Answers to Discrete maths Exercises

Thomas Forster

November 26, 2006

1988:6:9E (maths tripos)

The Master asked $2n+1$ people and got $2n+1$ different answers. Since the largest possible answer is $2n$ and the smallest is 0, there are in fact precisely $2n+1$ possible answers and that means he has got every possible answer from 0 up to $2n$ inclusive.

Think about the person who shook $2n$ hands. This person shook hands with everyone that they possibly could shake hands with : that is to say everyone except their spouse. So everybody except their spouse shook at least one hand. So their spouse shook no hands at all. Thus the person who shook $2n$ hands and the person who shook 0 hands are married. Henceforth disregard these two people and their handshakes and run the same argument to show that the person who shook $2n-1$ hands and the person who shook 1 hands are married. And so on.

Where does this get us? It tells us, after n iterations, that the person who shook $n+1$ hands and the person who shook $n-1$ hands are married. So what about the person who shook n hands, the odd man out? Well, it must be the odd *woman* out, because the only person of whom the Master asks this question who isn't married to another person of whom the Master asks this question is his wife.

Let's name people (other than the Master) with the number of hands they shook. (This is ok since they all shook different numbers of hands.) $2n$ didn't shake hands with its spouse, or itself, and there are only $2n$ people left, so it must have shaken hands with the Master. Correspondingly 0 didn't shake hands with anyone at all, so it certainly didn't shake hands with the Master. We continue reasoning in this way, about $2n-1$ and 1. $2n-1$ didn't shake hands with itself or its spouse or with 0, and that leaves only $2n-1$ people for it to shake hands with and since it shook $2n-1$ hands it must have shaken all of them, so in particular it must have shaken hands with the Master. Did 1 shake hands with the Master? No, because 1 shook only one hand, and that must have been $2n-1$'s. And so on. The people who shook the Master's hand were $2n$, $2n-1$, $2n-2 \ldots n+1$ and the people who didn't were 1, 2, 3, $\ldots n-1$. And of course, the Master's wife. So he shook n hands.

1988:6:10E (maths tripos)

Let R be a relation on a set X. Define the reflexive, symmetric and transitive closures $r(R)$, $s(R)$ and $t(R)$ of R. Let Δ be the relation $\{\langle x, x\rangle : x \in X\}$. Prove that

1. $R \circ \Delta = R$
2. $(R \cup \Delta)^n = \Delta \cup (\bigcup_{i \le n} R^i)$ for $n \ge 1$
3. $tr(R) = rt(R)$.

Show also that $st(R) \subseteq ts(R)$. If $X = \mathbb{N}$ and $R = \Delta \cup \{\langle x, y\rangle : y = px$ for some prime $p\}$ describe $st(R)$ and $ts(R)$.

The reflexive (symmetric, transitive) closure of R is the intersection of all reflexive (symmetric, transitive) relations of which R is a subset.

1. $R\Delta$ is R composed with the identity relation. x is related to y by R-composed-with-S if there is z such that x is related to z by R, and z is related to y by S. Thus $R\Delta = R$. (I would normally prefer to write '$R \circ \Delta$' here, using a standard notation for composition of relations: '$\circ$')

2. It is probably easiest to do this by induction on n. Clearly this is true for $n = 1$, since the two sides are identical in that case. Suppose it is true for $n = k$.
$$(R \cup \Delta)^k = \Delta \cup (\bigcup_{1 \le i \le k} R^i)$$
$(R \cup \Delta)^{k+1} = (R \cup \Delta)^k \circ (R \cup \Delta)$. By induction hypothesis this is
$$(\Delta \cup (\bigcup_{1 \le i \le k} R^i)) \circ (R \cup \Delta)$$
Now $(A \cup B) \circ (C \cup D)$ is clearly $(A \circ C) \cup (A \circ D) \cup (B \circ C) \cup (B \circ D)$ and applying this here we get
$$(\Delta \circ R) \cup (\Delta \circ \Delta) \cup ((\bigcup_{1 \le i \le k} R^i) \circ R) \cup ((\bigcup_{1 \le i \le k} R^i) \circ \Delta)$$
Now $\Delta \circ R$ is R; $\Delta \circ \Delta$ is Δ; $(\bigcup_{1 \le i \le k} R^i) \circ \Delta$ is $\bigcup_{1 \le i \le k} R^i$ and $(\bigcup_{1 \le i \le k} R^i) \circ R$ is $(\bigcup_{1 < i \le k+1} R^i)$ so we get
$$R \cup \Delta \cup (\bigcup_{1 < i \le k+1} R^i) \cup (\bigcup_{i \le k} R^i)$$
which is
$$\Delta \cup \bigcup_{1 \le i \le k+1} R^i$$

3. The transitive closure of the reflexive closure of R is the transitive closure of $R \cup \Delta$ which is $\bigcup_{n \in \mathbb{N}} (R \cup \Delta)^n$ which (as we have—more-or-less—just proved) is $\Delta \cup (\bigcup_{i \in \mathbb{N}} R^i)$ which is the reflexive closure of the transitive closure of R.

s is *increasing* so $R \subseteq s(R)$. t is *monotone*, so $t(R) \subseteq t(s(R))$. But the transitive closure of a symmetrical relation is symmetrical so $t(R) \subseteq t(s(R))$ implies $s(t(R)) \subseteq t(s(R))$ as desired.

Finally if $X = \mathbb{N}$ and $R = \Delta \cup \{\langle x, y \rangle : y = px \text{ for some prime } p\}$ then $st(R)$ is the relation that holds between two numbers when they are identical or one is a multiple of the other, and $ts(R)$ is the universal relation $\mathbb{N} \times \mathbb{N}$.

1990:1:9

Peter Dickman's model answer

We are asked to use generating functions to prove that:

$$c_n = \frac{1}{n} \left(\begin{array}{c} 2n-2 \\ n-1 \end{array} \right)$$

where c_n is the number of binary trees with n leaves (NB not n vertices) where no vertex has precisely one descendent. Now the formula given is remarkably similar to the one for Catalan numbers – which were introduced in the section of the course concerned with generating functions. So these may well be useful in answering this question.

Recall that for Catalan numbers:

$$C_n = \frac{1}{n+1} \left(\begin{array}{c} 2n \\ n \end{array} \right)$$

so it is (hopefully) clear that $C_n = c_{n+1}$.

To use generating functions it is necessary to find a recurrence relation...

Consider trees of the form described in the question. Clearly, any such tree which has more than one leaf can be viewed as being composed of two trees joined together by a single (new) root vertex, whose descendents are the two roots of the component smaller trees. Now the sum total of leaves in these subtrees will be the same as the number of leaves in the composite; and each tree will have at least one leaf. So, the number of trees with some given number of leaves can be determined by considering all of the ways such a tree can be split into left & right subtrees, and the parts combined together.

It follows that:

$$\forall n \geq 1 : c_{n+1} = c_1 c_n + c_2 c_{n-1} + c_3 c_{n-2} + \ldots + c_n c_1$$

Note that we have $n \geq 1$ in the above, because the equation is giving us an expression for $n+1$. The recurrence only holds for the trees with two or more leaves (as we assumed that the root had two descendents).

Also we know that $c_1 = 1$ by inspection.

Note that I've written this out for the case $n+1$ not, as I would normally do, the case n because it makes everything neater later. The result can be achieved from the n case but is a bit messier. The only hint I can give as to how to tell that this is helpful **in advance** is that we already knew that there was an "off by one" effect present in this question.

Now, let us consider $d_n = c_{n+1}, \forall n \geq 0$. Then we have that:

$$\forall n \geq 1 : d_n = d_0 d_{n-1} + d_1 d_{n-2} + \ldots + d_{n-1} d_0$$

Now, if we define $d_k = 0, \forall k < 0$ then we have that

$$\forall n \geq 1 : d_n = \sum_{i=0}^{n-1} d_i d_{n-1-i} = \sum_{i=0}^{\infty} d_i d_{n-1-i}$$

Now, the generating function for the d_n, called $D(z)$ say, has the property that the coefficient of z^n in $D(z)$ is d_n. So we have that:

$$[z^n]D(z) = \begin{cases} 1 & \text{if } n = 0 \\ \sum_{i=0}^{\infty} d_i d_{n-1-i} & \text{otherwise} \end{cases}$$

Whence we derive:

$$\begin{aligned} D(z) &= \sum_{n=0}^{\infty} d_n z^n \\ &= 1 + \sum_{n=1}^{\infty} \sum_{i=0}^{\infty} d_i d_{n-1-i} z^n \\ &= 1 + \sum_{n=1}^{\infty} \sum_{i=0}^{\infty} d_i d_{n-1-i} z^i z^{n-1-i} z \\ &= 1 + z \sum_{n=1}^{\infty} \sum_{i=0}^{\infty} d_i d_{n-1-i} z^i z^{n-1-i} \\ &= 1 + z \sum_{n=1}^{\infty} \sum_{i=0}^{\infty} d_i z^i \cdot d_{n-1-i} z^{n-1-i} \\ &= 1 + z(D(z))^2 \end{aligned}$$

since the penultimate line is a convolution.

This is a formula we recognise from the Catalan numbers, so we proceed by following the same argument as in the lecture notes...

Reorganising this gives us:

$$z(D(z))^2 - D(z) + 1 = 0$$

Solving this we find that:

$$D(z) = \frac{1 \pm \sqrt{1 - 4z}}{2z}$$

Since d_n is non-negative $\forall n$ and since $\sqrt{1-4z}$ has only negative signs after the first term we can eliminate the form with an addition in and find:

$$D(z) = \frac{1 - \sqrt{1-4z}}{2z}$$

Which, from a standard binomial identity leads us to:

$$D(z) = \frac{1 - \sqrt{1-4z}}{2z} = \sum_{k \geq 0} \frac{1}{k+1} \left(\begin{array}{c} 2k \\ k \end{array} \right) z^k$$

So we find that:

$$d_n = [z^n]D(z) = \frac{1}{n+1} \left(\begin{array}{c} 2n \\ n \end{array} \right)$$

However, $d_n = c_{n+1}, \forall n \geq 0$ therefore we have that:

$$\forall n \geq 1 : c_n = \frac{1}{n} \left(\begin{array}{c} 2n-2 \\ n-1 \end{array} \right)$$

as required.

Note that the formula is obviously useless for $n = 0$ as it would give $c_0 = \infty$ so we clearly aren't being expected to worry about that case. However it might be worth pointing this out

The second part of the question asks how many trees of the form considered, with n leaves, have depth $n-1$. Again let's look for a recurrence relation. I'll skip through this fairly quickly... I suggest that you draw some pictures as you read through this. Be aware of the assumption that $n > 2$ in the following.

Consider the trees of this form, that have n leaves and are of depth $n-1$, for arbitrary $n > 2$. Given such a tree, let the number of leaves at depth $n-1$ (ie the maximal depth) be k.

From such a tree we can construct k distinct trees of depth n which have $n+1$ leaves by taking one of the leaves at the $n-1$ level and replacing it with a vertex with two descendents, which are themselves leaves.

Now consider a tree of depth n with $n+1$ leaves, satisfying the condition on numbers of descendents. Selecting any leaf at the maximal depth, its parent is at depth $n-1$ and, by the condition on numbers of descendents, this has another child at depth n. Replacing these two leaves and their parent vertex with a single leaf at depth $n-1$ we either construct a tree with n leaves of depth $n-1$ (if we have removed the only pair of leaves at the maximal depth) or we have a tree of n leaves of depth n.

However the depth of one of our trees must be strictly less than the number of leaves. Assume otherwise, ie that for some such tree, the number of leaves is less than or equal to the depth. Since each 'plucking' operation of the form described above reduces the number of leaves by one and the number of levels

by at most one, we would be able to construct a tree with 2 leaves and depth of at least 2 – which is clearly impossible.

So, we have shown that each such tree has precisely 2 leaves at its terminal level, and that the only possible constructions are the k variants of each of the trees of one smaller size. But k is the number of leaves at the terminal level *i.e.* 2, so we have a doubling of the number of possible trees at each level. Given that there are $1 = 2^0$ trees with 2 leaves of depth 1, $2 = 2^1$ trees with 3 leaves of depth 2 and so forth we have that,

$$\forall n \geq 2, \exists\, 2^{n-2} \text{ trivalent trees with } n \text{ leaves and depth } n-1$$

1990:1:11

Equivalence relations correspond to partitions. A PER $\langle X, R\rangle$ that fails to be an equivalence relation features elements $x \in X$ such that $\langle x, x\rangle \notin R$. Such elements are not related to anything at all, since if x is related to y, then by symmetry y is related to x and by transitivity x is related to x.

