UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1 — INTRODUCTION TO COMPUTATION

Saturday 1 st April 2017

00:00 to 02:00

INSTRUCTIONS TO CANDIDATES

1. **Note that ALL QUESTIONS ARE COMPULSORY.**
2. **DIFFERENT QUESTIONS MAY HAVE DIFFERENT NUMBERS OF TOTAL MARKS. Take note of this in allocating time to questions.**
3. **CALCULATORS MAY NOT BE USED IN THIS EXAMINATION.**

You can complete this takehome paper in your own time.

You may find it helpful to set aside two hours,
and complete it under your own mock exam conditions.

You can then review your work, and adjust your revision planning accordingly.

The real thing will be a two hour, closed book examination,
currently scheduled as follows:

The Pleasance Sports Hall,
on Monday, 10th December 2018, 14:30 to 16:30

Changes can occur to the timetable therefore please ensure that you recheck
your exam details immediately prior to your examination.

THIS TAKEHOME EXAMINATION WILL BE MARKED ONLY BY YOU

The actual exam will be marked anonymously.

1. In this question we use $\oplus$ for exclusive or, whose truth table is defined by $A \oplus B \equiv (\neg A \wedge B) \vee (A \wedge \neg B)$

   (a) Is $\oplus$ associative? Justify your answer. [*4 marks*]

   (b) Use Karnaugh maps or Boolean algebra to derive a CNF equivalent to $R \leftrightarrow (A \oplus B)$ [*4 marks*]

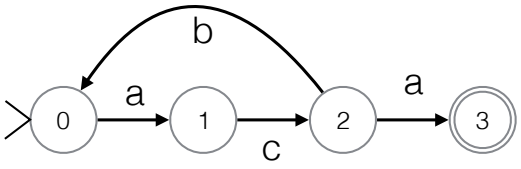
   (c) Use Karnaugh maps or Boolean algebra to derive a CNF equivalent to $R \leftrightarrow (A \rightarrow B)$ [*4 marks*]

   (d) Use the Tseytin procedure to give a CNF equisatisfiable with $A \oplus (B \rightarrow C)$ [*4 marks*]

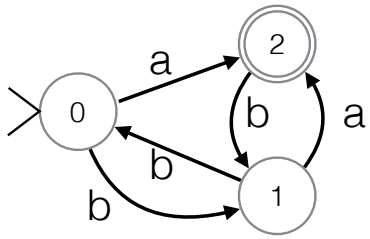
   (e) Use the Tseytin procedure to give a CNF equisatisfiable with $\neg A \rightarrow (B \oplus C)$ [*4 marks*]

2. For each of the following finite state machines (FSM), give a regex matching exactly the strings recognised the machine. Say whether the machine is deterministic; if it is not, construct an equivalent DFA, and label the states of your DFA to indicate how they were constructed. [*20 marks*]

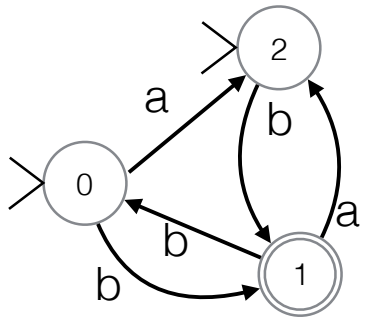
(a)



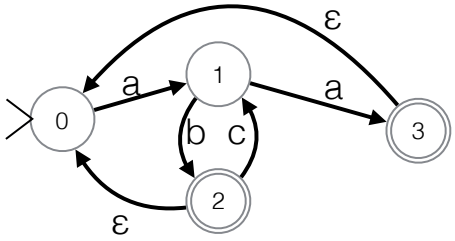
(b)



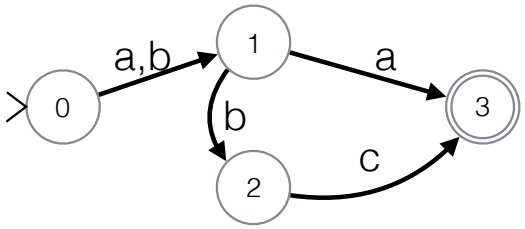
(c)



(d)



(e)



3. This question concerns finite state machines and regular expressions. You should present your machines as transition graphs with start and accepting states clearly labelled. You are encouraged to use the black hole convention and to omit any unreachable states. Your regular expressions should use only the basic patterns `a,b,c,...`, concatenation, parentheses, and the constructors `(*|)` .

