

- $\neg$ has the highest precedence
- $\wedge$ has the next highest precedence
- $\vee$ has the lowest precedence

For example, $t \vee \neg f \wedge t$ should be interpreted as $t \vee ((\neg f) \wedge t)$. As long as the grammar is unambiguous, you can choose whether or not to accept expressions that would need conventions about operator *associativity* to disambiguate them, like $t \vee t \vee t$. (10)

Here is a version that assumes that the binary operators are non-associative. (Thus the language accepted is not quite the same as for the ambiguous grammar. But that's OK according to the problem statement.)

$$\begin{aligned} E &\rightarrow E_1 \vee E_1 \mid E_1 \\ E_1 &\rightarrow E_2 \wedge E_2 \mid E_2 \\ E_2 &\rightarrow \neg E_2 \mid (\, E \,) \mid t \mid f \end{aligned}$$

The problem does not state whether using many logical negations immediately after one another should be OK or not (e.g. $\neg\neg\neg t$). The above grammar does allow that. The following grammar does not:

$$\begin{aligned} E &\rightarrow E_1 \vee E_1 \mid E_1 \\ E_1 &\rightarrow E_2 \wedge E_2 \mid E_2 \\ E_2 &\rightarrow \neg E_3 \mid E_3 \\ E_3 &\rightarrow (\, E \,) \mid t \mid f \end{aligned}$$

The grammar is unambiguous either way, so both versions are fine.

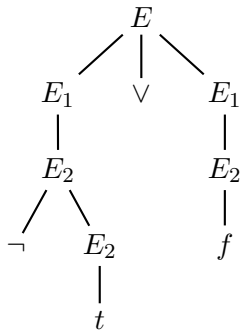
(c) Draw the derivation trees according to your unambiguous grammar for the following two expressions:

(i) $\neg t \vee f$

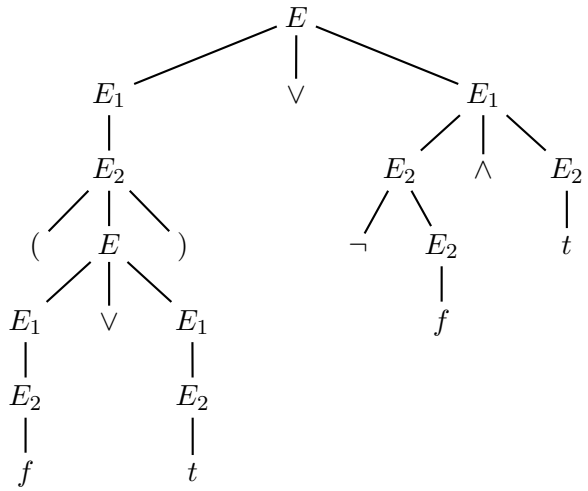
(ii) $\neg(f \vee t) \vee \neg f \wedge t$

(5)

Parse trees according to the first grammar above. Parse tree for $\neg t \vee f$*:*



Parse tree for $\neg(f \vee t) \vee \neg f \wedge t$*:*



(d) The binary operators $\wedge$ and $\vee$ can be considered to be:

- *left-associative*; i.e. an expression like $t \vee t \vee t$ would be interpreted as $(t \vee t) \vee t$
- *right-associative*; i.e. an expression like $t \vee t \vee t$ would be interpreted as $t \vee (t \vee t)$
- *non-associative*; i.e. ruling out expressions like $t \vee t \vee t$

Explain what is the case for your grammar and why, and how to change your grammar for the other possibilities. (5)

Left-associative: make the productions for the binary operators left recursive:

$$\begin{aligned} E &\rightarrow E \vee E_1 \\ E_1 &\rightarrow E_1 \wedge E_2 \end{aligned}$$

Right-associative: make the productions for the binary operators right recursive:

$$\begin{aligned} E &\rightarrow E_1 \vee E \\ E_1 &\rightarrow E_2 \wedge E_1 \end{aligned}$$

Non-associative: do not make the productions for the binary operators directly recursive, as in the original grammar.