So we put on one side all the $x \in X$ such that $\langle x, x\rangle \notin R$, leaving behind a subset $X' \subseteq X$ consisting of all those elements related to themselves by R. What is the restriction of R to this set? Clearly it is reflexive. Actually it is transitive and symmetrical as well, because $X' \times X'$ is transitive and symmetrical, and transitivity and symmetry are intersection-closed properties so $R \cap (X' \times X')$ will be transitive and symmetrical. So $R \cap (X' \times X')$ is an equivalence relation on X'.

How many PER's on a set with 4 elements?

There is one way of throwing away no elements, leaving 4. These can be either

all in one piece	1
one singleton, one triple	4
two pairs	3
two singletons	6
all singletons	1

There are 4 ways of throwing away one element, leaving 3. These can be either

all in one piece	$1 \times 4 = 4$
one singleton, one pair	$3 \times 4 = 12$
three singletons,	$1 \times 4 = 4$

There are 6 ways of throwing away two elements, leaving 2. These can be either

all in one piece	$1 \times 6 = 6$
two singletons	$1 \times 6 = 6$

There are 4 ways of throwing away 3 elements leaving 1.

This can be partitioned in only one way	$1 \times 4 = 4$
	$\overline{52}$

Let us prove that T is a PER. We first show that it is transitive. Suppose
(i) $\langle f, g\rangle \in T$ and
(ii) $\langle g, h\rangle \in T$. We want $\langle f, h\rangle \in T$.

By definition of T we infer

(iii) $(\forall x_1, x_2 \in X)(\langle x_1, x_2\rangle \in R \rightarrow \langle f`x_1, g`x_2\rangle \in S)$ and

(iv) $(\forall x_2, x_3 \in X)(\langle x_2, x_3\rangle \in R \rightarrow \langle g`x_2, h`x_3\rangle \in S)$. (We have relettered variable to make life easier)

Now let x_1 and x_3 be two elements of X such that $\langle x_1, x_3\rangle \in R$. We want to infer $\langle f(x_1), g(x_3)\rangle \in S$. R is symmetrical so $\langle x_3, x_1\rangle \in R$ too. So by transitivity we have $\langle x_1, x_1\rangle \in R$. By (iii) we can infer $\langle f(x_1), g(x_1)\rangle \in S$. We now use (iv) on our assumption that x_1 and x_3 are two elements of X such that $\langle x_1, x_3\rangle \in R$ to infer that $\langle g(x_1), h(x_3)\rangle \in S$. Finally, by transitivity of S we infer that $\langle f(x_1), h(x_3)\rangle \in S$ as desired.

It is much easier to show that T is symmetric. Suppose $\langle f, g\rangle \in T$ and let x_1 and x_2 be two elements of X such that $\langle x_1, x_2\rangle \in R$. We want to infer $\langle f(x_1), g(x_2)\rangle \in S$. R is symmetric, so we infer $\langle x_2, x_1\rangle \in R$, whence $\langle f(x_1), g(x_2)\rangle \in S$ as desired.

To show that T is not in general reflexive, even if R and S both are, take R to be the universal relation on X and S to be the identity relation on Y, where both X and Y have at least two members.

1993:11:11

$\langle A, \leq \rangle$ is a partially ordered set if

1. $(\forall x, y, z \in A)(x \leq y \rightarrow (y \leq z \rightarrow x \leq z)$ ($\leq$ is transitive)
2. $(\forall x, y \in A)(x \leq y \rightarrow (y \leq x \rightarrow x = y))$ ($\leq$ is antisymmetrical)
3. $(\forall x \in A)(x \leq x)$ ($\leq$ is reflexive)

In what follows we write '$x < y$' for '$x \leq y \wedge y \neq x$'

(a) If $\langle A, \leq \rangle$ is to form a totally ordered set then in addition $\leq$ must satisfy *connexity*.

$$(\forall x, y \in A)(x \leq y \vee y \leq x)$$

or equivalently $<$ must satisfy *trichotomy*

$$(\forall x, y \in A)(x < y \vee x = y \vee y < x)$$

(b) If $\langle A, \leq \rangle$ is to be wellfounded then in addition $<$ (which is the strict version of $\leq$, namely $\{\langle x, y \rangle : x \leq y \wedge x \neq y\}$) must satisfy *wellfoundedness*:

$$(\forall A' \subseteq A)(\exists x \in A')(\forall y \in A')(y \not< x)$$

(This *détour* via strict partial orders is necessary beco's no wellfounded relation can be reflexive.)

(c) If $\langle A, \leq \rangle$ is to be a complete partially ordered set then one of the following conditions on $\leq$ must be satisfied, depending on what your definition of complete poset is:

One definition is that every *subset* of A must have a least upper bound in the sense of $\leq$. This is

$$(\forall A' \subseteq A)(\exists x \in A)[(\forall y \in A')(y \leq x) \wedge (\forall z \in A)((\forall y \in A')(y \leq z) \rightarrow x \leq z)]$$

... or that every directed subset of A has a least upper bound. A' is a directed subset of A if $(\forall x, y \in A')(\exists z \in A')(x \leq z \wedge y \leq z)$. (They probably don't mean that tho'.)

To show that the restriction of a partial order of A to some subset B of A is a partial order of B we have to check that $R \cap (B \times B)$ is reflexive transitive and antisymmetrical. Now $B \times B$ is reflexive and transitive, as is R; reflexivity and transitivity are intersection-closed properties, so $R \cap (B \times B)$ is reflexive and transitive. To verify antisymmetry we have to check that if $\langle x, y\rangle$ and $\langle y, x\rangle$ are both in $R \cap (B \times B)$ then $x = y$. But if $\langle x, y\rangle$ and $\langle y, x\rangle$ are both in $R \cap (B \times B)$ then they are both in R, and we know R is antisymmetrical, whence $x = y$ as desired.

(A deeper proof can be obtained by noting only that all the clauses in the definition of partial order are universal. Any universal sentence true in A is true in any subset of A. After all, a universal sentence is true as long as there is no counterexample to it. If A contains no counterexamples, neither can any subset of A. This shows that a substructure of a total order is a total order which is useful later on in the question ...)

$\mathbf{Z}$ (i) $\leq$ is a partial order of $\mathbf{Z}$. Indeed it is a total order. (ii) It isn't well-founded (e.g.: no bottom element) nor (iii) is it a complete poset (e.g.: no top element).

Divisibility (i) is not a partial order because for any integer n, n and $-n$ divide each other but are distinct, so it isn't antisymmetrical. (ii) The relation "n divides m but not *vice versa*" is wellfounded on $\mathbf{Z}$ however. If $X \subseteq \mathbf{Z}$, then its minimal elements under "n divides m but not *vice versa*" are precisely the minimal elements of $\{|n| : n \in X\}$ under "n divides m but not *vice versa*", and this relation, being a subset of a wellfounded relation (and $\leq$ *is* wellfounded on $\mathbb{N}$) is itself wellfounded. (iii) $\mathbf{Z}$ is not a complete poset under divisibility for the same reason as before.

$\mathbb{N}$ $\leq$ is a partial order of $\mathbb{N}$. Indeed it is a total order. It is also wellfounded but it is not a complete poset (as before)

Divisibility is a partial order on $\mathbb{N}$ but not a total order, it is wellfounded. This time we do get a complete poset, because everything divides 0.

$\mathbb{N}^+$ As for $\mathbb{N}$ except that it is not a complete poset (e.g.: no top element)

1994:10:11

The way to do part 2 is to stop trying to be clever and do it the easy way. Let A_n, B_n, C_n be the number of valid strings in $\{A, B, C\}^n$ ending in A, B and C respectively. Clearly

$$C_{n+1} = A_n + B_n + C_n$$

and

$$A_{n+1} = B_{n+1} = B_n + C_n$$

This is because if the last character of a legal string is an A or a B then the penultimate character cannot be an A. We are not going to try to do anything clever like *derive* the equality we have been given, but we can at least confirm it! So let's try to simplify

$$2(A_{n+1} + B_{n+1} + C_{n+1}) + A_n + B_n + C_n$$

and hope that it simplifies to $A_{n+2} + B_{n+2} + C_{n+2}$.

Take out $B_{n+1} + C_{n+1}$ twice to give $A_{n+2} + B_{n+2}$, leaving $2A_{n+1} + A_n + B_n + C_n$. The last three terms add up to C_{n+1}, and $2A_{n+1} = A_{n+1} + B_{n+1}$ so this is $A_{n+1} + B_{n+1} + C_{n+1}$ which is C_{n+2}. Together with the $A_{n+2} + B_{n+2}$ this adds up to $v(n + 2)$ as desired.

Part 3 is 'A'-level maths that you remember from your crèche.

1995:5:4X (maths 1a)

Well, adapted from it!

```
fun f n = if n = 0 then 0 else g(f(n-1) + 1, 1) -1
and g(n,m) = f(f(n-1)) + m + 1;
```

What are the ML types of these two functions?

What are the running times of f and g?

By inspection we notice that $(\forall n \in \mathbb{N})(f(n) = n)$, but we had better prove it! It's true for $n = 0$. For the induction step the recursive declaration tells us that

$$f(n+1) = g(f(n)+1, 1) - 1 \text{ (by substituting } n+1 \text{ for } n)$$

But $f(n) = n$ by induction hypothesis so this becomes

$$f(n+1) = g(n+1, 1) - 1$$

Now, substituting $(n+1)$ for n and 1 for m in the declaration for `g` we get

$$g(n+1, 1) = (n+1-1) + 1 + 1$$

which is $n + 2$ giving $f(n+1) = n + 1$ as desired.

(d)

The mutual recursion gives us a pair of mutual recurrence relations:

A: $F(n) = G(f(n-1)+1, 1) + F(n-1)$
B: $G(n, m) = F(n-1) + F(f(n-1)) + k$

where F is the cost function for f and G is the cost function for g.

Using $f(n) = n$ we can simplify our recurrence relations as follows.

A′: $F(n) = G(n, 1) + F(n-1)$
B′: $G(n, m) = F(n-1) + F(n-1) + k$ whence
B″: $G(n, m) = 2 \cdot F(n-1) + k$

This gives

$$F(n) = F(n-1) + F(n-1) + F(n-1) + k$$

so $F(n)$ grows like 3^n.

G is exponential too. We have assumed that the cost of adding the second agument ('m') is constant, but altho' this simplification will cause no problems it is a simplification nevertheless. Adding two arguments takes time proportional to the logarithm of the larger of the two. Fortunately the cost functions of these algorithms are so huge that an extra log or two will make no difference to the order.

1996:1:7

A partial ordering is a relation that is reflexive, antisymmetrical and transitive.

'Topological sort' is Compsci jargon for refining a partial ordering, which just means adding ordered pairs to a partial ordering to get a total ordering. The two partial orders of $\mathbb{N} \times \mathbb{N}$ that you have seen are the **pointwise product** ($\langle x, y\rangle \leq_p \langle x', y'\rangle$ iff $x \leq x' \wedge y \leq y'$) and the **lexicographic product** ($\langle x, y\rangle \leq_{lex} \langle x', y'\rangle$ iff $x < x' \vee (x = x' \wedge y \leq y')$). The second is clearly a refinement of the first. It is also clear that the lexicographic product $\mathbb{N} \times \mathbb{N}$ is not isomorphic to $\mathbb{N}$ in the usual ordering, since it consists of ω copies of $\mathbb{N}$. (ω is the length of $\mathbb{N}$ in its usual ordering: the length of $\mathbb{N} \times \mathbb{N}$ in the product ordering is therefore said to be ω^2).

To get a refinement of the product ordering of $\mathbb{N} \times \mathbb{N}$ that is isomorphic to the usual ordering on $\mathbb{N}$ we notice that for a wellordering to be isomorphic to the usual ordering on $\mathbb{N}$ it is sufficient for each point to have only finitely many things below it (given that is also a wellordering, that is). Try $\langle x, y\rangle \leq \langle x', y'\rangle$ iff $(x+y) < (x'+y') \vee (x+y = x'+y' \wedge x \leq x')$. It's a total order, each element has only finitely many things below it (so it's isomorphic to the usual order on $\mathbb{N}$) and it refines the pointwise product ordering.

