THE AUSTRALIAN NATIONAL UNIVERSITY

Second Semester 2012

COMP2600 (Formal Methods for Software Engineering)

Writing Period: 3 hours duration

Study Period: 15 minutes duration

Permitted Materials: One A4 page with hand-written notes on both sides

Answer ALL questions
Total marks: 100
Note internal choice in Question 9

WITH SOME SAMPLE SOLUTIONS

The questions are followed by labelled blank spaces into which your answers are to be written.

Additional answer panels are provided (at the end of the paper) should you wish to use more space for an answer than is provided in the associated labelled panels. If you use an additional panel, be sure to indicate clearly the question and part to which it is linked.

The following spaces are for use by the examiners.

Q1 (NatDed)	Q2 (StrInd)	Q3 (Hoare)	Q4 (WP)	Q5 (FSA)	
Q6 (CFL)	Q7 (TM)	Q8 (Z)	Q9 (1 of 2)		Total

QUESTION 1 [12 marks] (Natural Deduction)

The following questions ask for proofs using natural deduction. Present your proofs in the Fitch style as used in lectures. You may use the introduction and elimination rules given in Appendix 1, as well as the following derived rule:

$$\text{(contradiction)}\ \frac{p \wedge \neg p}{q}$$

Number each line and include justifications for each step in your proofs.

(a) Give a natural deduction proof of $\dfrac{(p \wedge q) \Rightarrow r}{\neg(p \wedge (q \wedge \neg r))}$

QUESTION 1(a) **[4 marks]**

$$\begin{array}{r|l|ll}
1 & \underline{(p \wedge q) \Rightarrow r} & & \\
2 & & \underline{p \wedge (q \wedge \neg r)} & \\
3 & & p & \wedge\text{-E, } 2 \\
4 & & q \wedge \neg r & \wedge\text{-E, } 4 \\
5 & & q & \wedge\text{-E, } 4 \\
6 & & p \wedge q & \wedge\text{-I, } 3, 5 \\
7 & & r & \rightarrow\text{-E, } 1, 6 \\
8 & & \neg r & \wedge\text{-E, } 4 \\
9 & & r \wedge \neg r & \wedge\text{-I, } 7, 8 \\
10 & \neg(p \wedge (q \wedge \neg r)) & & \neg\text{I, } 2\text{–}9
\end{array}$$

(b) Give a natural deduction proof of $\dfrac{c \Rightarrow (a \vee b)}{(c \wedge \neg a) \Rightarrow b}$

QUESTION 1(b) **[4 marks]**

1	$\mid\ \underline{c \Rightarrow (a \vee b)}$	
2	$\mid\ \mid\ \underline{c \wedge \neg a}$	
3	$\mid\ \mid\ c$	$\wedge$-E, 2
4	$\mid\ \mid\ a \vee b$	$\rightarrow$-E, 1, 3
5	$\mid\ \mid\ \mid\ \underline{a}$	
6	$\mid\ \mid\ \mid\ \neg a$	$\wedge$-E, 2
7	$\mid\ \mid\ \mid\ a \wedge \neg a$	$\wedge$-I, 5, 6
8	$\mid\ \mid\ \mid\ b$	contradiction, 7
9	$\mid\ \mid\ \mid\ \underline{b}$	
10	$\mid\ \mid\ \mid\ b$	R, 9
11	$\mid\ \mid\ b$	$\vee$-E, 4, 5–8, 9–10
12	$\mid\ (c \wedge \neg a) \Rightarrow b$	$\rightarrow$-I, 2–11

(c) Give a natural deduction proof of $\dfrac{P \wedge \exists x.Q(x)}{\exists x.(P \wedge Q(x))}$ where x does not appear free in P.

Take care with parentheses and variable names.

QUESTION 1(c) **[4 marks]**

1		$P \wedge \exists x.Q(x)$	
2		P	$\wedge$-E, 1
3		$\exists x.Q(x)$	$\wedge$-E, 1
4	a	$Q(a)$	
5		$P \wedge Q(a)$	$\wedge$-I, 2, 4
6		$\exists x.\ (P \wedge Q(x))$	$\exists$-I, 5
7		$\exists x.\ (P \wedge Q(x))$	$\exists$-E, 3, 4–6

QUESTION 2 [12 marks] (Structural Induction)

In all proofs indicate the justification (eg, the line of a definition used) for each step.

(a) Given the following definitions of `(++)` and `count`:

```
[]     ++ ys  = ys                 -- (A1)
(x:xs) ++ ys  = x : (xs ++ ys)    -- (A2)

count []      = 0                  -- (C1)
count (x:xs)  = 1 + count xs       -- (C2)
```

Prove the following

```
count(xs ++ ys) = (count xs) + (count ys)
```

(i) State and prove the base case goal

QUESTION 2(a)(i) **[1 mark]**

We use induction on `xs` (not `ys`): the clue here is that the definition of `(++)` is recursive on its first argument `xs`.

Base case: `xs = []`
Show that `count([] ++ ys) = (count []) + (count ys)`

Proof:

```
  count([] ++ ys)
     = count ys                         -- by (A1)
     = 0 + count ys                     -- by arithmetic
     = count [] + count ys              -- by (C1)
```

(ii) State the inductive hypothesis

QUESTION 2(a)(ii) **[1 mark]**

For proving the step case with `xs = a:as`,
the inductive hypothesis is:

```
  count(as ++ ys) = (count as) + (count ys)    -- (IH)
```

(iii) State and prove the step case goal:

QUESTION 2(a)(iii) **[3 marks]**

Step case: (for `xs = a:as`)
Assuming the inductive hypothesis (IH) above, show that:
`count(a:as ++ ys) = (count a:as) + (count ys)`

Proof:

```
count(a:as ++ ys)
  = count(a:(as ++ ys))               -- by (A2)
  = 1 + count(as ++ ys)               -- by (C2)
  = 1 + count as + count ys           -- by (IH)
  = (count a:as) + (count ys)         -- by (C2)
```

(b) Binary trees can be represented in Haskell with the following algebraic data type:

```
data Tree a = Nul | Node a (Tree a) (Tree a)
```

We can flatten the tree to a list using a pre-order or in-order traversal, given by functions `pre` and `in` below:

```
pre  Nul            = []                             -- (PR1)
pre (Node a t1 t2)  = (a : (pre t1)) ++ (pre t2)     -- (PR2)

in  Nul             = []                             -- (IN1)
in (Node a t1 t2)   = (in t1) ++ (a : (in t2))       -- (IN2)
```

Prove the following property:

```
count(pre t) = count(in t)
```

You may use the result you proved in Part (a) of this question:

```
count(xs ++ ys) = (count xs) + (count ys)          -- (LEMMA1)
```

(i) State and prove the base case goal.