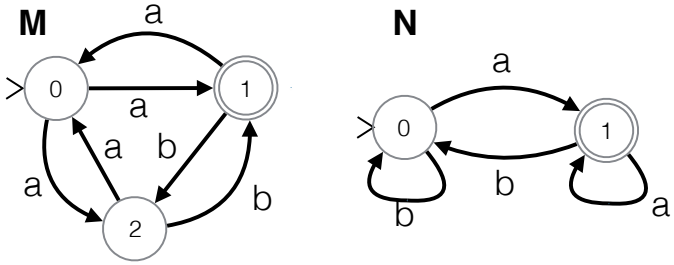
   (a) For each of the following regex, with the alphabet, {`a`, `b`}, give a simple FSM that recognises the language of strings matching this expression. You may simplify the expression given if you identify a simpler, equivalent regex. Number each state, `0, 1, 2, 3,...` If your FSM is not deterministic, construct an equivalent DFA using the subset construction. Label the states of the DFA to indicate how they have been constructed.

      i. `a(ab)*b` [*3 marks*]

      ii. `ab*|ba*` [*3 marks*]

      iii. `a(a|b*)*a*`

   (b) Two machines,`M, N`, are shown in the diagram.



      i. One of these machines is not deterministic. Construct a DFA equivalent to this machine. [*3 marks*]

      ii. Describe, using a regular expression ***or otherwise***, the languages recognised by the two machines. [*3 marks*]

      iii. Describe the product of the two DFA, by giving its states, start state, accepting state(s), and state transition table. Draw the product machine. [*3 marks*]

      iv. Describe how the language recognised by the product machine is related to the languages recognised by M and N. Give a regex for this language. [*2 marks*]

4. For this question we consider a propositional language with 8 propositional letters A, B, C, D, E, F, G, H. $A \downarrow B$ is Quine's dagger representing `not (a || b)`.

   (a) How many satisfying valuations are there for each of the following Wffs?

   i. $A \wedge (B \downarrow C)$ [4 *marks*]

   ii. $(A \to B) \wedge (B \to A) \wedge (C \to B)$ [4 *marks*]

   iii. $(A \to B) \wedge (B \to E) \wedge (A \to D) \wedge (D \to E) \wedge (E \to C)$ [4 *marks*]

   iv. $(A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (D \to G)$
   $\wedge (A \to F) \wedge (F \to G) \wedge (G \to H) \wedge (H \to E) \wedge (B \to H)$ [4 *marks*]

   v. $(A \to B) \wedge (\neg B \vee C) \wedge (A \vee D)$ [4 *marks*]

5. In this question we use the Haskell representation used in class, where a `Form` is represented as the conjunction of a list of clauses, `And cs`, and each clause, `c` is represented as the disjunction of a list of literals, `Or xs`. Literals are negative or positive atoms (`N 'a' | P 'a'`). Atoms are characters. A valuation, which may be partial, is represented as a list of literals. You *may* use this representation in your answers.

   Consider the following Clausal Form:

```
f = And[ Or[N 'a', N 'c', N 'd'], Or[P 'a', N 'c', P 'b'],
         Or[P 'a', P 'c', P 'd'], Or[P 'a', N 'b', P 'd'],
         Or[P 'a', P 'c', N 'd'], Or[P 'a', P 'b', N 'd'],
         Or[ P 'a', N 'c', N 'd']
```

   We give each clause in `f` a label (i)–(vii), for ease of reference in your answers to Questions 5d and 5e.

   (i) `Or[N 'a', N 'c', N 'd']`
   (ii) `Or[P 'a', N 'c', P 'b']`
   (iii) `Or[P 'a', P 'c', P 'd']`
   (iv) `Or[P 'a', N 'b', P 'd']`
   (v) `Or[P 'a', P 'c', N 'd']`
   (vi) `Or[P 'a', P 'b', N 'd']`
   (vii) `Or[ P 'a', N 'c', N 'd']`

   The DPLL algorithm tests a partial valuation, V, represented as a *consistent* list of literals, against a clausal form, φ.

   (a) When is a valuation V **inconsistent**? [*3 marks*]

   (b) When is a clause `(Or xs)` **not satisfied** (`not(vs |= Or xs)`) by a valuation `vs`? [*3 marks*]

   (c) When is a clause **refuted** (`vs |=/ Or xs`) by a valuation? [*3 marks*]

   (d) For each clause, (i)–(vii), in the Form `f` given above, say whether it is satisfied or refuted (or neither) by the following partial valuation: `[N 'a', P 'b', N 'd']` ? [*5 marks*]

   (e) Use the example Form, `f`, given above to explain how unit propagation is used in DPLL. Start with the partial valuation `[N 'a']` that makes `a` false. [*6 marks*]