1996:1:8

The recurrence

$$R: \ w(n,k) = w(n-2^k, k) + w(n, k-1)$$

can be justified as follows. Every representation of n pfatz as a pile of coins of size no more than 2^k pfatz either contains a 2^k pfatz piece or it doesn't. Clearly there are $w(n, k-1)$ representations of n pfatz as a pile of coins of size no more than 2^{k-1} pfatz so that's where the $w(n, k-1)$ comes from. The other figure arises from the fact that a representation of n pfatz as a pile of coins of size no more than 2^k pfatz and containing a 2^k pfatz piece arises from a representation of $n - 2^k$ pfatz as a pile of coins of size no more than 2^k.

Base case. $w(n, 0) = 1$. That should be enough.

To derive $w(4n, 2) = (n+1)^2$, substitute $4n$ for n, and 2 for k in R, getting

$$w(4n, 2) = w(4n - 2^2, 2) + w(4n, 1)$$

But this rearranges to

$$w(4n, 2) = w(4(n-1), 2) + w(4n, 1)$$

$w(4n, 1)$ is $2n + 1$, since we can have between 0 and $2n$ 2-pfatz pieces in a representation of $4n$. This gives

$$w(4n, 2) = w(4(n-1), 2) + 2n + 1$$

This is a bit clearer if we write this as $f(n) = f(n-1) + 2n + 1$. This recurrence relation obviously gives $f(n) = (n+1)^2$ as desired.

We can always get an estimate of $w(n,k)$ by applying equation R recursing on n, and this works out quite nicely if n is a multiple of 2^k because then we hit 0 exactly, after $n/(2^k)$ steps. Each time we call the recursion we add $w(n,k-1)$ (or rather $w(n-y,k-1)$ for various y) and clearly $w(n,k-1)$ is the biggest of them. So $w(n,k)$ is no more than $n/(2^k) \cdot w(n,k-1)$.

Finally, using R with 2^{k+1} for n again we get $w(2^{k+1},k) = w(2^k,k) + w(2^{k+1},k-1)$. The hint reminds us that every representation of 2^k pfatz using the first k coins gives rise to a representation of 2^{k+1} pfatz using the first $k+1$ coins. Simply double the size of every coin. It's also true that every representation of 2^k pfatz using the first k coins gives rise to a representation of 2^{k+1} pfatz using the first $k+1$ coins by just adding a 2^k pfatz piece. The moral is: $w(2^{k+1},k+1) = 2 \cdot w(2^k,k)$. This enables us to prove the left-hand inequality by induction on k.

To prove the right-hand inequality we note that any manifestation of 2^k pfatz using smaller coins can be tho'rt of as a list of length k where the ith member of the list tells us how many 2^i pfatz coins we are using. How many lists of length k each of whose entries are at most 2^k are there? Answer $(2^k)^k$, which is 2^{k^2}.

1997:1:2

1997:1:7

(a) Yes: equality is a partial order, and it is tree-like beco's the set of strict predecessors is always empty.

(b) Yes. The usual order is a partial (indeed *total*) order and every total order is tree-like.

(c) No. This is a partial order but is not tree-like beco's (for example) 6 has two immediate strict predessors.

(d) This is reflexive and antisymmetrical (if xRy and yRx so that x and y are either equal or each is the greatest prime factor of the other then they are equal). The hard part is to show that it is transitive. Suppose xRy and yRz. If $x = y$ or $z = z$ we deduce xRz at once so consider the case where xRy and yRz hold *not* in virtue of $x = y$ or $y = z$. But this case cannot arise, because if yRz and $y \neq z$, then y is a prime, and the only x such that xRy is y itself. Finally, it's easy to show this relation is tree-like, because no number can have more than one greatest prime factor.

It seem to me that the number of treelike partial orderings of n elements is precisely $n!$. Each treelike partial ordering of n chaps gives rise to n new partial orderings beco's the extra chap can be stuck on top of any of the n things already there. No new partial ordering gets counted twice.

2002:1:8

The last part seems to have caused problems for some. Let's have a look.

We are contemplating relations that hold between elements of Ω and subsets of Ω. An example of the sort of thing the examiner has in mind is the relation that a point y in the plane bears to a (typically non-convex) region X when y is in the convex hull of X. (A picture would be nice at this point!) The idea is that y one of the points you have to "add" to obtain something convex. (Check that you know what a convex set is, as i'm going to procede on the assumption that you do, and use it as a—one hopes!—illuminating illustration)

What is $\mathcal{R}$? $\mathcal{A}$ is an intersection-closed family of subsets of Ω. (As it might be, the collection of convex subsets of the plane). We are told that it is the relation that relates y to X whenever anything in A that extends X also contains y. In our illustration—where $\mathcal{A}$ is the collection of convex subsets of the plane— $\mathcal{R}$ is the relation that hold between X and y whenever y is in the convex hull of X. Certainly in this case any set that is $\mathcal{R}$-closed is convex.

Assume C is $\mathcal{R}$-closed. That is to say

$$\forall(X, y) \in \mathcal{R}.X \subseteq C \rightarrow y \in C \tag{1}$$

But $\mathcal{R} = \{(X, y) \in \mathcal{P}(\Omega) \times \Omega | \forall A \in \mathcal{A} X \subseteq A \rightarrow y \in A\}$. Substituting this for '$\mathcal{R}$' in 1 we obtain

$$\forall (X, y) \in \{(X, y) \in \mathcal{P}(\Omega) \times \Omega | \forall A \in \mathcal{A} X \subseteq A \rightarrow y \in A\}. X \subseteq C \rightarrow y \in C \quad (2)$$

which reduces to

$$\forall (X, y)[(\forall A \in \mathcal{A})(X \subseteq A \rightarrow y \in A) \wedge X \subseteq C. \rightarrow y \in C] \quad (3)$$

The examiners suggest you should consider the set $\{A \in \mathcal{A} : C \subseteq A\}$. I think they want you to look at $\bigcap\{A \in \mathcal{A} : C \subseteq A\}$.

If you've followed the action this far you would probably think of this anyway, since this is a set that you know must be in $\mathcal{A}$ and it seems to stand an outside chance of being equal to C. So let's look again at 3 to see if it does, in fact, tell us that $\bigcap\{A \in \mathcal{A} : C \subseteq A\}$ is C.

And—of course—it does. First we instantiate 'X' to 'C' in 3 to obtain:

$$\forall y[(\forall A \in \mathcal{A})(C \subseteq A \rightarrow y \in A) \rightarrow y \in C] \quad (4)$$

Now let y be an arbitrary member of $\bigcap\{A \in \mathcal{A} : C \subseteq A\}$. That means that y satisfies the antecedent of 4. So it satisifes the consequent of 4 as well. So we have proved that $\bigcap\{A \in \mathcal{A} : C \subseteq A\}$ is a subset of C. It was always a superset of C, so it is equal to C. So $C \in \mathcal{A}$ as desired.

Question ??.??

Show that $\bigcup_{n \in \mathbb{N}} R^n$ is the smallest transitive relation extending R.

To do this it will be sufficient to show

1. $\bigcup_{n \in \mathbb{N}} R^n$ is transitive
2. If S is a transitive relation $\supset R$ then $\bigcup_{n \in \mathbb{N}} R^n \subseteq S$

For (1) We need to show that if $\langle x, y\rangle$ and $\langle y, z\rangle$ are both in $\bigcup_{n \in \mathbb{N}} R^n$ then $\langle x, z\rangle \in \bigcup_{n \in \mathbb{N}} R^n$. If $\langle x, y\rangle \in \bigcup_{n \in \mathbb{N}} R^n$ then $\langle x, y\rangle \in R^k$ for some k and if $\langle y, z\rangle \in \bigcup_{n \in \mathbb{N}} R^n$ then $\langle y, z\rangle \in R^j$ for some j. Then $\langle x, z\rangle \in R^{j+k} \subseteq \bigcup_{n \in \mathbb{N}} R^n$.

For (2) Let $S \supset R$ be a transitive relation. So $R \subseteq S$. We prove by induction on $\mathbb{N}$ that for all $n \in \mathbb{N}$, $R^n \subseteq S$. Suppose $R^n \subseteq S$. Then

$$R^{n+1} = R^n \circ R \ \subseteq^{(a)} \ S \circ R \ \subseteq^{(b)} \ S \circ S \ \subseteq^{(c)} \ S.$$

(a) and (b) hold because $\circ$ is *monotone*: if $X \subseteq Y$ then $X \circ Z \subseteq Y \circ Z$. (c) holds because S is transitive.

Question ??.??

Let R and S be equivalence relations. We seek the smallest equivalence relation that is a superset of $R \cup S$. We'd better note first that this really is well defined, and it is, because being-an-equivalence-relation is the conjunction of three properties all of them intersection closed, so it is itself intersection-closed.

This least equivalence relation extending $R \cup S$ is at least transitive, so it must be a superset of $t(R \cup S)$, the transitive closure of $R \cup S$. Wouldn't it be nice if it actually *were* $t(R \cup S)$? In fact it is, and to show this it will be sufficient to show that $t(R \cup S)$ is an equivalence relation. Must check: transitivity, reflexivity and symmetry. Naturally $t(R \cup S)$ is transitive by construction. R and S are reflexive so $R \cup S$ is reflexive. In constructing the transitive closure we add new ordered pairs but we never add ordered pairs with components we haven't seen before. This means that we never have to add any ordered pairs $\langle x, x \rangle$ beco's they're all already there. Therefore $t(R \cup S)$ is reflexive as long as R and S are. Finally we need to check that $t(R \cup S)$ is symmetrical. The transitive closure of a symmetrical relation is also symmetrical. First we show by induction on n that R^n is symmetrical as long as R is. Easy when $n = 1$. Suppose R^n is symmetrical: i.e., $R^n = (R^n)^{-1}$. $R^{n+1} = R \circ R^n$ anyway. The inverse of this is $(R^{-1})^n \circ R^{-1}$. $(R^{-1})^n$ is of course R^{-n}, so $(R^{-1})^n \circ R^{-1}$ is $(R^{-n}) \circ R^{-1}$. $R^{-n} = R^n$ by induction hypothesis so $(R^{-1})^n \circ R^{-1}$ is $R^n \circ R$ which is of course R^{n+1}. Then the union of a lot of symmetrical relations is symmetrical, so the transitive closure (which is the union of all the (symmetrical) iterates of R) is likewise symmetrical.

Actually we can give another—perhaps simpler—proof of this. $t(R) = \bigcap\{S : R \subseteq S \wedge S^2 \subseteq S\}$, or $\bigcap X$ for short. Notice that R is symmetrical, then X is closed under taking inverses (the inverse of anything in X is also in X). And clearly the intersection of a class closed under taking inverses is symmetrical.

Question ??.??

Show that if R and S are transitive relations, so is $R \cap S$.

$(R \cap S) \circ (R \cap S) \subseteq R \circ R \subseteq R$
$(R \cap S) \circ (R \cap S) \subseteq S \circ S \subseteq S$
so
$(R \cap S) \circ (R \cap S) \subseteq R \cap S$

Notice that the same argument shows that the intersection of any number of transitive relations is a transitive relation: i.e., transitivity is an intersection closed property of relations.

Question ??.??

> Let R be a relation on A. ('r', 's' and 't' denote the reflexive, symmetric and transitive closure operations respectively.)
>
> 1. Prove that $rs(R) = sr(R)$.

2. Does R transitive imply $s(R)$ transitive?
3. Prove that $rt(R) = tr(R)$ and $st(R) \subseteq ts(R)$.
4. If R is symmetrical must the transitive closure of R be symmetrical? Prove or give a counterexample.
5. Think of a binary relation R, and of its graph, which will be a directed graph $\langle V, E\rangle$. On any directed graph we can define a relation "I can get from vertex x to vertex y by following directed edges" which is certainly transitive, and we can pretend it is reflexive because after all we can get from a vertex to itself by just doing nothing at all. Do this to our graph $\langle V, E\rangle$, and call the resulting relation S. How do we describe S in terms of R?

(a) Prove that $rs(R) = sr(R)$:

$$\begin{aligned} r(s(R)) &= s(R) \cup I \\ &= (R \cup R^{-1}) \cup I \\ &= (R \cup I) \cup (R^{-1} \cup I) \\ &= (R \cup I) \cup (R^{-1} \cup I^{-1}) \\ &= (R \cup I) \cup (R \cup I)^{-1} \\ &= s(r(R)) \end{aligned}$$

(b) The symmetric closure of a transitive relation is not automatically transitive: take R to be set inclusion on a power set.