QUESTION 2(b)(i) **[1 mark]**

Base case: `t = Nul`
Show that `count(pre Nul) = count(in Nul)`

Proof:

```
count(pre Nul)
  = count []                  -- by (PR1)
  = count(in Nul)             -- by (IN1)
```

(ii) State the inductive hypotheses.

QUESTION 2(b)(ii) **[1 mark]**

for the step case `t = Node a t1 t2`:

```
count(pre t1) = count(in t1)                              -- (IH1)
count(pre t2) = count(in t2)                              -- (IH2)
```

(iii) State and prove the step case goal.

QUESTION 2(b)(iii) **[5 marks]**

Step case: (for `t = Node a t1 t2`)
Assuming the inductive hypotheses (IH1) and (IH2) as above, prove
`count(pre (Node a t1 t2)) = count(in (Node a t1 t2))`

Proof:

```
count(pre (Node a t1 t2))
  = count (a : (pre t1) ++ (pre t2))        -- by (PR2)
  = count (a : (pre t1)) + count(pre t2)    -- by (LEMMA1)
  = 1 + count (pre t1) + count (pre t2)     -- by (C2)
  = 1 + count (in t1)  + count (pre t2)     -- by (IH1)
  = 1 + count (in t1)  + count (in t2)      -- by (IH2)
  = count (in t1)  + 1 + count (in t2)      -- by arithmetic
  = count (in t1)  + count (a : (in t2))    -- by (C2)
  = count ((in t1) ++ a : (in t2))          -- by (LEMMA1)
  = count (in (Node a t1 t2))               -- by (IN2)
```

QUESTION 3 [12 marks] (Hoare Logic)

(a) The following algebraic ***equivalence*** will be useful in both this section, and the next section on Weakest Precondition Calculus. You may use it at any time, regardless of whether you succesfully answer this question, so long as you explictly refer to it as 'Lemma'.

Prove, using standard algebraic manipulations, that

$$t + x = \left(\frac{x+8-4}{4}\right)^2 \quad \Leftrightarrow \quad t = \left(\frac{x-4}{4}\right)^2$$

QUESTION 3(a) **[2 marks]**

$$\begin{array}{rrcl}
 & t + x & = & \left(\frac{x+8-4}{4}\right)^2 \\
 & & = & \left(\frac{x+4}{4}\right)^2 \\
 & & = & \frac{x^2+8x+16}{16} \\
\Leftrightarrow & t + \frac{16x}{16} & = & \frac{x^2+8x+16}{16} \\
\Leftrightarrow & t & = & \frac{x^2+8x+16}{16} - \frac{16x}{16} \\
 & & = & \frac{x^2-8x+16}{16} \\
 & & = & \left(\frac{x-4}{4}\right)^2
\end{array}$$

Consider the following code fragment:

```
t := 1;                                          ⎫
x := 0;                                          ⎪
while (x ≠ 8*n) do                ⎫              ⎬ OddSquare
    t := t + x;        ⎫ Body     ⎬ Loop         ⎪
    x := x + 8         ⎭          ⎭              ⎭
```

We wish to use the rules of ***Hoare Logic*** (Appendix 3) to show that the value of t, if and when this code terminates, will be the n'th ***odd square***. That is, we will try to prove

$\{True\}\ OddSquare\ \{Post\}$.

where

$Post \equiv (\ t = (2n-1)^2\ )$

You may assume that all variables are typed integer. In the questions below we will refer to the code fragment as *OddSquare*, to the loop code as *Loop*, and to the body of the loop as *Body*. ***Make sure that every step of your proof is numbered, and is justified by citing the rule, and any previous proof steps, that you are using.***

(b) We will need an invariant for *Loop*. We suggest

$$Inv \quad \equiv \quad (\, t = \left(\frac{x-4}{4}\right)^2 \,).$$

Prove that

$$\{Inv\} \; Body \; \{Inv\}.$$

QUESTION 1(b) **[4 marks]**

1. $\{t = \left(\frac{x+8-4}{4}\right)^2\} \; \texttt{x} := \texttt{x} + 8 \; \{Inv\}$ (Asst.)
2. $\{t + x = \left(\frac{x+8-4}{4}\right)^2\} \; \texttt{t} := \texttt{t} + \texttt{x} \; \{t = \left(\frac{x+8-4}{4}\right)^2\}$ (Asst.)
3. $\{Inv\} \; \texttt{t} := \texttt{t} + \texttt{x} \; \{t = \left(\frac{x+8-4}{4}\right)^2\}$ (Lemma, Precond. Equiv.)
4. $\{Inv\} \; Body \; \{Inv\}$ (1,3, Seq.)

(c) Using part (b), prove that

$$\{Inv\} \; Loop \; \{Post\}.$$

QUESTION 1(c) **[3 marks]**

5. $(Inv \wedge x \neq 8 * n) \Rightarrow Inv$ (Basic Logic)
6. $\{Inv \wedge x \neq 8 * n\} \; Body \; \{Inv\}$ (**(b)**,5, Precond. Strength.)
7. $\{Inv\} \; Loop \; \{Inv \wedge x = 8 * n\}$ (6, While)
8. $(Inv \wedge x = 8 * n) \Rightarrow Post$ (Basic Math.)
9. $\{Inv\} \; Loop \; \{Post\}$ (7,8, Postcond. Weak.)

