Examen Md2 Febrero. 2001 (con soluciones)

Verificación [1]

1. Con los términos que se dan a continuación, rellenar los huecos de la siguiente tabla de forma que los términos de la segunda columna tengan como tipo los correspondientes situados en la tercera. Da una explicación de tu elección. (La primera línea de la tabla es un ejemplo: *I)* `[_:nat]nat` *es una función constante que a cualquier valor de tipo* `nat` *le hace corresponder el mismo objeto* `nat` *de tipo* `Set`)

   a) `(n:nat){m:nat | (m=O\/n=(S m))}`

   b) `[X:Set][b:bool][x,y:X]Cases b of true => x | _ => y`

   c) `nat -> Set`

   d) `nat -> Prop`

   f)
   ```
   Fixpoint inf_egal [n:nat] : nat-> bool :=
       [m:nat]
       Cases n m of
       O m => true
       | (S n) O => false
       | (S n) (S m) => (inf_egal n m)
       end.
   ```

I	`[_:nat]nat`	**c**
II	**b**	`(X:Set)bool->(x,y:X)X`
III	`[m:nat]m=O\/O=(S m)`	**d**

IV	`[n:nat] Cases n of`
	`O =>(exist nat [m:nat]m=O\/O=(S m) O`
	`(or_introl O=O O=(S O) (refl_equal nat O)))`
	`|(S n0) => (exist nat [m:nat]m=O\/(S n0)=(S m) n0`
	`(or_intror n0=O (S n0)=(S n0) (refl_equal nat (S n0))))`
	`end`
		a

V	**f**	`nat -> nat -> bool`

Indicaciones:

```
Inductive sig [A:Set; P:A->Prop] : Set :=
      exist : (x:A)(P x)->{x:A | (P x)}

Inductive or [A:Prop; B:Prop] : Prop :=
      or_introl : A->A\/B | or_intror : B->A\/B
```

[1] Calificación: 1:(3pt:1.25 por cubrir bien el cuadro + 1.75 por explicar bien);2:(2pt);3:(1.5pt);4:(3.5pt)

`Inductive eq [A:Set; x:A] : A->Prop :=  refl_equal : (eq A x x)`

II: *Un término de este tipo ha de ser un procedimiento que a cada* `X:Set` *le haga corresponder una función de tipo* `bool ->(x,y:X)X`*. Será pues* `[X:Set]f` *donde* `f:bool ->(x,y:X)X`*. Como el tipo* `(x,y:X)X` *es un producto no dependiente, es lo mismo que* `X -> X ->X`*, de modo que el término buscado ha de ser de la forma* `[X:Set][b:bool][x,y:X](f b x y)`*. Claramente el término* **b** *tiene esta forma para un* `f` *concreto.*

III: *Este término hace corresponder, a cada* `m:nat` *la disyunción de dos proposiciones y por lo tanto el tipo resultante es una proposición*

IV: *Se tiene* `(refl_equal nat O)):(O=O)` *y*

`(or_introl O=O O=(S O) (refl_equal nat O))):(O=O \/ O=(S O))`*. Y entonces*

`(exist nat [m:nat]m=O\/O=(S m) O (or_introl O=O O=(S O) (refl_equal nat O)))` *tiene tipo (es una prueba de)* `{m:nat | (O=m\/O=(S m))}`*. Nótese que los dos últimos argumentos de* `exist` *son, precisamente el testigo* `O` *y una prueba de que* `O` *cumple la propiedad* `m=O\/O=(S m)`*.*

La otra opción tiene una explicación análoga

V: *La función* `inf_egal` *es una función recursiva definida en Coq mediante el mecanismo* `Fixpoint` *que declara el tipo* `nat->nat->bool`*. Además el código de su definición asegura que los resultados son siempre de tipo* `bool`*, siendo la llamada recursiva hecha sobre valores "inferiores" a los del patrón de la izquierda. Ello asegura la terminación.*

De hecho, la aceptación de una definición recursiva en este sistema asegura su terminación, pues Coq hace siempre un test sobre el hecho de que las llamadas recursivas sean hechas sobre valores "menores" cada vez.