(c) Prove that $rt(R) = tr(R)$:

$$\begin{aligned} r(t(R)) &= t(R) \cup I = R \cup R^2 \cup \ldots R^n \ldots \cup I \\ &= (R \cup I) \cup (R^2 \cup I) \cup (R^n \cup I) \ldots \end{aligned}$$

At this point it would be nice to be able to say $(R^n \cup I) = (R \cup I)^n$ but this isn't true. $(R \cup I)^n$ is actually $R \cup R^2 \ldots R^n \cup I$. But this is enuff to rewrite the last line as

$$(R \cup I) \cup (R \cup I)^2 \cup (R \cup I)^3 \ldots$$

which is of course $t(r(R))$ as desired.

The transitive closure of a symmetrical relation is also symmetrical. First we show by induction on n that R^n is symmetrical as long as R is. Easy when $n = 1$. Suppose R^n is symmetrical. $R^{n+1} = R \circ R^n$. The inverse of this is $(R^{-1})^n \circ R^{-1}$ which by induction hypothesis is $R^n \circ R$ which is of course R^{n+1}. Then the union of a lot of symmetrical relations is symmetrical, so the transitive closure (which is the union of all the (symmetrical) iterates of R is likewise symetrical.

Finally S is the reflexive transitive closure of R.

Question ??.??

> Show that—at least if $(\forall x)(\exists y)(\langle x, y\rangle \in R)$—$R \circ R^{-1}$ is a fuzzy. What about $R \cap R^{-1}$? What about $R \cup R^{-1}$?

If $\langle x, y\rangle \in R$ then $\langle y, x\rangle \in R^{-1}$ so $\langle x, x\rangle \in R \circ R^{-1}$. That takes care of reflexivity. Suppose $\langle x, z\rangle \in R \circ R^{-1}$. Then there is a y such that $\langle x, y\rangle \in R$ and $\langle y, z\rangle \in R^{-1}$. But then $\langle z, y\rangle \in R$. So $\langle x, z\rangle \in R \circ R^{-1}$ is the same as $(\exists y)((\langle x, y\rangle \in R) \wedge (\langle z, y\rangle \in R))$. But this is clearly symmetric in x and z, so we can rearrange it to get $(\exists y)((\langle y, x\rangle \in R^{-1}) \wedge (\langle z, y\rangle \in R))$ which is $\langle z, x\rangle \in R \circ R^{-1}$ as desired.

$R \cup R^{-1}$ is the symmetric closure of R and is of course symmetric, but there is no reason to expect it to be reflexive: it'll be reflexive iff R is reflexive.

Question ??.??

> Given any relation R there is a least $T \supseteq R$ such that T is transitive, and a least $S \supseteq R$ such that S is symmetrical, namely the transitive and symmetric closures of R. Must there also be a maximal $S \subseteq R$ such that S is transitive? And must there be a maximal $S \subseteq R$ such that S is symmetrical? The answer to one of these last two questions is 'yes': find a cute formulation.

$R \cap R^{-1}$ is the largest symmetrical relation included in R. The unwary sometimes think *this* is the symmetric closure of R. The point is that altho' being-the-complement-of-a-transitive-relation is not an intersection-closed property, nevertheless being-the-complement-of-a-symmetric-relation **is** intersection-closed, since it is the same as being symmetrical. $R \cap R^{-1}$ is the complement of the symmetric closure of the complement of R. Do not confuse complements with converses!!

Question ??.??

If $x \leq S(y)$ and $y \leq S(x)$ then x and y are neighbouring naturals. This is $R \cap R^{-1}$. x and y are related by the transitive closure of this relation iff there is a finite sequence $x_0, x_1, x_2 \ldots x_n = y$ such that each x_i is adjacent to x_{i+1}. But clearly any two naturals are connected by such a chain, so the transitive closure is the universal relation. For part 2, remember that x is related to y by $R \setminus R^{-1}$ if it is related to y by R but not by R^{-1}. In this case that means $x \leq S(y) \wedge y \not\leq S(x)$. This is $x \leq S(y) \wedge S(x) < y$. The second conjunct implies the first so we can drop the first, getting $S(x) < y$. Getting the transitive closure of this is easy, 'cos it's transitive already!

Question ??.??

> Are the two following conditions on partial orders equivalent?

1. $(\forall xyz)(z > x \not\leq y \not\leq x \rightarrow z < y)$
2. $(\forall xyz)(z > x \not\leq y \not\leq x \rightarrow z > y)$.

Assume (i) $(\forall xyz)(z \leq x \not\leq y \not\leq x \rightarrow z \leq y)$ and aim to deduce (ii) $(\forall xyz)(z \geq x \not\leq y \not\leq x \rightarrow z \geq y)$. To this end assume $z \geq x$, $x \not\leq y$ and $y \not\leq x$ and hope to deduce $z \geq y$.

$x \leq z$ tells us that $z \not\leq y$ for otherwise $x \leq y$ by transitivity, contradicting hypothesis. Next, assume the negation of what we are trying to prove. This gives us $y \not\leq z$. But then we have $y \not\leq z \not\leq y$ and $x \leq z$ so by (i) we can infer $x \leq y$, contradicting assumption.

I think the proof in the other direction is similar but i haven't checked it.

For the record: to any partial order there corresponds in a obvious way a strict partial order. (like $\leq$ and $<$ on $\mathbb{N}$, for example.) Consider the strict partial order corresponding to a partial order satisfying this condition we have just been discussing. If it is wellfounded it is said to be a **prewellordering**. This is because we can think of it as a total ordering of the equivalence classes (under the relation $x \simeq y$ iff $x = y \vee x \not\leq y \not\leq x$), and if $<$ is wellfounded this in in fact a wellordering of the equivalence classes.

Question ??.??

Show that $R \subseteq S$ implies $R^{-1} \subseteq S^{-1}$

The way to do this is to assume that $R \subseteq S$ and let $\langle x, y\rangle$ be an arbitrary ordered pair in R^{-1}. We then want to infer that $\langle x, y\rangle$ is in S^{-1}.

If $\langle x, y\rangle$ is in R^{-1} then $\langle y, x\rangle$ is in R, because R^{-1} is precisely the set of ordered pairs $\langle x, y\rangle$ such that $\langle y, x\rangle$ is in R. (We would write this formally as: $R^{-1} = \{\langle x, y\rangle : \langle y, x\rangle \in R\}$.) But $R \subseteq S$, so $\langle y, x\rangle$ is in S, and so (flip things round again) $\langle x, y\rangle$ is in S^{-1}.

Notice that to tell this story successfully we have to come out of the closet and think of R and S as sets of ordered pairs, that is, as relations-in-extension.

Question ??.??

Show that $R \circ (S \circ T) = (R \circ S) \circ T$

That is to say $xR \circ (S \circ T)y$ iff $x(R \circ S) \circ Ty$

Now, by definition of relational composition,

$$xR \circ (S \circ T)y$$

is

$$(\exists z)(xRz \wedge z(S \circ T)y)$$

and expand the second '$\circ$' to get

$$(\exists z)(xRz \wedge (\exists w)(zSw \wedge wTy))$$

We can pull the quantifiers to the front because[1] '$(\exists u)(A \wedge \phi(u))$' is the same as '$A \wedge (\exists u)\phi(u)$' getting

$$(\exists z)((\exists w)(xRz \wedge (zSw \wedge wTy)))$$

and

$$(\exists z)(\exists w)(xRz \wedge (zSw \wedge wTy))$$

and we can certainly permute the quantifiers getting

$$(\exists w)(\exists z)(xRz \wedge (zSw \wedge wTy))$$

we can permute the brackets in the matrix of the formula because '$\wedge$' is associative getting

$$(\exists w)(\exists z)((xRz \wedge zSw) \wedge wTy)$$

import the existential quantifier again getting

$$(\exists w)((\exists z)(xRz \wedge zSw) \wedge wTy)$$

and reverse the first few steps by using the definition of $\circ$ to get

$$(\exists w)(x(R \circ S)w \wedge wTy)$$

and

$$x(R \circ S) \circ Ty$$

as desired.

Question ??.??

The lexicographic order on $\mathbb{N}^2$ is wellfounded, so we can do wellfounded induction on it. This means that if we can prove that, if every ordered pair below p has some property ϕ then the pair p has property ϕ as well, then every ordered pair in $\mathbb{N}^2$ has that property.

Now let $\phi(\langle x, y\rangle)$ say that if the bag is started with x black balls and y white balls in it the process will eventually halt with only one ball in the bag. Suppose $\phi(\langle x', y'\rangle)$ holds for every $\langle x', y'\rangle$ below $\langle x, y\rangle$ in the lexicographic product $\mathbb{N}^2$. We want to be sure that if the bag is started with x black balls and y white balls in it the process will eventually halt with only one ball in the bag. The first thing that happens is that we pick two balls out of the bag and the result is that at the next stage we have either $x - 2$ black balls and an unknown number of white balls, or we have x black balls and $y - 1$ white balls. But both these situations are described by ordered pairs below $\langle x, y\rangle$ in the lexicographic product $\mathbb{N}^2$, so by induction hypothesis we infer that if the bag is started with x black balls and y white balls in it the process will eventually halt with only one ball in the bag, as desired.

[1] At least as long as 'u' is not free in A.

Question ??.??

Everybody loves my baby, so in particular my baby loves my baby. **My baby loves nobody but me.** That is to say, if x is loved by my baby, then $x =$ me. So my baby = me.

Question ??.??

The answer is the relation that holds between k and $k+1$ for $0 \leq k < n$ and between n and 0.

Question ??.??

Paula Buttery's answer to one of the fiddly ones.

```
- fun f g b a = g a b;
val f = fn : ('a -> 'b -> 'c) -> 'b -> 'a -> 'c
- fun ff g = let fun fa a = let val (b,c) = g a in b end;
= fun fe a = let val (b,c) = g a in c end;
= in (fa, fe) end;
val ff = fn : ('a -> 'b * 'c) -> ('a -> 'b) * ('a -> 'c)
-
```

Question ??.??

Show that the largest and smallest elements of a totally ordered set with n elements can be found with $\lceil 3n/2 \rceil - 1$ comparisons if n is odd, and $3n/2 - 2$ comparisons if n is even.

First check this for a few small values. If $n = 2$ we need 1, for $n = 3$ we need 3, for $n = 4$ we need 4.

The induction step requires us to show that adding two more elements to a set requires us to perform no more than three extra comparisons.

So suppose we have a set X with n members, and we have found the top and bottom elements in $3n/2 - 1$ comparisons. Call them t and b. Let the two new elements be x and y. With one comparison we can find out which is bigger. Without loss of generality suppose it is x. Compare x with t to find the biggest element of $X \cup \{x, y\}$, and compare y with b to find the smallest. This has used three extra comparisons.

Question ??.??

The exponent on the LHS is $x^{x^{x^{x^{x^{\cdots}}}}}$ which is 2, so $x^2 = 2$ and $x = \sqrt{2}$. That was easy. The problem with this is that the second equation gives $x^4 = 4$ and thence $x = \sqrt{2}$ again. They can't both be right!

Of course the answer is that the reasoning that led us to conclude that $x = \sqrt{2}$ in the first place doesn't prove that that is the answer. All we have done is show that **if** there is a solution it must be $\sqrt{2}$. We haven't shown that there **is** a solution. In fact it is a simple matter to show by induction that the approximants to the LHS, which we generate as follows

$$a_0 =: \sqrt{2}; \quad a_{n+1} =: \sqrt{2}^{a_n}$$

... are all less that 2. So the sequence has a limit which is ≤ 2.

Let's see what we can do that is more general.

Let $F(x) =_{df} x^{x^{x^{x^{x^{\cdots}}}}}$

We have $x^{F(x)} = F(x)$. The inverse to this function is the function $\lambda x.x^{1/x}$. This is much easier to understand. For example we can differentiate it. It is the same as $e^{(log\ x)/x}$ whose differential is of course $e^{(log\ x)/x} \cdot (1/x^2 - (log\ x)/x^2)$. This is zero when $x = e$, and this is clearly a maximum. The fact that the differential is zero there of course means that F reaches a maximum at $e^{1/e}$ and that $F'(e^{1/e})$ is infinite. This gives us the amusing but (as far as i know) useless fact that

$$(e^{1/e})^{(e^{1/e})^{(e^{1/e})^{(e^{1/e})^{(e^{1/e})^{\cdots}}}}} = e$$

(Check this: if the LHS is to evaluate to x we must have $(e^{(1/e)})^x = x$ and e is certainly a solution to this equation.)