(d) Using part (c), prove that

$$\{\mathit{True}\}\ \mathit{OddSquare}\ \{\mathit{Post}\}.$$

QUESTION 1(d) **[3 marks]**

10. $\{t = \left(\frac{0-4}{4}\right)^2\}\ \texttt{x := 0}\ \{\mathit{Inv}\}$ (Asst.)
11. $\{1 = \left(\frac{0-4}{4}\right)^2\}\ \texttt{t := 1}\ \{t = \left(\frac{0-4}{4}\right)^2\}$ (Asst.)
12. $\{\mathit{True}\}\ \texttt{t := 1}\ \{t = \left(\frac{0-4}{4}\right)^2\}$ (11, Precond. Equiv.)
13. $\{\mathit{True}\}\ \texttt{t := 1; x := 0}\ \{\mathit{Inv}\}$ (10,12, Seq.)
14. $\{\mathit{True}\}\ \mathit{OddSquare}\ \{\mathit{Post}\}$ (13,**(c)**, Seq.)

QUESTION 4 [12 marks] (Weakest Precondition Calculus)

As with the previous question, we will consider the code fragment *OddSquare*:

```
t := 1;                                      }
x := 0;                                      }
while (x ≠ 8*n) do                 }         } OddSquare
    t := t + x;        }           } Loop    }
    x := x + 8         } Body      }         }
```

We will use the rules of the ***Weakest Precondition Calculus*** (Appendix 4) to calculate

$$wp(OddSquare,\ t = (2n-1)^2).$$

As in the previous question we will use the abbreviations *Loop* and *Body* for the indicated parts of the code. You may continue to make use of the lemma from part (a) of the previous question by referring to it explicitly as ‘Lemma’. ***Remember to simplify your answers wherever possible, and show all your working when you do so.***

(a) We will want to calculate

$$wp(Loop,\ t = (2n-1)^2).$$

First, prove that P_0 (the predicate expressing success for this weakest precondition after zero loop iterations) is equal to

$$x = 8n \ \wedge\ t = \left(\frac{x-4}{4}\right)^2$$

QUESTION 4(a) **[2 marks]**

$$\begin{aligned}
P_0 &\equiv \neg\,(x \neq 8*n) \wedge t = (2n-1)^2 \\
&\equiv x = 8*n \wedge t = \left(2*\frac{x}{8} - 1\right)^2 \\
&\equiv x = 8n \wedge t = \left(\frac{x}{4} - 1\right)^2 \\
&\equiv x = 8n \wedge t = \left(\frac{x}{4} - \frac{4}{4}\right)^2 \\
&\equiv x = 8n \wedge t = \left(\frac{x-4}{4}\right)^2
\end{aligned}$$

(b) Calculate P_1 (the predicate expressing success after one loop iteration).

QUESTION 4(b) **[3 marks]**

$$\begin{aligned}
P_1 &\equiv x \neq 8n \wedge wp(Body, P_0) \\
&\equiv x \neq 8n \wedge wp(\mathtt{t := t + x},\ wp(\mathtt{x := x + 8}, P_0)) \\
&\equiv x \neq 8n \wedge wp\left(\mathtt{t := t + x},\ x + 8 = 8n \wedge t = \left(\frac{x+8-4}{4}\right)^2\right) \\
&\equiv x \neq 8n \wedge x + 8 = 8n \wedge t + x = \left(\frac{x+8-4}{4}\right)^2) \\
&\equiv x + 8 = 8n \wedge t = \left(\frac{x-4}{4}\right)^2
\end{aligned}$$

($x \neq 8n$ can be eliminated because we know that $x+8 = 8n$. The simplification on the right is the lemma from Question 2.)

(c) Hence suggest (without proof) an expression for P_k (the predicate expressing success after k loop iterations) that holds for all $k \geq 0$.

QUESTION 4(c) **[1 mark]**

$$P_k \equiv x + 8k = 8n \wedge t = \left(\tfrac{x-4}{4}\right)^2$$

(d) Given your answer to part (c), state

$$wp(Loop,\ t = (2n-1)^2).$$

Do not attempt any simplification at this stage.

QUESTION 4(d) **[1 mark]**

$$\begin{aligned}
&\exists k.\, k \geq 0 \wedge P_k \\
\equiv\quad &\exists k.\, k \geq 0 \wedge x + 8k = 8n \wedge t = \left(\tfrac{x-4}{4}\right)^2
\end{aligned}$$

(e) Hence find

$$wp(\mathit{OddSquare},\ t = (2n-1)^2).$$

State this result in the simplest form possible.

QUESTION 4(e) **[3 marks]**

$$\begin{aligned}
&wp(\mathit{OddSquare}, t = (2n-1)^2)\\
&\equiv wp(\texttt{t} := \texttt{1},\ wp(\texttt{x} := \texttt{0},\ wp(\mathit{Loop}, t = (2n-1)^2)\\
&\equiv wp(\texttt{t} := \texttt{1},\ \exists\, k.\, k \geq 0 \land 0 + 8k = 8n \land t = \left(\tfrac{0-4}{4}\right)^2)\\
&\equiv \exists\, k.\, k \geq 0 \land 0 + 8k = 8n \land 1 = \left(\tfrac{0-4}{4}\right)^2)\\
&\equiv \exists\, k.\, k \geq 0 \land k = n \land \mathit{True}\\
&\equiv n \geq 0
\end{aligned}$$

(f) Given any code fragment *S* and assertion *Q*, does it always hold that

$$wp(S, \neg\, Q) \ \Leftrightarrow\ \neg\, wp(S, Q) \quad ?$$

If you answered ***yes***, give a short explanation in plain English (not a formal proof). If you answered ***no***, give a counter-example (you do not need to formally prove that it is a counter-example).

QUESTION 4(f) **[2 marks]**

No.

