Module Title: Informatics 1A
Exam Diet (Dec/April/Aug): Dec 2006
Brief notes on answers:

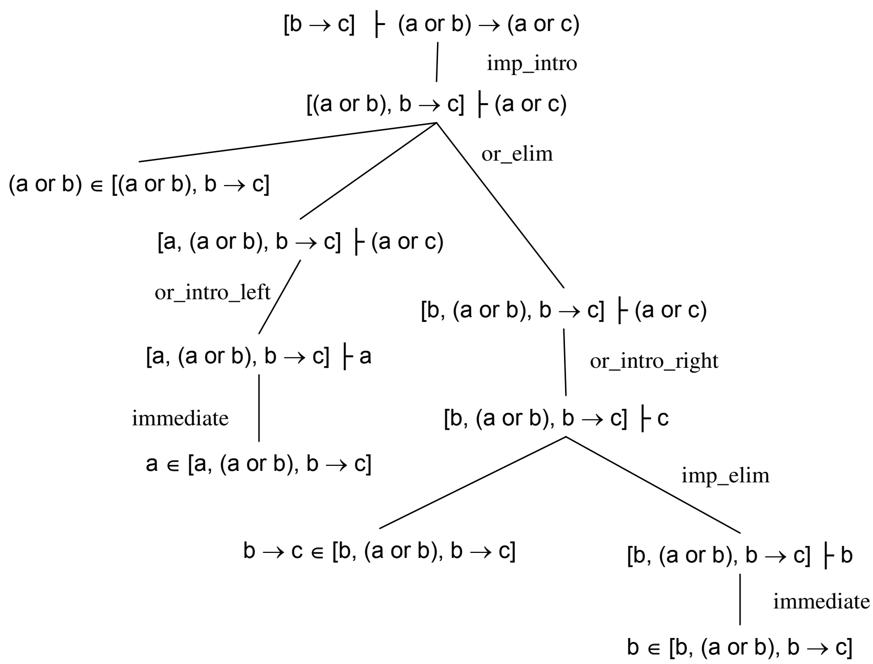
1. (a)
```
f :: [Int] -> Int
f xs  =  sum [ x^3 | x <- xs, x > 0 ]
```
   (b)
```
f' :: [Int] -> Int
f' []  =  0
f' (x:xs) | x > 0      =  x^3 + f' xs
          | otherwise  =  f' xs
```
   (c)
```
f'' :: [Int] -> Int
f'' =  foldr (+) 0 . map (^3) . filter (>0)

test1 = t f && t f' && t f''
  where
  t f =
    f [-1,1,3,-5,2] == 36
```

2. (a)
```
match :: Char -> Char -> Bool
match '_' y | isAlpha y              =  True
match x y | isAlpha x && isAlpha y  =  x == y
```
   (b)
```
matches :: String -> String -> Bool
matches xs ys  =  and [ match x y | (x,y) <- zip xs ys ] &&
                                            length xs == length ys
```
   (c)
```
matches' :: String -> String -> Bool
matches' [] []          =  True
matches' (x:xs) (y:ys)  =  match x y && matches' xs ys
matches' _ _            =  False

test2 = t matches && t matches'
  where
  t m =
    m "_e__o" "hello" &&
    m "_e__o" "pesto" &&
    not (m "_e__o" "hallo") &&
    not (m "_e__o" "helloa") &&
```

3. (a)
```
insert x n ys  =  take n ys ++ [x] ++ drop n ys
```
   (b)
```
insert' x 0 ys          =  x:ys
insert' x (n+1) []      =  [x]
insert' x (n+1) (y:ys)  =  y : insert' x n ys

test3 = t insert && t insert'
  where
  t i  =
    insert 'x' 0 "abcd" == "xabcd" &&
    insert 'x' 2 "abcd" == "abxcd" &&
```

```
insert 'x' 4 "abcd" == "abcdx"
```

4. (a) An appropriate proof is:



Students may use any precise way of describing the proof tree, as long as the steps in the proof are easily seen. This is not direct bookwork.