We can get a power series expansion of F for values of x not much bigger than 1. Let Σ be the power series for $F(1 + x)$. Then we have

$$(1 + x)^{\Sigma} = \Sigma$$

and we can use the binomial theorem to expand the left hand side. This gives us a sequence of equations expressing later coefficients of Σ in terms of earlier coefficients in a wellfounded way. I haven't worked out the general formula for a_n the coefficient of x^n in $F(1 + x)$ tho' in principle it could be done. ($a_0 = 1$ for a start!)

Question ??.??

Let $m = |F|$ and $p = |\bigcup F|$. Let $C = \{\langle x, A\rangle : x \in A \in F\}$.

Given $x \in \bigcup F$, pick $B \in F$ with $x \in B$. Let $Y_x = \{A \in F : x \in A\}$ and $N_x = \{A \in F : x \notin A\}$. The map $\lambda A.(A \Delta B)$ permutes F and swaps Y_x and N_x. Hence $|Y_x| = |N_x| = m/2$.

So $|C| = (1/2)mp$, as each x is in exactly $m/2$ A's. But each A contains $\leq k$ things, and one A contains none at all, so $|C| \leq (m-1)k$ whence $p \leq \frac{m-1}{m} \cdot 2k < k$.

Question ??

The problem is to find a cute way of getting lots of variables, literals, call them what you will. One solution is to cast `int`s as literals. That way one can represent a state as a list of `bool`s, and one has a nice recursion over `int`s for the base (literal) case of the declaration of the `eval` function.

Various strategies are adopted in the following code submitted by my supervisees over several years. I don't know who did the second answer.

```
(*  This code was cooperatively produced by Mike Bond and Joseph Lord.  It
produces truth results for all possible inputs into a boolean expression.
To run use the function tttable (x) where x is a boolean expression using
the characters a-z excluding o as the variables and symbols ! (NOT), &
(AND) and + (OR).  Precedence is set in that order.  & and + are infixes
and ! precedes the complemented variable.  *)

infix 7 !;
infix 6 &;
infix 5 +;

nonfix !;


datatype BOOL  = VAR of string
       | NOT of BOOL
       | AND of BOOL*BOOL
       | OR of BOOL*BOOL;

fun (a & b) = ( AND( a , b ) );
fun (a + b) = ( OR( a , b ) );
fun ! x = NOT ( x );

fun lvars( VAR(x) ) = [x] |
    lvars( AND(x,y) ) = lvars(x) @ lvars(y) |
    lvars( OR(x,y) ) = lvars(x) @ lvars(y) |
    lvars( NOT(x) ) = lvars(x);

fun ispresent(x,[]) = false |
    ispresent(x,t::ts) = if x=t then true
     else ispresent(x,ts);

fun member(bh,[]) = false |
    member(bh,mainh::maint) = if bh=mainh then true
      else member(bh,maint);
```

```
fun fr(main,[]) = main |
    fr(main,bh::bt) = if member(bh,main) then fr(main,bt)
else fr(bh::main,bt);

fun evl(expression,alist) =
    let fun evaluate( AND(x,y) ) = evaluate(x) andalso evaluate(y) |
    evaluate( OR(x,y) ) = evaluate(x) orelse evaluate(y) |
    evaluate( NOT(x) ) = not (evaluate(x)) |
    evaluate( VAR(x) ) = ispresent(x,alist)
     in
evaluate(expression)
     end;

fun tline(expression,alist) =
     if evl(expression,alist) then (alist,"TRUE")
      else (alist,"FALSE");


fun powset ([],base) = [base]
  | powset (x::xs,base) = powset (xs,base) @ powset(xs,x::base);


fun ttable(expression,[]) = []
  | ttable(expression,p::ps)=[tline(expression,p)]::ttable(expression,ps);

fun tttable(expression) =
ttable(expression,powset( fr([],lvars(expression)) , [] ));


val a=VAR("a");
val b=VAR("b");
val c=VAR("c");
val d=VAR("d");
val e=VAR("e");
val f=VAR("f");
val g=VAR("g");
val h=VAR("h");
val i=VAR("i");
val j=VAR("j");
val k=VAR("k");
val l=VAR("l");
val m=VAR("m");
val n=VAR("n");
val p=VAR("p");
```

```
val q=VAR("q");
val r=VAR("r");
val s=VAR("s");
val t=VAR("t");
val u=VAR("u");
val v=VAR("v");
val w=VAR("w");
val x=VAR("x");
val y=VAR("y");
val z=VAR("z");
```

```
(* We'll need things for dealing with lazy-lists so here's some.        *)
datatype lazylist = Tip
                  | Cell of (int->bool) * (unit->lazylist);

fun head (Cell (x,y)) = x;
fun tail (Cell (x,y)) = y ();
```

```
(* Basic list functions.                                                *)
(*                                                                      *)
(* Membership test                                                      *)
fun mem [] x = false
  | mem (y::ys) x = if x=y then true else mem ys x;
```

```
(* Merge two lists set-union style.  Only x add to ys if it isn't       *)
(* already in ys.                                                       *)
fun merge ([], ys) = ys
  | merge (x::xs, ys) = if mem ys x then merge (xs, ys)
                                    else x::merge (xs, ys);
```

```
(* Simply prefixes x to each list y in the list of lists ys.            *)
fun prefix (x, []) = []
  | prefix (x, y::ys) = (x::y) :: prefix (x, ys);
```

```
(* Return a list of all the possible subsets of the list given as an    *)
(* argument.                                                            *)
fun subsets [] = [[]]
  | subsets (x::xs) = prefix (x, subsets xs) @ subsets xs;
```

```
(* Here's the data-type we're going to use for our formulae.           *)
datatype fmla   = Lit of int
                | Not of fmla
                | And of fmla * fmla
                | Or of fmla * fmla
                | Implies of fmla * fmla
                | Iff of fmla * fmla;


(* The SAT function.                                                   *)
(*                                                                     *)
(* Evaluates the function given that the literals take the states      *)
(* returned by the function given as the second parameter.  States     *)
(* should be a function with type int->bool.                           *)
fun SAT (Lit (l), states) = states l
  | SAT (Not (f), states) = not (SAT (f, states))
  | SAT (And (f, g), states) = SAT (f, states) andalso SAT (g, states)
  | SAT (Or (f, g), states) = SAT (f, states) orelse SAT (g, states)
  | SAT (Implies (f, g), states) = (not (SAT (f, states))) orelse
                                     SAT (g, states)
  | SAT (Iff (f, g), states) = SAT (Implies (f, g), states) andalso
                               SAT (Implies (g, f), states);
  (*SAT (Iff (f, g), states) = (SAT (f,states) = SAT(g,states))   *)

(* All the below is for the valid function.                            *)

(* First we want a list of the literals in the function.  We will need *)
(* this in order to work out all the possible true/false combinations. *)
fun getlits (Lit l) = [l]
  | getlits (Not l) = getlits l
  | getlits (And (l, m)) = merge (getlits l, getlits m)
  | getlits (Or (l, m)) = merge (getlits l, getlits m)
  | getlits (Implies (l, m)) = merge (getlits l, getlits m)
  | getlits (Iff (l, m)) = merge (getlits l, getlits m);


(* Return a lazy list the head of which is a function which returns    *)
(* whether its argument is a member of that list (in the lazy list of  *)
(* lists which it is.                                                  *)
fun allstates [] = Tip
  | allstates (x::xs) = Cell (mem x, fn () => allstates xs);


(* Now the crunchy bit.  Valid takes a single argument - the function. *)
(* It extracts all the literals from it and then finds all the subsets *)
(* of the list.  We will use this to find all the true/false           *)
(* combinations: Each subset will count as a separate case.  If the    *)
```

```
(* literal in question is a member of the subset then it is true        *)
(* in this case.  If it is not a member then it is false.   We test     *)
(* phi with each case returning false as soon as we get a false, but    *)
(* only returning true when we've tested all the cases and all were     *)
(* true.                                                                *)
fun valid phi =
    let fun validbit (Tip) = true
          | validbit (Cell (f, g)) =
            if SAT (phi, f) then validbit (g ())
                            else false
    in  validbit (allstates (subsets (getlits phi)))
    end;


(* Test formulae.                                                       *)
(*      P /\ Q -> Q /\ P                                                *)
val ok = Implies (And (Lit 1, Lit 2), And (Lit 2, Lit 1));

(*      P \/ Q -> P /\ Q                                                *)
val bad = Implies (Or (Lit 1, Lit 2), And (Lit 1, Lit 2));

(*      ((P -> Q) -> P) -> P                                            *)
val peirceslaw = Implies (Implies (Implies (Lit 1, Lit 2), Lit 1), Lit 1);

----------------------------------->8-------------------------------------
```

Question ?? part b

```
Below is a version of SAT which has been extended to cope with the
unary box operator, and a revised data-type which includes the
operator.

----------------------------------->8-------------------------------------
(* Here's the data-type we're going to use for our formulae.           *)
datatype fmla   = Lit of int
                | Not of fmla
                | Box of fmla
                | And of fmla * fmla
                | Or of fmla * fmla
                | Implies of fmla * fmla
                | Iff of fmla * fmla;
(* The SAT function.                                                   *)
(*                                                                     *)
(* Evaluates the function given that the literals take the states      *)
(* returned by the function given as the second parameter.  States     *)
```

```
(* should be a function with type int->bool.                          *)
(* In this version the unary box operator has been implemented.       *)
fun SAT (Lit (l), states) = states l
  | SAT (Not (f), states) = not (SAT (f, states))
  | SAT (And (f, g), states) = SAT (f, states) andalso SAT (g, states)
  | SAT (Or (f, g), states) = SAT (f, states) orelse SAT (g, states)
  | SAT (Implies (f, g), states) = (not (SAT (f, states))) orelse
                                        SAT (g, states)
  | SAT (Iff (f, g), states) = SAT (Implies (f, g), states) andalso
                               SAT (Implies (g, f), states)
  | SAT (Box (f), states) =
    let fun validbit (Tip) = true
          | validbit (Cell (g, h)) =
            if SAT (f, g) then validbit (h ())
                          else false
    in  validbit (allstates (subsets (getlits f)))
    end;
----------------------------------->8-------------------------------------
```

Question ?? part c

Below is a version of SAT which has been extended to cope with the R relation on the set of valid states.

```
----------------------------------->8-------------------------------------
(* The SAT function. (part c)                                          *)
(*                                                                     *)
(* Evaluates the function given that the literals take the states      *)
(* returned by the function given as the second parameter.  States     *)
(* should be a function with type int->bool.                           *)
(* In this version the unary box operator has been implemented in the  *)
(* style of part c.  The idea is that the third parameter R is the     *)
(* relation between states.  Given a state R it will return a lazy list *)
(* of states (in the manner of allstates).                             *)
fun SAT (Lit (l), states, R) = states l
  | SAT (Not (f), states, R) = not (SAT (f, states, R))
  | SAT (And (f, g), states, R) = SAT (f, states, R) andalso SAT (g, states, R)
  | SAT (Or (f, g), states, R) = SAT (f, states, R) orelse SAT (g, states, R)
  | SAT (Implies (f, g), states, R) = (not (SAT (f, states, R))) orelse
                                        SAT (g, states, R)
  | SAT (Iff (f, g), states, R) = SAT (Implies (f, g), states, R) andalso
                               SAT (Implies (g, f), states, R)
  | SAT (Box (f), states, R) =
    let fun validbit (Tip) = true
          | validbit (Cell (g, h)) =
            if SAT (f, g, R) then validbit (h ())
```

```
                          else false
    in  validbit (R states)
    end;
```