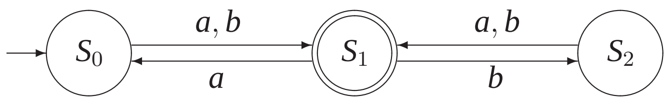
Let *S* be a program that never terminates, e.g. `while False do x:=x`. Let *Q* be *False*.

(then $wp(S, \neg\, Q) \ \equiv\ wp(S, \mathit{True}) \ \equiv\ \mathit{False}$;

but $\neg\, wp(S, Q) \ \equiv\ \neg\, wp(S, \mathit{False}) \ \equiv\ \neg\, \mathit{False} \ \equiv\ \mathit{True}$.)

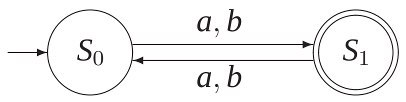
QUESTION 5 [12 marks] (Finite State Automata)

(a) Consider the deterministic finite state automaton A:



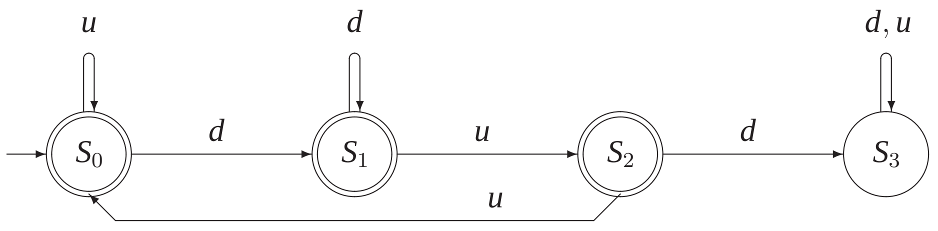
Define a new deterministic finite state automaton that accepts the same language as A but is ***minimal***.

QUESTION 5(a) **[2 marks]**



(S_0 and S_2 are minimal; S_0 is the start state so cannot be removed, so we remove S_2 and redirect the arc into S_2 to go to S_0.)

(b) Consider the deterministic finite state automaton B:



Give a fully rigorous proof that any string of the form $\alpha\, d\, u\, d\, \beta$, where α, β are arbitrary strings in $\{d,\ u\}^*$, is ***not*** accepted by B.

You may use without proof the ***lemma***

$$N^*(S_3,\ \beta) \ = \ S_3 \ \text{ for all } \ \beta \in \{d,\ u\}^*$$

as long as you are explicit about when you are using it.

QUESTION 5(b) **[4 marks]**

We will prove that $N^*(S_0, \alpha\, d\, u\, d\, \beta) = S_3$, as S_3 is the only non-final state.

By the append theorem $N^*(S_0, \alpha\, d\, u\, d\, \beta) = N^*(N^*(S_0, \alpha), d\, u\, d\, \beta)$.

We cannot know what $N^*(S_0, \alpha)$ is, so we will try every possibility:

$N^*(S_0, d\, u\, d\, \beta) = N^*(S_1, u\, d\, \beta) = N^*(S_2, d\, \beta) = N^*(S_3, \beta) = S_3$ (lemma).
$N^*(S_1, d\, u\, d\, \beta) = N^*(S_1, u\, d\, \beta) = N^*(S_2, d\, \beta) = N^*(S_3, \beta) = S_3$ (lemma).
$N^*(S_2, d\, u\, d\, \beta) = N^*(S_3, u\, d\, \beta) = S_3$ (lemma).
$N^*(S_3, d\, u\, d\, \beta) = S_3$ (lemma).

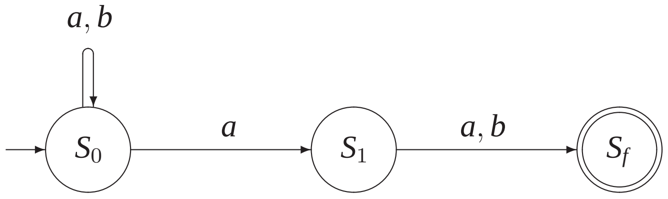
So no matter what state $N^*(S_0, \alpha)$ is, we end up in S_3 as required.

(c) Consider the right-linear grammar

$$\begin{aligned} S_0 &\to aS_0 \mid bS_0 \mid aS_1 \\ S_1 &\to a \mid b \end{aligned}$$

Using the algorithm given in lectures, draw a (non-deterministic) finite state automaton that accepts the same language.

QUESTION 5(c) **[2 marks]**



(d) Describe the language that the grammar and non-deterministic finite state automaton of question (c) accept.

Your answer may be in mathematical notation or plain English. Formal proof of your claim is not required.

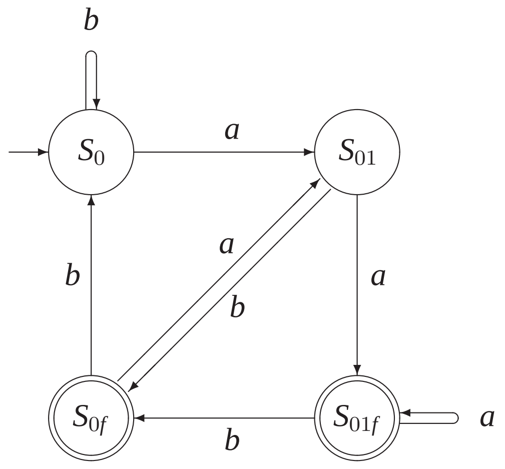
QUESTION 5(d) **[1 mark]**

Strings over $\{a, b\}$ whose last-but-one token is an a.

(e) Using the ***subset construction*** algorithm, derive a ***deterministic*** finite state automaton from the non-deterministic automaton you gave as your answer to question (c).

Be clear about how states of your deterministic automaton are derived from states of the original non-deterministic automaton.

QUESTION 5(e) **[3 marks]**



QUESTION 6 [10 marks] (Context-Free Languages)

Let L be the language

$$\{a^n\, b\, a^n \mid n \geq 0\}$$

(a) Give a clear explanation why L is ***not*** a ***regular language***.