(b) We are missing the counterpart, for conjunction, of the *or_elim* proof rule. In other words, we need a rule that can "unpack" conjunctive expressions in our set of axioms. This is not direct bookwork.

(c) A sound proof system permits no invalid proofs. A complete proof system permits all valid proofs. This is bookwork.

5. (a) Representing each circuit as a propositional expression gives:

- Circuit 1: $not((not(a)\ or\ b)\ and\ not(b))\ or\ not(a)$
- Circuit 2: $not(not(a\ and\ not(b))\ and\ not(b))\ or\ not(a))$

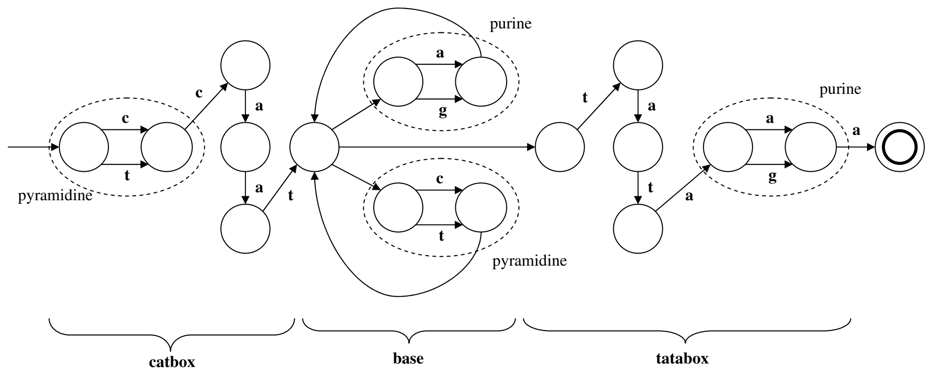
(b) The truth table for both circuits is then as below:

a	b	not(a)	not(b)	not(a) or b	(not(a) or b) and not(b)	not((not(a) or b) and not(b))	not((not(a) or b) and not(b)) or not(a)	a and not(b)	not(a and not(b))	not(a and not(b)) and not(b)	not(not(a and not(b)) and not(b))	not(not(a and not(b)) and not(b)) or not(a)
t	**t**	**f**	**f**	**t**	**f**	**t**	**t**	**f**	**t**	**f**	**t**	**t**
t	**f**	**f**	**t**	**f**	**f**	**t**	**t**	**t**	**f**	**f**	**t**	**t**
f	**t**	**t**	**f**	**t**	**f**	**t**	**t**	**f**	**t**	**f**	**t**	**t**
f	**f**	**t**	**t**	**t**	**t**	**f**	**t**	**f**	**t**	**t**	**f**	**t**
				Circuit 1				Circuit 2				

This is not direct bookwork.

(c) The two expressions are equivalent. Both expressions are tautologous - they give the output "true" regardless of the inputs. So the engineer was right on both counts. This is not direct bookwork.

6. (a) An appropriate FSM is as follows:



This is not direct bookwork. Note that the example above translates quite literally the specification in the question but the component for a base sequence can be simplified (since the specification allows any character sequence). Answers that spot this simplification are accepted (though not marked more highly than those with the more complex solution).

(b) This FSM can't be built because the base sequences can be of arbitrary length but are required to be maintained with equal numbers of the character "a". To enforce this constraint we would need some means of counting. This is bookwork re-applied to a new setting.

7. The basic method given in the lectures is this:

- First find all traces through the FSM that would accept the target string, noting the probability on each transition. For the example, there are two accepting traces for `aabb`:

```
1 a(0.6) 1 a(0.4) 2 b(0.5) 2 b(0.5) 3
1 a(0.6) 1 a(0.6) 1 b(0.2) 1 b(0.8) 3
```

- Then take the product of the probabilities on each trace:

```
0.6 * 0.4 * 0.5 * 0.5 = 0.06
0.6 * 0.6 * 0.2 * 0.8 = 0.0576
```

- Then take the sum of the probabilities per trace: `0.06 + 0.576 = 0.1176`

Variants on this calculation are acceptable as long as they make sense as calculations of probability. Students do not have to do the arithmetic - the equations suffice. This is bookwork.