And an answer from David Burleigh

```
(* A formula datatype and its implementation.  By David Burleigh. *)
datatype formula = lit of string
   | fand of formula * formula
   | for of formula * formula
   | fimp of formula * formula
   | fnot of formula
   | feq of  formula * formula;

fun eval (fand(x,y) :formula) statefun =
 statefun x andalso statefun y
  | eval (for(x,y) :formula) statefun =
 statefun x orelse statefun y
  | eval (fimp(x,y) :formula) statefun =
 not(statefun x) orelse statefun y
  | eval (feq(x,y) :formula) statefun =
 (statefun x) = (statefun y)
  | eval (fnot(x) :formula) statefun =
 not(statefun x)
  | eval (a :formula) statefun = statefun a;

fun state (s::ss :(string * bool) list) x =
 let val states = s::ss
 in
  let fun staten [] x = eval x (staten states)
          | staten ((a,b)::ss) x =
    if x = lit a then b
    else staten (ss) x
  in
   staten states x
  end
 end;

val statelist = [[("p",false),("q",false)],[("p",true),("q",false)],
   [("p",false),("q",true)],[("p",true),("q",true)]];

val statefuns = map state statelist;

fun truthtable fmula = map (eval fmula) statefuns;

val nd = fand(lit "p",lit "q");
```

```
val or = for(lit "p", lit "q");=09

(* A formula datatype and its implementation.  By
David Burleigh. *)
datatype formula = lit of = string | fand of formula * formula | for of formula * formula| fimp of
formula | feq of formula * formula
fun eval (fand(x,y) :formula) statefun = statefun
x andalso statefun y | eval (for(x,y) :formula) statefun = statefun x orelse statefun y | eval (fi
not(statefun x) orelse statefun y | eval (feq(x,y) = :formula)
statefun = (statefun x) = (statefun y) | eval =
(fnot(x) :formula)
statefun = not(statefun x) | eval (a :formula) statefun =
= statefun a

fun state (s::ss :(string * bool) list) x =
   let val states = s::ss;

let fun staten [] x = eval x = (staten states) | =
staten((a,b)::ss) x if x = lit a then = else staten (ss) x; staten states =
x;

val statelist = [[(&quot;p&quot;,false),(&quot;q&quot;,false)],[(&quot;p&quot;,true),(&qu=
ot;q&quot;,false)],<BR>  
[(&quot;p&quot;,false),(&quot;q&quot;,true)],[(&quot;p&quot;,true),(&quot=
;q&quot;,true)]];</FONT> </DIV>
<DIV><FONT color=#000000></FONT> </DIV>
<DIV><FONT color=#000000>val statefuns = map state =
statelist;</FONT></DIV>
<DIV><FONT color=#000000></FONT> </DIV>
<DIV><FONT color=#000000>fun truthtable fmula = map (eval fmula)
statefuns;</FONT></DIV>
<DIV><FONT color=#000000></FONT> </DIV>
<DIV><FONT color=#000000>val nd = fand(lit &quot;p&quot;,lit
&quot;q&quot;);<BR>val or = for(lit &quot;p&quot; lit
```

Some funny stuff here ...

0.0.1 ????

$$\dfrac{\dfrac{\dfrac{\dfrac{X \quad X \to Y}{Y} \quad \dfrac{X \quad X \to (Y \to Z)}{Y \to Z}}{Z} \quad \dfrac{\dfrac{X \quad X \to Y}{Y} \quad \dfrac{X \quad X \to (Y \to (Z \to W))}{Y \to Z \to W}}{Z \to W}}{W}}{X \to W}}{(X \to Y) \to (X \to W)}}{(X \to (Y \to Z)) \to ((X \to Y) \to (X \to W))}}{X \to (Y \to (Z \to W)) \to (X \to (Y \to Z)) \to ((X \to Y) \to (X \to W))}$$

The weighing problem

Weigh four against four. If they don't balance, weigh two (potentially) heavy and two (potentially) light against two normals and a heavy and a light.

1993:10:6

$f(n, r) = f(n-1, r-1) + rf(n-1, r)$

Equivalence relations give rise to partitions

The way to explain how equivalence relations give rise to partitions is to suppose that you start with a set X with an equivalence relation on it. You pick an element of X and stick it in a bin. Thereafter you pick up an object and stick it in the bin (if any) containing things to which it is related. If there aren't any start a new bin. Compscis understand that sort of thing.

****

First check that you know what is meant by the word 'partition'. This matters, since people use it in different ways: number theorists talk of partitions of members of $\mathbb{N}$ (and that stuff is a lot of fun too). However here we are concerned with partitions of *sets*. A partition of a set X is a collection of subsets of X which are disjoint (make sure you know what that means) and between them contain every member of X.

Suppose R is an equivalence relation on a set X. We must find a partition of X that corresponds to R in a natural way. The partition we want is the partition of X into R-equivalence classes.

For $x \in X$, the R-equivalence class of x, (written "$[x]_R$") is $\{y \in X : yRx\}$.

We want to know that the collection of all equivalence classes is a partition of X, that is to say, every $x \in X$ belongs to an equivalence class, and any two equivalence classes are either identical or disjoint. The first is easy: every x belongs to $[x]_R$ (even if it is its sole member). The second needs a bit of work.

Let us suppose there are two equivalence classes which overlap but are distinct. Then there are x and y (which are not R-equivalent) and there is a z such that $z \in [x]_R \cap [y]_R$. But then xRz and yRz and so on, so by transitivity and symmetry xRy, so x and y belong to the same equivalence class, so $[x]_R = [y]_R$ as desired.

The difficulty with this problem is to work out exactly what it is you have to prove, since once you understand it it is all so obvious. One thing worth making a big fuss about is that the notation '$[x]_R$' for equivalence classes can be a wee bit misleading, because *any* member of an equivalence class can be used to grab it in this fashion: the various x's in it are all, well, equivalent!

1992:11:8

First of all, the word 'cover' in here is a red herring, and we can just as well think of **partitions** of P into antichains. Why? Well, if we have a cover of P by antichains, we can wellorder the cover, and delete from any later member of the cover any element of P that has already appeared in an earlier member of the cover. This may even remove elements of the cover altogether. At all events, we can be sure that there will be at least one cover of minimal size that is a partition, so wlog we can assume that we are dealing with a partition.

"Length of a longest chain in P" is a bit naughty, because it could mean either "maximal chain" or "chain of maximal length". In the partial order whose Hasse diagram is

there are two maximal chains, one of length three (has three elements) and the other of length four(has four elements). However whoever it was that dreamed up this question probably meant "*chain of maximal length*" rather than "*maximal chain*". Let us procede on that assumption.

So what we are trying to prove it that for any (finite!) partial ordering $\langle P, \leq_P \rangle$, the least $n \in \mathbb{N}$ such that there is a covering of P by antichains (call it n_0) is also the length of a chain of maximal length in P (call that n_1).

If Π is a partition of P into antichains, and C is a chain in P, clearly every element of Π must meet C, and no two elements of Π can contain the same element of C, so $n_0 \leq n_1$.

There is a **canonical partition** of P into antichains, as follows. The *rank* $\rho(x)$ of an element x of P is defined by recursion on $\leq_P$ by $\rho(x) =$ sup of

$\{(\rho(y)) + 1 : y <_P x\}$. We then partition P according to the ranks of its elements. Think of an element of P of maximal rank, n, say. It must have something below it of rank $n-1$, and so on down. That way we show there is a chain in P of length n. Clearly no chain in P can be longer, so this is of maximal length, namely n_1. Fix one such chain and call it C.

Now let Π be an arbitrary partition of P into antichains. Each antichain must contain at most one element of C: if it contained more it wouldn't be an antichain! So there must be at least as many antichains in Π as there are elements of C, namely n_1. So $n_1 \leq n_0$.

1992:10:8

Part the first: a message from AGT

There are $\binom{r+k-1}{k-1}$ ways to select r fruits from a greengrocer who sells k kinds, so thats it. For $k_i > 0$, put $l_i = k_{i-1}$ and note the $l - i$ sum to $r - k$.

For lower bounds on codes, just pick greedily. Each codeword picked rules out $b =$ size of $B(x, 2e+1)$ where e is number of errors you hope to correct. So you can get a gode of size N/b, where N is total number of words.

Second part:

$n(r, A)$ is one less than the smallest number we cannot express as a sum of r things in whose denominations belong to A.

The tricky part is to work out precisely *how* to prove this triviality. The first thing that leaps to mind is that we should attempt to prove it by induction. The trouble with proving things by induction is that we have to do induction on some integer vbls and (perhaps) UG on others. Therefore, **the first thing to do is to put back all the implicit quantifiers that have been left out**. This gives us

$$(\forall k \in \mathbb{N})(\forall r)(\forall A)(|A| = k \rightarrow n(r, A) \leq \binom{k+r}{k})$$

All the initial quantifiers are universal so it doesn't matter which order they are written in. So what do we do? We can do UG on r by fixing r and then proving the theorem by induction on k, or we can prove by induction on k that for all r ... or we can prove it by induction on r.

Commentary on Peter Robinson's proof of Inclusion-Exclusion

This proof is elliptical but perfectly correct. It's elliptical beco's he has a lot of material to get thru' and not much time, and alse beco's he wants you to put in some work. Like me, he is a director of studies and probably believes like me that students should be introduced—very early on—to the idea that Life is Hard. I think its present form is the result of many years of people honing the proof to get it over and done with as quickly as possible: surgeons in the days before anæsthetics had the same problem. I append some comments my supervisees have found helpful.

I'm assuming you understand the result and have some idea of why it should be true. Let's read it. (Look for "The principle states that ..."). Vertical bars either side of a notation for a set form a notation for the number of elements of that set. This is standard. The thing on the left hand side of the equation is the number of things that belong to the union of the A_i. The family of As is **indexed**: each A has a pointer pointing at it from an **index set**—which in this case is called 'S'.

So in English the equation reads something like

> "The number of things in the union of the As is minus the sum—over nonempty subsets T of S—of minus-one-to-the-power-of-the-number-of-things-in-T times the number of things in the intersection of all those As whose subscripts are in T."

Or—plainer still—for each nonempty $T \subseteq S$, take the intersection of all the As whose subscripts are in T (those As pointed to by elements of T) take its cardinality and take it negative or positive depending on whether T has an odd or even number of elements. Add them all together, for all such T, and make the answer positive.

The first thing to take note of is a bit of **overloading**. Primarily we write 'A_s' to denote one of the As, and the subscript is a member of the index set. However we are now going to write 'A_T', where T is a *subset* of S not a member, and this expression will denote the intersection of all the As whose subscripts are in T. It's easy to detect which of these two usages are in play at any one time, beco's the indices themselves are lower-case Roman letters and the *sets* of indices are upper-case Roman letters. This is a common use of the difference between upper and lower case Roman letters. Notice that $A_\emptyset$ is the whole universe of discourse—Ω. This probably bears thinking about, so we pause for a brief digression on the empty disjunction and the empty conjunction. Any disjunction D can always be tho'rt of as $D \vee$ `false`. so the empty disjunction is just `false`. Analogously the empty conjunction is true: any conjunction C can be tho'rt of as $C \wedge$ `true`.

Now **indicator functions**. You will need to know about these for a variety of reasons. For example they crop up in 1b computation theory where they

are called **characteristic** functions. Each subset B of Ω has its own indicator function, written 'I_B'. This is the function which (on being given $x \in \Omega$) returns **true** or **false** depending on whether or not $x \in B$. Except that it returns `int`s instead of `bool`s. This piece of **casting** is so that we can use *arithmetic* operations on the truth-values. It's universal practice in machine code Hacky but clever. This ensures that

- $I_{B\cap C}(x) = I_A(x) \cdot I_B(x)$ (PR alludes to this on the line beginning "indicator functions", tho' he leaves out the 'x') and
- $I_{\overline{B}}(x) = 1 - I_B(x)$.

Notice that PR uses a convention (standard, and not explained here) that $\overline{A_s}$ is the complement of A_s. ("Overlines mean complementation").

For the third displayed line (the one beginning with the big Σ) consider what happens when you muliply out things like $(1-a)(1-b)(1-c)(1-d)$: you get $1-$ lots of things like $-abc$ and $+bd$ which are positive or negative depending on the number of factors. "But shouldn't it start with a '$1-$' before the big Σ?" i hear you cry. It should indeed, but that $1-$ is in fact included beco's one of the Ts you sum over is the empty set! Very cunning.

The next nonindented line begins "Now sum over ...". What's going on here? $I_{\overline{A_1}\cap\overline{A_2}\cap\ldots\overline{A_n}}(x)$ looks nasty so just ignore the subscript for the moment. The sum of $I_B(x)$ over all $x \in \Omega$ is simply the number of things in B, or—using the vertical bar notation—$|B|$. This gives the left hand side of the equation on that line.