Your explanation should be in English; a rigorous mathematical proof is not necessary.

QUESTION 6(a) **[2 marks]**

A finite state automata for L would need to keep track of how many a's it has read. But the only 'memory' a FSA has are its finitely many states, so it cannot possibly keep count of arbitrarily long strings of a's.

(b) Give a ***context-free grammar*** that generates L.

Use S as the start symbol of your grammar.

QUESTION 6(b) **[2 marks]**

$$S \longrightarrow a\,S\,a \quad | \quad b$$

(c) Using the algorithm given in lectures, convert the context-free grammar you defined in (b) into a description of a ***push-down automaton***.

Use q_0 as your start state, Z as the initial stack symbol, and indicate your final states by underlining them. The first transition is given for you.

QUESTION 6(c) **[3 marks]**

$$
\begin{aligned}
\delta(q_0,\, \epsilon,\, Z) &= q_1 \,/\, S\,Z\\
\delta(q_1,\, \epsilon,\, S) &= q_1 \,/\, a\,S\,a\\
\delta(q_1,\, \epsilon,\, S) &= q_1 \,/\, b\\
\delta(q_1,\, a,\, a) &= q_1 \,/\, \epsilon\\
\delta(q_1,\, b,\, b) &= q_1 \,/\, \epsilon\\
\delta(q_1,\, \epsilon,\, Z) &= \underline{q_2} \,/\, \epsilon
\end{aligned}
$$

(d) The algorithm used for question (c) produces a ***non-deterministic*** push-down automaton. Design a ***deterministic*** push-down automaton that recognises L.

Use q_0 as your start state, Z as the initial stack symbol, and underline any final states.

(Note: this machine needs to be designed from scratch, so you may be able to answer this question even if you did not answer questions (b) and (c).)

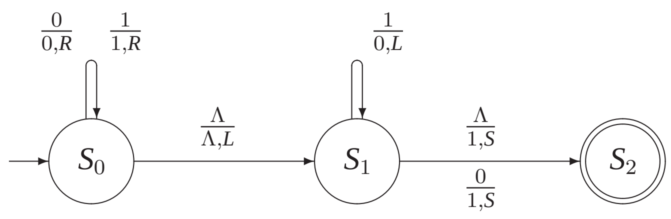
QUESTION 6(d) **[3 marks]**

$$
\begin{aligned}
\delta(q_0,\, a,\, Z) &= q_0 \,/\, a\,Z\\
\delta(q_0,\, a,\, a) &= q_0 \,/\, a\,a\\
\delta(q_0,\, b,\, Z) &= \underline{q_2} \,/\, \epsilon \qquad \text{(to accept immediately if string is } b\text{)}\\
\delta(q_0,\, b,\, a) &= q_1 \,/\, a\\
\delta(q_1,\, a,\, a) &= q_1 \,/\, \epsilon\\
\delta(q_1,\, \epsilon,\, Z) &= \underline{q_2} \,/\, \epsilon
\end{aligned}
$$

QUESTION 7 [6 marks] (Turing Machines)

The Turing machine *M* below, previously given in lectures,

- accepts any binary number (non-empty string over $\{0, 1\}$) and increments it by one;
- accepts the empty string ϵ and outputs 1;
- rejects any other string:



(a) Give a full trace that shows the action of the machine *M* on the input 101.

QUESTION 7(a) **[1 mark]**

$$\begin{aligned}
[S_0]101 &\Rightarrow 1[S_0]01 \\
&\Rightarrow 10[S_0]1 \\
&\Rightarrow 101[S_0]\Lambda \\
&\Rightarrow 10[S_1]1 \\
&\Rightarrow 1[S_1]00 \\
&\Rightarrow 1[S_2]10
\end{aligned}$$

(b) Design a Turing machine that takes as input strings of the form

$$\# \underbrace{1\,1\,\ldots\,1}_{n}$$

for $n > 0$, and returns as output

$$k\,\#$$

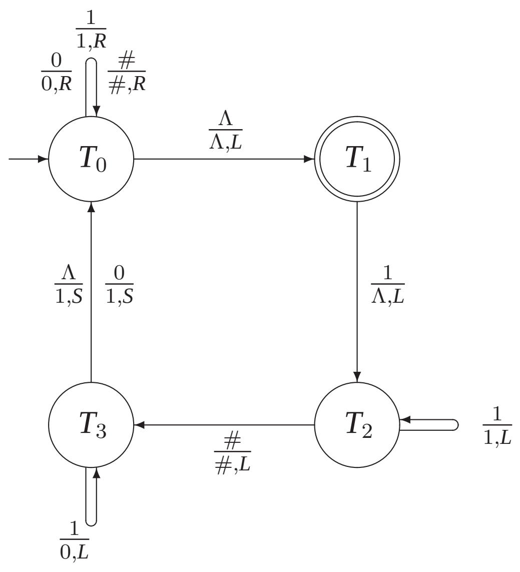
where *k* is the ***binary*** representation of *n*.

In other words, this machine should convert ***unary numbers into binary numbers***. The $\#$ symbol is simply there to help your machine separate your input and output as you go.

For this question you do not need to worry about the case that the input is not in the expected form, e.g. if it is empty, or contains 0's; you will be marked correct if your machine works correctly on the expected input.

(Hint: you may like to re-use parts of the Turing machine M.)

QUESTION 7(b) **[5 marks]**



- State T_0 moves the head to the far right of the tape; corresponds to state S_2 of the machine M.
- State T_1 deletes the furthest right 1 if it can. If it cannot (i.e. if there are no 1's and it reads a $\#$) it halts accepting.
- State T_2 moves the head left over the $\#$ to be at the far right of the binary number; corresponds to state S_0 of M.
- State T_3 increments the binary number; corresponds to state S_1 of M.