Question 5

Consider the following Pushdown Automaton (PDA) P:

$$P = (Q = \{q_0, q_1\},\ \Sigma = \{a, b, c\},\ \Gamma = \{a, \#\},\ \delta,\ q_0,\ Z_0 = \#,\ F = \{q_1\})$$

where the transition function δ is given by

$$\begin{aligned} \delta(q_0, a, \#) &= \{(q_0, a\#)\} \\ \delta(q_0, c, \#) &= \{(q_0, \#)\} \\ \delta(q_0, a, a) &= \{(q_0, aa)\} \\ \delta(q_0, b, a) &= \{(q_0, \epsilon)\} \\ \delta(q_0, c, a) &= \{(q_0, a)\} \\ \delta(q_0, \epsilon, \#) &= \{(q_1, \#)\} \\ \delta(q, w, z) &= \emptyset \qquad \text{everywhere else} \end{aligned}$$

Acceptance is by *final state*.

(a) Which of the following words are accepted by the PDA P?

(i) *acabbc*

(ii) *cbcac*

(iii) ϵ

For those words that are accepted, provide a sequence of Instantaneous Descriptions (IDs) leading to an accepting configuration as evidence. For those words that are not accepted, explain why there is no sequence of IDs leading to an accepting configuration. (12)

(i) The word acabbc is accepted. ID sequence:

$$\begin{aligned}(q_0, acabbc, \#) &\vdash (q0, cabbc, a\#)\\ &\vdash (q0, abbc, a\#)\\ &\vdash (q0, bbc, aa\#)\\ &\vdash (q0, bc, a\#)\\ &\vdash (q0, c, \#)\\ &\vdash (q0, \epsilon, \#)\\ &\vdash (q1, \epsilon, \#)\end{aligned}$$

The word is accepted because q_1 is an accepting state and since all input has been read. (Marking: 5 points.)

(ii) The word cbcac is not accepted. On seeing c in state q_0 with # on top of the stack, there are two possibilities. The PDA can either read and discard the c, staying in q_0, or it can move to q_1 without reading the c.

We observe that the machine gets stuck as soon as state q_1 is reached.

Thus, if we want to accept a string starting with a c, the first transition must be to read and discard that c:

$$(q_0, cbcac, \#) \vdash (q_0, bcac, \#)$$

We are now in state q_0, reading a b, with # on top of the stack. The only possibility here is to move to q_1 without reading any input:

$$\vdash (q_1, bcac, \#)$$

But this is a stuck configuration! And since all input has not been read, it is not an accepting configuration.

Thus there are no sequences of IDs leading to an accepting configuration. (Marking: 5 points.)

(iii) The word ϵ is accepted:

$$(q_0, \epsilon, \#) \vdash (q_1, \epsilon, \#)$$

This is an accepting configuration since q_1 is an accepting state and since all input (none!) has been read. (Marking: 2 points.)

(b) Describe the language accepted by P in English in one sentence. (5)

Strings over $\Sigma = \{a, b, c\}$ where a acts as an opening parenthesis, b as a closing parenthesis, and parentheses has to be balanced in the usual fashion.

(c) Explain how to modify P to make it accept by *empty stack* instead of accepting by final state (without changing the accepted language). (2)

One possibility is to replace the equation

$$\delta(q_0, \epsilon, \#) = \delta(q_1, \#)$$

with

$$\delta(q_0, \epsilon, \#) = \delta(q_1, \epsilon)$$

or even

$$\delta(q_0, \epsilon, \#) = \delta(q_0, \epsilon)$$

(getting rid of one state).

Either way, this gives the PDA the possibility to completely empty the stack when it is in a state where the "parentheses" seen so far has balanced out. If the PDA makes that move in a state where all input has been read it will enter an accepting configuration (and block), just as the given PDA does when entering state q_1.