1 Answers to some of Peter Robinson's exercises

Here is why Euclid's algorithm works and what it means. Anything that divides x and y divides $x - y$, so if i want to find HCF(x, y) i should keep repeating the step of replacing the larger of x and y by $|(x - y)|$. The HCF of the pair of numbers in hand is a loop invariant, and when the process stops with both elements the same we have found the HCF.

If the bigger number is *much* bigger than the smaller one then we could end up subtracting the smaller one many times, and we can save ourselves time by conflating lots of these subtractions together by dividing the bigger number by the smaller and keeping only the remainder.

Remember there is no sensible notion of **betweenness** for integers mod n.

1.1 Exercises pp. 5-6

4 is the only one i hadn't seen before.

RTP: $5|2^{3n+1} + 3^{n+1}$

Check that it is true for $n = 0$. Suppose true for n, and aspire to deduce it for $n+1$. For this it will be sufficient to show that the difference $2^{3n+4} + 3^{n+2} - (2^{3n+1} + 3^{n+1})$

is divisible by 5. Collect like terms to get

$2^{3n+4} - 2^{3n+1} + 3^{n+2} - 3^{n+1})$

$2^{3n+4} - 2^{3n+1} + 3^{n+2} - 3^{n+1})$

$7 \cdot 2^{3n+1} + 2 \cdot 3^{n+1}$

$5 \cdot 2^{3n+1} + 2 \cdot (2^{3n+1} + 3^{n+1})$

Both things being added up are multiples of 5.

question 8: see

1.2 Exercises pp. 12-13

1.
   - No: try $(3, 6)(2, 5)$
   - No: try $(2, 4)(2, 5)$
   - Yes. Think: disjoint multisets!

2. Euclid's algorithm gives us

   $57 = 44 + 13$

   $44 = 3.13 + 5$

   $13 = 2.5 + 3$

   $5 = 3 + 2$

   $3 = 2 + 1$

$2 = 1 + 1$

We now work downwards to get, one after the other, the numbers on the LHS of these equations expressed as differences of multiples of 44 and 57.

$13 = 57 - 44$

$5 = 44 - 3 \cdot 13 = 44 - 3(57 - 44)$ $= 4 \cdot 44 - 3 \cdot 57$

$3 = 13 - 2 \cdot 5 = (57 - 44) - 2 \cdot (4.44 - 3.57)$ $= 7 \cdot 44 - 10 \cdot 57$

$2 = 5 - 3 = 4 \cdot 44 - 3 \cdot 57 - 7 \cdot 57 + 9 \cdot 44$ $= 13 \cdot 44 - 10 \cdot 57$

$1 = 3 - 2 = 7 \cdot 57 - 9 \cdot 44 - 13 \cdot 44 + 10 \cdot 57$ $= 17 \cdot 57 - 22 \cdot 44$

This gives the solution $x = 17; y = -22$. Then

$57 \cdot 17 + 44 \cdot (-22) = 1$

$57 \cdot x \quad + 44 \cdot y \quad = 1$ ... and subtract to get

$57 \cdot (17 - x) = 44 \cdot (22 + y)$. This gives

$(22 + y)/57 = (17 - x)/44$. Call this quantity k. Then

$y = 57k - 22$ and $x = 17 - 44k$ as desired.

Why is k an integer? Beco's 44 and 57 are coprime. 44 divides the LHS and it doesn't divide 57 so it must divide $22 + y$.

3. It has no solution in integers beco's $(1992, 1752) = 24$. The rest of the question is a rerun of the last one.

4. Ternary Euclid. Look at the first two variables—in this case $56x + 63y$. Do a Euclid on them to discover that the HCF is 7. That means that all you are ever going to get out of $56x + 63y$ is a multiple of 7, so replace $56x + 63y$ by $7a$ and solve $7a + 72z = 1$.

5.

6. Use the hint. Any prime factor of N must a a prime bigger than p_n. Any prime must be congruent to 1 or -1 mod 4. N itself is congruent to -1 mod 4 so at least one of its factors is a prime congruent to -1 mod 4, which is to say a prime of the form $4k + 3$, and it is bigger than p_n, n was arbitrary, so there are arbitrarily large primes of this form.

7.

```
fun factor(n,2) = if n<3   then [n]
                  else if n mod 2 = 0 then 2::factor(n div 2,2)
                  else factor(n,3)
   |factor(n,a) = if n<a*a then [n]
                  else if n mod a = 0 then a::factor(n div a,a)
                  else factor(n,a+2);

fun out(0) = ""
```

```
   |out(x) = out(x div 10)^chr(48+(x mod 10));

fun foutput(x::[]) = out(x)
   |foutput(x::xs) = out(x) ^ "*" ^ foutput(xs);

fun findfactor(n)  = foutput(factor(abs(n),2));

(* test data: fermat prime 2^(2^5)+1 *)
findfactor(floor(exp(exp(5.0*ln(2.0))*ln(2.0)))+1);
```

8. P. Satangput's ML function that implements Euclid

```
fun dogcd(m,0,a,b,c,d) = [a,b,m]
  | dogcd(m,n,a,b,c,d) = dogcd(n ,m mod n,c,d,a-c*(m div n), b-d*(m div n));

fun gcd(m,n) = dogcd(m,n,1,0,0,1);
```

9. (a) Show that $f_{n+k} = f_k \cdot f_{n+1} + f_{k-1} \cdot f_n$.
   Let's take the hint. (Bear in mind that in any problem with lots of integer variables there may be only one variable you can do the induction on!) We shall prove by induction on k that for all n, $f_{n+k} = f_k \cdot f_{n-1} + f_{k-1} \cdot f_n$. Best to check first that this is true for $k = 1$. If we take $f_0 = f_1 = 1$ this becomes $f_{n+1} = f_{n-1} + f_n$ which is of course the recursive declaration of the Fibonacci numbers.
   Suppose true for k. Then $f_{n+k+1} = f_{n+k} + f_{n+k-1}$. Rearrange this last term on the RHS to $f_{(n-1)+k}$ (the induction hypothesis is that the identity holds for *all* n!) to get
   $$\begin{aligned} f_{n+k+1} &= f_{n+k} + f_{(n-1)+k} \\ &= f_k \cdot f_n + f_{k-1} \cdot f_n + f_k \cdot f_n + f_{k-1} \cdot f_{n-1} \\ &= f_k \cdot (f_{n-1} + f_n) + f_{k-1} \cdot (f_n + f_{n-1}) \\ &= f_k \cdot f_{n+2} + f_{k-1} \cdot f_{n+1} \end{aligned}$$
   as desired.

   (b) First we prove by induction on n that $(\forall l)(f_n | f_{n \cdot l})$. It's true for $l = 0$. We want $f_n | (f_{n \cdot l+l})$. Using part one to expand the RHS we reduce the problem to showing $f_n | (f_n \cdot f_{n \cdot l+1} + f_{n-1} \cdot f_{n.l})$. Obviously $f_n | f_n \cdot f_{n \cdot l+1}$ and $f_n | f_{n-1} \cdot f_{n.l}$ by induction hypothesis.

   (c) ("Deduce also that $(f_m, f_n) = (f_{(m-n)}, f_n)$"), substitute $m - n$ for n and n for k in part one to get
   $$A : f_m = f_n \cdot f_{(m-n)+1} + f_{(n-1)} \cdot f_{(m-n)}.$$
   Then for any x, if $x | f_n$ and $x | f_{(m-n)}$ then x divides $f_n \cdot f_{(m-n)+1} + f_{(n-1)} \cdot f_{(m-n)}$ and therefore $x | f_m$. One such x is $(f_{(m-n)}, f_n)$ whence $(f_{(m-n)}, f_n) | f_m$. $(f_{(m-n)}, f_n) | f_n$ holds anyway giving $(f_{(m-n)}, f_n) | (f_m, f_n)$.

For the other direction suppose $x|f_m$ and $x|f_n$. We can rearrange the equation A to get

$$f_m - f_n \cdot f_{(m-n)+1} = f_{(n-1)} \cdot f_{(m-n)}$$

Then x divides the LHS and so $x|f_{(n-1)} \cdot f_{(m-n)}$. This isn't quite what we wanted: we needed $x|f_{(m-n)}$. But we deduce this once we know that x does not divide $f_{(n-1)}$. That will follow once we can show that f_n and $f_{(n+1)}$ are always coprime. That is proved by an induction so easy i sha'n't write it out.

So we have proved $x|f_m$ and $x|f_n$ implies that $x|f_{(m-n)}$. In particular $(f_m, f_n)|f_{(m-n)}$. In any case we have $(f_m, f_n)|f_n$ giving $(f_m, f_n)|(f_{(m-n)}, f_n)$. We have already proved $(f_{(m-n)}, f_n)|(f_m, f_n)$ so we infer $(f_m, f_n) = (f_{(m-n)}, f_n)$.

(d) Show that $f_m \cdot f_n | f_{mn}$ iff $(m, n) = 1$.

1.3 Exercises pp. 19-21

1. Every power of 10 is congruent to 1 mod 9. So if we express a number as a sum of multiples of powers of 10 its residue mod 9 is just the sum of the coefficients of the powers of 10 that we have used.

2. Odd powers of 10 are congruent to -1 mod 11, and even powers are congruent to 1 mod 11.

3.

4. The probability is 1. A number is divisible by 99 iff it is divisible by 9 and by 11. Rearranging the digits makes no difference to divisibility by 9, and reversing the digits makes no difference to divisibility by 11.

5. ISBN numbers

6. - Since (11,40) = 1 we can divide thru' by 11 getting $7x \equiv 1 \pmod{40}$. We then use Euclid, but we can do it by inspection. $6.7 = 42 \equiv 2$, so $12.7 = 84 (\equiv 4)$ so $14.7 \equiv 8 \bmod 40$ and $13.7 \equiv 1 \bmod 40$. So x is 13.
   - By inspection 54 + 30 = 84 which by a happy accident is 12.7 so $y = 7$. It would be a bit of a bugger otherwise since 12 and 54 aren't coprime.
   - Chinese remainder theorem here ...

7. $20! \cdot 21^{20}$ is $21! \cdot 21^{19}$. We want to use Wilson's theorem and we are doing this mod 23 so we will turn $21! \cdot 21^{19} \pmod{23}$ into

   $\frac{22! \cdot 21^{19}}{22} \pmod{23}$

   and since division is OK as long as the base of the modulus is prime. 22 = -1 (mod 23) and 1/(-1) = -1 so this becomes

   $22! \cdot 21^{19} \cdot (-1) \pmod{23}$

   Now we can use Wilson's theorem to turn 22! into -1. The two minus signs cancel and we are left with 21^{19}. This is $(-2)^{19}$ which in turn is -2^{19} which is $-16 \cdot 16 \cdot 16 \cdot 16 \cdot 8$. 16 = -7 so this is $-7 \cdot 7 \cdot 7 \cdot 7 \cdot 8$. 7^2 is 49 which is 3; 9.8 = 72 which is 3. So i make it -3.

8.

9. 2^8 is congruent to -1 mod 257. The monstrous number is an even power of 2^8 and so is congruent to 1 mod 257.

10. Let's try to factorise $n^7 - n$ and see what happens. It becomes

$$n(n-1)(n+1)(n^2+n+1)(n^2-n+1)$$

The first three factors guarantee that the product is a multiple of 6. and as long as n is congruent to 0, 1 or 6 mod 7 they ensure it will be a multiple of 7 as well. That leaves only the cases where n is congruent to 2, 3, 4 or 5 mod 7 and we can check by hand that in the even cases $n^2 + n + 1$ is a multiple of 7 and in the odd cases $n^2 - n + 1$ is.

There is a cuter proof (Thank you, Nicola Whiteoak!). Think about $(n^7 - n)$ (mod 7). Check that for $n = 1, 2, 3, 4, 5, 6$, $n^7 \equiv n$ mod 7. Or we can use Fermat's little theorem.

11. $3901 = 83.46.\phi(m) = 82.46$. Seek multiplicative inverse of 1997 mod 3772. It is .....

12.

13. a^{k+1} is alleged to be the square root of a mod p as long as $p = 4k + 3$ and is a prime. If this is so then $a^{2\cdot(k+1)}$ $(= a^{2k+2}$—let's abbreviate it to 'b') should be congruent to a mod p. We know at least—from Fermat's little theorem—that $a^{4k+3} \equiv a$ (mod p). This gives $a^{4k+4} \equiv a^2$ (mod p) which gives $b^2 \equiv a^2$ ldots not quite working

    If a^{k+1} is a sqrt of a then a^{2k+2} should be a and a^{2k+1} should be 1. Well, a^{4k+2} is 1, by Fermat's little theorem, so a^{2k+1} is at least a sqrt of 1.