QUESTION 8 [16 marks] (Specification in Z)

The system, *CarInsurance*, that is partially modelled in this question is one where a company insures cars, writing insurance policies with individual owners. Each policy is for one car, of some model (which includes make and year), that is owned by a particular person of known age. The annual premium is set on the basis of the car's model, the owner's age and sex, together with the number of years insured.

```
[Car]
[Model]
[Person]
```

$Age == \{15\,..\,90\}$

$Sex ::= male \mid female$

$| \; sexOf : Person \to Sex$

Policy

$car : Model$
$owner : Person$
$age : Age$
$yearsWithoutClaim : \mathbb{N}$

$| \; noClaimBonus : \mathbb{N} \to \mathbb{R}$
$| \; modelOfCar : Car \to Model$

CarInsurance

$policies : \mathbb{P}\,Policy$
$basicPremium : Model \nrightarrow \mathbb{R}$
$riskFactor : (Model \times Age \times Sex) \nrightarrow \mathbb{R}$

$\forall m : Model \bullet basicPremium(m) > 0$
$\forall m : Model,\ a : Age,\ s : Sex \bullet riskFactor(m, a, s) \geq 1.0$
$\forall p_1, p_2 : Policy \bullet ((p_1.owner = p_2.owner) \land (p_1.age > p_2.age)) \Rightarrow (p_1.age - p_2.age) \leq 1$

(a) In the above prelude to the specification of operations and queries, which identifiers are

(i) the *given types*,
(ii) the *introduced types?*

QUESTION 8(a) **[1 mark]**

1. The given types are *Car*, *Model* and *Person*.
2. The introduced types are *Age*, *Sex* and *Policy*.

(b) In the above prelude to the specification of operations and queries, which identifiers are

(i) the *introduced constants* (both defined and axiomatically introduced),
(ii) the *global variables* for the system?

QUESTION 8(b) **[1 mark]**

1. The introduced constants are *sexOf*, *noClaimBonus*, *modelOfCar*, *male* and *female*.
2. The global variables are *policies*, *basicPremium* and *riskFactor*.

(c) Why are the global variables *basicPremium* and *riskFactor* declared as *partial* functions?

QUESTION 8(c) **[1 mark]**

It is unlikely that the insurance company will want to determine the basic premiums and risk factors for *all* models of cars.

(d) Describe in simple terms what the third system invariant says.

QUESTION 8(d) **[2 marks]**

The age of a client should not differ in two of his/her policies by more than a year.

(e) There is a system invariant missing. It is that no car should appear in two policies. Suggest, using Z notation, a suitable extra invariant for the state schema.

QUESTION 8(e) **[1 mark]**

$$\forall p_1, p_2 : Policy \bullet \{p_1, p_2\} \subset policies \Rightarrow (p_1.car = p_2.car) \Rightarrow (p_1 = p_2)$$

(f) Write the predicate part of the *Initial* schema for this system.

QUESTION 8(f) **[1 mark]**

$policies := \varnothing$
$basicPremium := \varnothing$
$riskFactor := \varnothing$

(g) Write a schema for the enquiry, *InvoiceAmount*, which yields in variable *premium*! the amount the client will be charged for the coming year on policy $p?$.
The premium to be computed is the product of the basic premium, the risk factor and the no-claim-bonus (which depends on the number of years with no claims).

QUESTION 8(g) **[1 mark]**

InvoiceAmount

$\Xi CarInsurance$
$p? : Policy$
$premium! : \mathbb{R}$

$$premium' = basicPremium(modelOfCar(p?.car)) * riskFactor(p?.car,\ p?.age,\ sexOf(p?.owner)) * noClaimBonus(p?.yearsWithoutClaim)$$

(h) The following schema specifies the operation of an existing client renewing his/her insurance policy for a particular car. It has a problem.

$$
\begin{array}{|l}
\textit{RenewPolicy}_o \\
\hline
\Delta \textit{CarInsurance} \\
p? : \textit{Policy} \\
\textit{newPolicy} : \textit{Policy} \\
\hline
p? \in \textit{policies} \\
\textit{newPolicy}.\textit{car} = p?.\textit{car} \\
\textit{newPolicy}.\textit{owner} = p?.\textit{owner} \\
\textit{newPolicy}.\textit{age} = p?.\textit{age} + 1 \\
\textit{newPolicy}.\textit{yearsWithoutClaim} = p?.\textit{yearsWithoutClaim} + 1 \\
\textit{policies}' = \textit{policies} \setminus \{p?\} \cup \{\textit{newPolicy}\} \\
\textit{riskFactor}' = \textit{riskFactor} \\
\hline
\end{array}
$$

(i) What is the problem in this schema? How do you fix it?

QUESTION 8(i) **[1 mark]**

There is no postcondition for the final value of the state variable *basicPremium*. The following should be added:

$\textit{basicPremium}' = \textit{basicPremium}$

(j) Write an operation schema, *RaisePremium*, that increases the basic premium of all models by 1 percent.

QUESTION 8(j) **[2 marks]**

$$
\begin{array}{|l}
\textit{RaisePremium} \\
\hline
\Delta \textit{CarInsurance} \\
\hline
\textit{policies}' = \textit{policies} \\
\forall\, m : \textit{Model} \bullet\ m \in \mathrm{dom}(\textit{basicPremium}) \ \Rightarrow \\
\qquad\qquad \textit{basicPremium}'(m) = \textit{basicPremium}(m) \ * \ 1.01 \\
\textit{riskFactor}' \ = \ \textit{riskFactor} \\
\hline
\end{array}
$$

(k) Write a schema, *NumberOfOwners*, specifying an enquiry that gives the number of car owners with policies currently held; the output variable should be *ownerCount*!.

QUESTION 8(k) **[2 marks]**

NumberOfOwners

$$
\begin{array}{l}
\Xi CarInsurance \\
ownerCount! : \mathbb{N} \\
owners : \mathbb{P}\, Person \\
\hline
\forall x : Person \bullet x \in owners \Leftrightarrow \\
\qquad \exists p : Policy \bullet p \in policies \wedge x = p.owner \\
ownerCount! = \# owners
\end{array}
$$

(l) Write a schema $NewPolicy_o$ that specifies the operation of creating a new car insurance policy for a client.
Hint: Adapt the schema $RenewPolicy_o$. You will need to define some different variables and write some different postconditions.