(d) What does it mean for a PDA to be deterministic? State the formal condition and explain what it means. Is P deterministic? Justify your answer! (6)

That the PDA never has any choice. Formally:

$$|\delta(q, x, z)| + |\delta(q, \epsilon, z)| \leq 1$$

for all $q \in Q$, $x \in \Sigma$, and $z \in \Gamma$.

P is not deterministic as it has a choice in state q_0 when # is the top stack symbol for input symbols a and c. Formally, we have e.g.

$$|\delta(q_0, a, \#)| + |\delta(q_0, \epsilon, \#)| = 1 + 1 = 2$$

which is not less than or equal to 1.

Question 6

(a) What is a Turing Machine (TM)? Your account does not have to be formal, but it should be comprehensive and clearly outline the central ideas. (5)

A Turing Machine is a mathematical model of a general computer. It consists of finite control plus an infinite tape for storage of the input, the output, and any other data needed during a computation. The tape is divided into cells, each cell is capable of storing one symbol. A read/write head scans one cell of the tape, and depending on the symbol in that cell and the state of the finite control, the TM updates the scanned symbol with a new symbol (possibly the same as before), moves the head one step left or right, and changes state.

(b) Outline a strategy for constructing a TM that accepts the language

$$\{a^m b^n c^m d^n \mid m, n \in \mathbb{N}\}$$

(5)

If an a is being read, overwrite it with a marking symbol not in the input alphabet, say X, move to the right until a corresponding c is found, but only moving across first a's, then b's, then X's if any, mark that as well, and then move left until we find the first remaining a, and repeat. If at any point a c cannot be found, stop in a non-accepting state.

If there was no a to begin with, or once there are no remaining a's, we do as above for b and d.

If the machine has reached a state where the last remaining a and b has been matched with a corresponding c or d, the TM is in a state where it is scanning left for a remaining a or b. But instead of finding one, it is going to encounter a blank symbol to the left of the portion of the tape where the input initially was written. At this point, the machine starts scanning right. If it only reads X's before the first blank symbol to the right of the portion of the tape where the input was stored is found, the TM moves to an accepting state and stops. Otherwise it stops in a non-accepting state.

(c) Explain and relate the following terms in the context of Turing Machines: *recursive*, *recursively-enumerable*, *decidable*, *undecidable*. (5)

A recursive language is a language that is accepted by a Turing Machine that always halts. This is the same as saying that the language is decidable (or that the problem represented by the language is decidable).

A recursively enumerable language is a language that is accepted by a Turing Machine that does not necessarily halt for input not belonging to the language.

A language that is not recursive is undecidable. The undecidable languages thus includes the recursively enumerable (or semi-decidable) languages and those languages which are not even recursively-enumerable.

(d) What is the *Church-Turing thesis*? Give a brief explanation. (5)

The unproven assumption that any reasonable notion of what "computation" means is equivalent to what a Turing machine can compute.

(e) One could argue that a computer is really a finite state automaton since any computer only has a finite memory. Is this a useful characterization? Provide a good argument for your position. (5)

Arguing either way is fine as long as the argument is good!

Here is my position:

Even a very modest real computer has far too many states for an FA characterization to provide any useful model of its behaviour. If one does not accept that, then note that it would be possible to design and program a computer in such a way that it dynamically can ask for more memory should the need arise (e.g. by making additional storage resources available over a network). Then, given given some time, the computer would always have as much memory as it needs, which for practical purposes is the same as an unbounded amount of memory. Again, a finite automaton is not a useful characterization.

(If one wants to push the argument, one could say that the resources on the Earth are limited, and thus it is conceivable that a program might need more memory than actually could be constructed using the resources available on the Earth. But nothing in principle stops us from moving beyond the Earth. So ultimately what is going to set the limit is what we are prepared/can afford to do to extend the memory, and for how long we can wait for the final answer, not that it is theoretically impossible to extend the memory.)