$(A \times C) \cup (B \times D) = (A \cup B) \times (C \cup D)$?

Remember, sets are **extensional**: two sets with the same mebers are the same sets. So it will be sufficient to check whether or not these two sets have the same members. We will need to arm ourselves with two functions `fst` and `snd`...

x belongs to the LHS iff
$x \in A \times C \ \vee \ x \in B \times D$
iff $(\texttt{fst}(x) \in A \wedge \texttt{snd}(x) \in C) \vee (\texttt{fst}(x) \in B \wedge \texttt{snd}(x) \in D)$
x belongs to the LHS iff
$\texttt{fst}(x) \in (A \cup B) \wedge \texttt{snd}(x) \in (C \cup D)$
Now we can do some abbreviations:
'$\texttt{fst}(x) \in A$ to 'a'
'$\texttt{fst}(x) \in B$ to 'b'
'$\texttt{snd}(x) \in C$ to 'c'
'$\texttt{snd}(x) \in D$ to 'd'
The two conditions reduce to $(a \wedge c) \vee (b \wedge d)$ and $(a \vee b) \wedge (c \vee d)$ which are clearly distinct.

Question 5 page 15 of the second set

For the first line to imply the second assume the first line, namely that there is $g : A/R \to B/S$ such that $g \circ p = q \circ f$, and assume that a_1 and a_2 satisfy the antecedent of the second line.

So $\langle a_1, a_2 \rangle \in R$. This is the same as saying that $p(a_1) = p(a_2)$. (consider the definition of p). So $g \circ p(a_1) = g \circ p(a_2)$. But $g \circ p = q \circ f$, so substituting $q \circ f$ for $g \circ p$ we get $q \circ f(a_1) = q \circ f(a_2)$ which is to say that $q(f(a_1)) = q(f(a_2))$ which—by definition of q—says that $\langle f(a_1), f(a_2) \rangle \in S$.

For the second line to imply the first line declare g by: $g[a]_R =: [f(a)]_S$. Line 2 tells us that it doesn't matter which element of an equivalence class we consider when trying to determine what g of that equivalence class is, so this definition is legitimate.

1.4 Some relevant ML code

```
Please find attached (and copied below) a better prime number builder (see
notes in code).  I'm intrigued as to exactly how this works and why it is
better.  If you have any time before the supervision I would be grateful if
you could have a look at it.

Andrew Rose

(*-----------------------------------------------------*)

(*  You may recall that I sent you some code last term
    which produced prime numbers.
    I thought of a different method using integer
    streams (ML Tick 6 & 6*).
    I asked both versions (old version included) to
    produce the first 887 prime numbers.
    The old version took 11.5 seconds.
    The new version took  1.5 seconds!
    Unfortunately, the new version seems to be very
    space inefficient.  Trying to generate more primes
    with the new version causes it to crash CML (windows
    closes it down!).
       *)


(*-------------Generic Stream Manipulations------------*)

datatype stream = Item of int * (unit->stream);
fun cons (x,xs) = Item(x,xs);
fun head (Item(i,xf)) = i;
fun tail (Item(i,xf)) = xf();
fun makeints n = cons(n, fn()=> makeints(n+1));
fun nth(s,n) = if n=1 then head(s) else nth(tail(s),n-1);

fun ton(n,s,xs) = if head(s)>n then
   xs
      else
   head(s)::ton(n,tail(s),xs);

fun filter f xs = if f(head(xs)) then
      cons(head(xs),fn()=>(filter(f) (tail(xs))))
```

```
    else
      filter (f) (tail(xs));
(*-------------------------------------------------------*)



(*-------------Prime Number Generation Code------------*)

fun notdiv n x =
 if (x mod n)=0 andalso n<>x then false else true;

fun sqr(x:int) = x*x;

fun makeprimes(n) =
  let fun xmakeprimes(n,acc,prms)=
    if n=acc then
      ton(sqr(nth(prms,n+1))-1,prms,[])
    else
      xmakeprimes(n,acc+1,(filter (notdiv (nth(prms,acc+1))) (prms)))
  in
    xmakeprimes(n,1,makeints 1)
  end;
(*-------------------------------------------------------*)




(*-----------Old Prime Number Generation Code----------*)

fun pIsPrime(n) =
 let
  fun xpIsPrime(n,sf) =
   if sf=n then
    true
   else
    if n mod sf = 0 then
     false
    else
     xpIsPrime(n,sf+1)
    (**)
   (**)
  (**)
 in
  xpIsPrime(n,2)
```

```
 end
;

fun pBuild(n) =
 let
  fun xpBuild(n,sf,acc) =
   if n=0 then
    acc
   else
    if pIsPrime(sf) then
     xpBuild(n-1,sf+1,sf::acc)
    else
     xpBuild(n,sf+1,acc)
    (**)
   (**)
  (**)
 in
  rev(xpBuild(n,2,[]))
 end
;

(*-------------------------------------------------*)

------=_NextPart_000_0006_01BD26BA.23FD0940
Content-Type: application/octet-stream;
        name="Primes.ml"
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment;
        filename="Primes.ml"

(*  You may recall that I sent you some code last term
    which produced prime number.
    I thought of a different method using integer
    streams.
    I asked both versions (old version included) to
    produce the first 887 prime number.
    The old version took 11.5 seconds.
    The new version took  1.5 seconds!
    Unfortunately, the new version seems to be very
    space inefficient.  Trying to generate more primes
    with the new version causes it to crash CML (windows
    closes it down!).
                                                      *)
```

```
(*-------------Generic Stream Manipulations------------*)

datatype stream = Item of int * (unit->stream);
fun cons (x,xs) = Item(x,xs);
fun head (Item(i,xf)) = i;
fun tail (Item(i,xf)) = xf();
fun makeints n = cons(n, fn()=> makeints(n+1));
fun nth(s,n) = if n=1 then head(s) else nth(tail(s),n-1);

fun ton(n,s,xs) = if head(s)>n then
                        xs
                    else
                        head(s)::ton(n,tail(s),xs);

fun filter f xs = if f(head(xs)) then
                    cons(head(xs),fn()=>(filter(f) (tail(xs))))
                  else
                    filter (f) (tail(xs));
(*-----------------------------------------------------*)



(*-------------Prime Number Generation Code------------*)

fun notdiv n x =
        if (x mod n)=0 andalso n<>x then false else true;

fun sqr(x:int) = x*x;

fun makeprimes(n) =
  let fun xmakeprimes(n,acc,prms)=
    if n=acc then
      ton(sqr(nth(prms,n+1))-1,prms,[])
    else
      xmakeprimes(n,acc+1,(filter (notdiv (nth(prms,acc+1))) (prms)))
  in
    xmakeprimes(n,1,makeints 1)
  end;
(*-----------------------------------------------------*)
```

```
(*-----------Old Prime Number Generation Code----------*)

fun pIsPrime(n) =
        let
                fun xpIsPrime(n,sf) =
                        if sf=n then
                                true
                        else
                                if n mod sf = 0 then
                                        false
                                else
                                        xpIsPrime(n,sf+1)
                                (**)
                        (**)
                (**)
        in
                xpIsPrime(n,2)
        end
;

fun pBuild(n) =
        let
                fun xpBuild(n,sf,acc) =
                        if n=0 then
                                acc
                        else
                                if pIsPrime(sf) then
                                        xpBuild(n-1,sf+1,sf::acc)
                                else
                                        xpBuild(n,sf+1,acc)
                                (**)
                        (**)
                (**)
        in
                rev(xpBuild(n,2,[]))
        end
;

(*-----------------------------------------------------*)
------=_NextPart_000_0006_01BD26BA.23FD0940--

Joseph Marshall's prime finding and factorization program.

fun DivBy(m,[])=false
```

```
  | DivBy(m,l::ls)=if((m mod l)=0) then true
      else DivBy(m,ls);

fun PrimeList(n)=
  let fun PLCalc(n,m,ls)= if(m>n)then ls else
        if DivBy(m,ls) then PLCalc(n,m+1,ls)
                       else PLCalc(n,m+1,m::ls)
  in  PLCalc(n,2,[])
  end;


fun Factorize(n)=
  let val flist=PrimeList(n)
  in
    let fun FCalc(n,[],fl)=fl
      | FCalc(n,pf::pfs,fl)=if((n mod pf)=0) then FCalc(n div
pf,pf::pfs,pf::fl)
                               else FCalc(n,pfs,fl)
    in
      FCalc(n,flist,[])
    end
  end;




(*  ------------------------------------------------  *)
(*                     PRIME FACTORS                  *)
(*                      ANDREW ROSE                   *)
(*                       20/11/97                     *)
(*  ------------------------------------------------  *)

(*  The code below defines the following functions...

pIsPrime        - Tests to see if a number is prime
pBuild(n)       - Builds a list of the first n primes
pFact(n)        - Finds the prime factors of a list
pFactFromList(n)- Finds the prime factors of a list
                  given a list of primes.  If the list
                  of primes is exhausted before all the
                  factors have been foundd then the last
                  factors are found using pFact.

                                                       *)
```

```
fun pIsPrime(n) =
        let
                fun xpIsPrime(n,sf) =
                        if sf=n then
                                true
                        else
                                if n mod sf = 0 then
                                        false
                                else
                                        xpIsPrime(n,sf+1)
                                (**)
                        (**)
                (**)
        in
                xpIsPrime(n,2)
        end
;

fun pBuild(n) =
        let
                fun xpBuild(n,sf,acc) =
                        if n=0 then
                                acc
                        else
                                if pIsPrime(sf) then
                                        xpBuild(n-1,sf+1,sf::acc)
                                else
                                        xpBuild(n,sf+1,acc)
                                (**)
                        (**)
                (**)
        in
                rev(xpBuild(n,2,[]))
        end
;

fun pFact(n) =
        let
                fun xpFact(n,sf,acc) =
                        if sf > n then
                                acc
                        else
                                if (n mod sf) = 0 then
```

```
                                        xpFact(n div sf, sf, sf::acc)
                                else
                                        xpFact(n, sf+1, acc)
                                (**)
                        (**)
                (**)
        in
                xpFact(n,2,[])
        end
;

exception BotchUp

fun pFactFromList(n,pList) =
        let
                fun xpFact(n,sf,acc) =
                        if sf > n then
                                acc
                        else
                                if (n mod sf) = 0 then
                                        xpFact(n div sf, sf, sf::acc)
                                else
                                        xpFact(n, sf+1, acc)
                                (**)
                        (**)
                (**)
                fun xpFactFromList(n,[],acc) = raise BotchUp
                  | xpFactFromList(n,sf::[],acc) =
                        xpFact(n,sf,acc)
                  | xpFactFromList(n,pList,acc) =
                        if hd(pList) > n then
                                acc
                        else
                                if (n mod hd(pList)) = 0 then
                                        xpFactFromList(n div hd(pList), pList, hd(pList)
                                else
                                        xpFactFromList(n, tl(pList), acc)
                                (**)
                        (**)
                (**)
        in
                xpFactFromList(n,pList,[])
        end
;
```

$(A \times C) \cup (B \times D) = (A \cup B) \times (C \cup D)$?

Sets are **extensional**: two sets with the same members are the same set. So let's see which things belong to the LHS and to the RHS. '`fst`' and '`snd`' are the obious operation for taking ordered pairs apart.

$x \in \text{LHS}$ iff

$x \in A \times C \vee x \in B \times D$ iff

$(\texttt{fst}\ x \in A \wedge \texttt{snd}\ x \in C) \vee (\texttt{fst}\ x \in B \wedge \texttt{snd}\ x \in D)$

and

$x \in \text{RHS}$ iff

$\texttt{fst}x \in (A \cup B) \wedge \texttt{snd}x \in (C \cup D)$ iff

$(\texttt{fst}x \in A \vee \texttt{fst}x \in B) \wedge (\texttt{snd}x \in C \vee \texttt{snd}x \in D)$.

We can introduce some abbreviations to make this legible: abbreviate

'$\texttt{fst}x \in A$ to 'a'

'$\texttt{fst}x \in B$ to 'b'

'$\texttt{snd}x \in C$ to 'c'

'$\texttt{snd}x \in D$ to 'd'

and then the LHS and the RHS become

$(a \wedge c) \vee (b \wedge d)$ and

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

which are clearly inequivalent.