QUESTION 8(l) **[3 marks]**

$NewPolicy_o$

$$
\begin{array}{l}
\Delta CarInsurance \\
car? : Car \\
age? : Age \\
owner? : Person \\
newPolicy : Policy \\
\hline
\forall p : Policy \bullet (p \in policies) \Rightarrow car? \neq p.car \\
newPolicy.car = car? \\
newPolicy.owner = owner? \\
newPolicy.age = age? \\
newPolicy.yearsWithoutClaim = 0 \\
policies' = policies \cup \{newPolicy\} \\
basicPremium' = basicPremium \\
riskFactor' = riskFactor
\end{array}
$$

QUESTION 9 [8 marks] (Choice of *one* of two topics)

(a) Lambda Calculus and Prolog

(b) Computability

Attempt just ONE of the two parts (a) or (b) of this question (noting that some parts have several subparts, totalling 8 marks). (If you start to write answers to more than one part, indicate CLEARLY which one is to be marked).

(a) Lambda Calculus and Prolog

(i) Use α-conversion to change the following to an equivalent expression in which distinct bound variables are represented by different letters:

$$(\lambda a.\ \lambda b.\ (\lambda a.\ (\lambda a.\ b)\ (\lambda b.\ (\lambda a.\ a)\ b))\ a)$$

QUESTION 9(a)(i) **[1 mark]**

$$(\lambda a.\ \lambda b.\ (\lambda a.\ \underline{(\lambda a.\ b)}\ (\lambda b.\ \underline{(\lambda a.\ a)}\ b))\ a)$$

The easy bits underlined!

$$=\ \ (\lambda a.\ \lambda b.\ (\lambda f.\ (\lambda c.\ b)\ (\lambda e.\ (\lambda d.\ d)\ e))\ a)$$

(ii) Reduce the following expression to normal form, showing each step in the reduction:

$$(\lambda m.\ \lambda n.\ \ n\ \ m)\ (\lambda s.\ \lambda z.\ \ s\ (s\ \ z))\ (\lambda s.\ \lambda z.\ \ s\ \ z)$$

QUESTION 9(a)(ii) **[2 marks]**

$$\begin{aligned}
&(\lambda m.\ \lambda n.\ n\ m)(\lambda s.\ \lambda z.\ s\ (s\ z))(\lambda s.\ \lambda z.\ s\ z)\\
&= (\lambda n.\ n\ (\lambda s.\ \lambda z.\ s\ (s\ z)))(\lambda s.\ \lambda z.\ s\ z)\\
&= (\lambda s.\ \lambda z.\ s\ z)(\lambda s.\ \lambda z.\ s\ (s\ z))\\
&= (\lambda z.\ (\lambda s.\ \lambda z.\ s\ (s\ z))z)\\
&= (\lambda s.\ (\lambda z.\ s\ (s\ z)))
\end{aligned}$$

PROLOG Questions

The following items constitute the fact database in a Prolog program:

```
uses(sue, java)
uses(tom, java)
uses(ann, perl)
uses(tom, perl)
```

(iii) Give a goal appropriate to finding programmers using both Java and Perl.

QUESTION 9(a)(iii) **[1 mark]**

uses(X, java), uses(X, perl)

(iv) Describe what happens during execution of this goal; your description should have enough detail to include an account of the backtracking involved, and to give the result(s) of the goal.

QUESTION 9(a)(iv) **[2 marks]**

- uses(sue, java) matches uses(X, java) leading to subgoal uses(sue, perl) which fails.
- uses(tom, java) then matches uses(X, java) leading to subgoal uses(tom, perl) which succeeds. So first answer is X=tom
- Nothing else matches uses(X, java) so the main goal finally fails. Thus no more answers.

(v) The `usesall` relation between a programmer and a list of languages succeeds if the programmer uses all the languages in the list. For example:

```
usesall(tom, [java, perl])
```

is satisfied given the above database.

Give rules (one of which will be recursive) to implement the `usesall` relation.

(Recall that `[X|Xs]` is the Prolog equivalent of the Haskell notation `(x:xs)`. Thus `[X|Xs]` matches `[1,2,3,4]` by setting `X = 1` and `Xs = [2,3,4]`.)

QUESTION 9(a)(v) **[2 marks]**

usesall(X,[ ]).
usesall(X, [Y | Ys]) :- uses(X,Y), usesall(X, Ys).

(b) Computability

(i) In this course you have learned that certain types of abstract machine correspond exactly to certain types of language. For example, ***finite state automata*** correspond to ***regular languages***.

What type of abstract machine corresponds exactly to the ***recursive languages***?

QUESTION 9(b)(i) **[1 mark]**

Total Turing Machines (i.e. TMs that **always halt**).

(ii) Let U be the ***universal Turing machine*** that takes as input pairs (M, w), where M is an (encoded) Turing machine and w is any string, and then acts as M would on w.

Let $L(U)$ be the language of U, i.e. the pairs (M, w) for which M accepts w.

Prove that $L(U)$ is ***not*** a recursive language.

QUESTION 9(b)(ii) **[7 marks]**

Suppose for contradiction that there exists a total Turing Machine N that accepts $L(U)$.

Let P be a Turing Machine that takes as input encoded Turing Machines M and

(a) accepts if N halts rejecting (M, M);

(b) halts rejecting if N accepts (M, M).

Does P accept P?

Yes – then N rejects (P, P), so P rejects P – **contradiction**;

No – then N accepts (P, P), so P accepts P – **contradiction**.

Either way we have a contradiction, so P cannot exist.

But P was derived in a straightfoward manner from N, so N cannot exist.

Additional answers. Clearly indicate the corresponding question and part.

Additional answers. Clearly indicate the corresponding question and part.

Additional answers. Clearly indicate the corresponding question and part.

Additional answers. Clearly indicate the corresponding question and part.

Appendix 1 — Natural Deduction Rules

Propositional Calculus

$$(\wedge I) \quad \frac{p \qquad q}{p \wedge q} \qquad\qquad (\wedge E) \quad \frac{p \wedge q}{p} \qquad \frac{p \wedge q}{q}$$

$$(\vee I) \quad \frac{p}{p \vee q} \qquad \frac{p}{q \vee p} \qquad\qquad (\vee E) \quad \frac{p \vee q \qquad \begin{array}{c}[p]\\ \vdots \\ r\end{array} \qquad \begin{array}{c}[q]\\ \vdots \\ r\end{array}}{r}$$

$$(\Rightarrow I) \quad \frac{\begin{array}{c}[p]\\ \vdots \\ q\end{array}}{p \Rightarrow q} \qquad\qquad (\Rightarrow E) \quad \frac{p \qquad p \Rightarrow q}{q}$$

$$(\neg I) \quad \frac{\begin{array}{c}[p]\\ \vdots \\ q \wedge \neg q\end{array}}{\neg p} \qquad\qquad (\neg E) \quad \frac{\begin{array}{c}[\neg p]\\ \vdots \\ q \wedge \neg q\end{array}}{p}$$

Predicate Calculus

$$(\forall I) \quad \frac{P(a) \qquad (a \text{ arbitrary})}{\forall x.\, P(x)} \qquad\qquad (\forall E) \quad \frac{\forall x.\, P(x)}{P(a)}$$

$$(\exists I) \quad \frac{P(a)}{\exists x.\, P(x)} \qquad\qquad (\exists E) \quad \frac{\exists x.\, P(x) \qquad \begin{array}{c}[P(a)]\\ \vdots \\ q\end{array} \qquad (a \text{ arbitrary})}{q \quad (a \text{ is not free in } q)}$$

Appendix 2 — Truth Table Values

p	q	$p \vee q$	$p \wedge q$	$p \Rightarrow q$	$\neg p$	$p \Leftrightarrow q$
T	T	T	T	T	F	T
T	F	T	F	F	F	F
F	T	T	F	T	T	F
F	F	F	F	T	T	T

Appendix 3 — Hoare Logic Rules

- Precondition Strengthening:

$$\frac{P_s \;\Rightarrow\; P_w \qquad \{P_w\}\; S\; \{Q\}}{\{P_s\}\; S\; \{Q\}}$$

- Postcondition Weakening:

$$\frac{\{P\}\; S\; \{Q_s\} \qquad Q_s \;\Rightarrow\; Q_w}{\{P\}\; S\; \{Q_w\}}$$

- Assignment:

$$\{Q(e)\}\; \texttt{x} := \texttt{e}\; \{Q(x)\}$$

- Sequence:

$$\frac{\{P\}\; S_1\; \{Q\} \qquad \{Q\}\; S_2\; \{R\}}{\{P\}\; S_1;\; S_2\; \{R\}}$$

- Conditional:

$$\frac{\{P \wedge b\}\; S_1\; \{Q\} \qquad \{P \wedge \neg\, b\}\; S_2\; \{Q\}}{\{P\}\; \textbf{if}\; b\; \textbf{then}\; S_1\; \textbf{else}\; S_2\; \{Q\}}$$

- While Loop:

$$\frac{\{P \wedge b\}\; S\; \{P\}}{\{P\}\; \textbf{while}\; b\; \textbf{do}\; S\; \{P \wedge \neg\, b\}}$$

Appendix 4 — Weakest Precondition Rules

$$\begin{aligned}
wp(\texttt{x := e},\, Q(x)) &\equiv Q(e) \\
wp(S_1;\; S_2,\, Q) &\equiv wp(S_1, wp(S_2,\, Q)) \\
wp(\texttt{if}\;\, b\;\, \texttt{then}\;\, S_1\;\, \texttt{else}\;\, S_2,\, Q) &\equiv (b \Rightarrow wp(S_1,\, Q)) \wedge (\neg\, b \Rightarrow wp(S_2,\, Q)) \\
&\equiv (b \wedge wp(S_1,\, Q)) \vee (\neg\, b \wedge wp(S_2,\, Q))
\end{aligned}$$

P_k is the weakest predicate that must be true before while b do S executes, in order for the loop to terminate after exactly k iterations in a state that satisfies Q.

$$\begin{aligned}
P_0 &\equiv \neg\, b \wedge Q \\
P_{k+1} &\equiv b \wedge wp(S, P_k) \\
wp(\texttt{while}\;\, b\;\, \texttt{do}\;\, S,\, Q) &\equiv \exists\, k.\, (k \geq 0 \wedge P_k)
\end{aligned}$$

Appendix 5 — Short Glossary of Mathematical Symbols in Z

Logic

$\wedge$	conjunction	$\forall$	for all	$\Rightarrow$	implies
$\vee$	disjunction	$\exists$	there exists	$\Leftrightarrow$	if and only if
$\neg$	negation	$\mathbb{B}$	type boolean		

Sets

$\varnothing$	empty set	$\subseteq$	subset	$\times$	cartesian product
$\{\,\}$	empty set	$\supseteq$	superset	$\mathbb{P}$	power set
$\in$	in set	$\cup$	set union	$\#$	set size
$\notin$	not in set	$\cap$	set intersection	$\ldots$	up to (as in $\{1 \ldots 7\}$)
min	smallest in set	*max*	greatest in set	$\mathbb{N}$	natural numbers

Relations and Functions

$\leftrightarrow$	relation	dom	domain	$\lhd$	domain restriction
$\rightarrow$	total function	ran	range	$\rhd$	range restriction
$\nrightarrow$	partial function	R^{-1}	inverse of R		
$\mapsto$	maplet	$R(\!\vert\, S \,\vert\!)$	image of set S under R		

Schemas

Δ	indicates operation	Ξ	indicates enquiry	$\widehat{=}$	schema definition