2. Calcula el valor de `([n:nat](O=n) O)` y calcula el tipo del término `(nat_ind [n:nat](O=n) p)` siendo `p:O=O`.

   Se trata de calcular el valor de la función de tipo `nat -> Prop` *que a cada* `n:nat` *le hace corresponder la proposición* `n=O`*. Por lo tanto, al aplicarla a* `O` *obtenremos* `O=O`*.*

   Teniendo en cuenta que el tipo de `nat_ind` *es:*

   `(P:(nat->Prop))(P (O))->((n:nat)(P n)->(P (S n)))->(n:nat)(P n)`

   el término en cuestión será

   `((n:nat)O=n->O=(S n))->(n:nat)O=n`*.*

   pues a `nat_ind` *le pasamos el primer y segundo argumentos.*

3. Sea `R:nat->nat->Prop` una relación binaria en `nat`.

   Si `p` es una prueba de la especificación `(x:nat){y:nat | (R x y)}` (o sea `p:(x:nat){y:nat | (R x y)}`). ¿Qué se obtiene al realizar en Coq `Extraction p` y qué tipo tiene?

   El término `p` *tiene que ser de la forma*

   `[x:nat](exist nat [y:nat](R x y) y' p')` *donde* `y':nat` *es el testigo (elemento que existe) y* `p':(R x y')`*. Al extraer el programa de* `p`*, la prueba desaparece y nos quedamos con la función que a* `x:nat` *le asocia* `y':nat`*. Se obtiene pues una función de tipo* `nat -> nat` *que se dice que implementa correctamente la especificación dada por la relación* `R`*.*

4. :

   i) Explica el significado de las dos siguientes expresiones:

   - ```
     Inductive le [n:nat] : nat->Prop :=
            le_n : (le n n) | le_S : (m:nat)(le n m)->(le n (S m))
     ```
     ```

Esto es la definición inductiva del predicado binario (o relación binaria) `le` *(menor o igual que). Los axiomas o propiedades que le pedimos a esta relación son la reflexividad (cuya prueba está dada por el constructor (o axioma)* `le_n`*, y la que dice que para todo m si (n,m) están relacionados (n es menor o igual que m), entonces (n, (S m)) también lo están. Nótese que se trata de un tipo recursivo pues para tener un objeto de tipo* `(le n (S m))` *(es decir, una prueba de esta proposición) necesitamos otro de tipo* `(le n m)`*.*

- `lt = [n,m:nat](le (S n) m) : (_,_:nat)Prop`
  `lt` *es una función de tipo* `nat->nat->Prop` *que define la relación de ser menor que entre naturales.* `(lt n m)=(le (S n) m)`*.*

ii) Explica detalladamente el siguiente comportamiento:

```
  n : nat
  m : nat
  p : nat
  H : (le n m)
  H0 : (le m p)
  ============================
   (le n p)

 < Elim H0.
2 subgoals

  n : nat
  m : nat
  p : nat
  H : (le n m)
  H0 : (le m p)
  ============================
   (le n m)

subgoal 2 is:
 (m0:nat)(le m m0)->(le n m0)->(le n (S m0))
```

Indicación: El esquema de inducción del tipo `le` es

```
le_ind: : (n:nat; P:(nat->Prop))
     (P n)
     ->((m:nat)(le n m)->(P m)->(P (S m)))
     ->(n0:nat)(le n n0)->(P n0)
```

Para simplificar escribiremos, a veces, $x \leq y$ *en lugar de* `(le x y)`*.*

Al hacer la eliminación del elemento `H0` *de tipo* `(le m p)` *aplicamos el esquema* `le_ind` *con primer argumento* `m` *y segundo argumento el predicado* `P` *dado por* $P(x) = (n \leq x)$ *(en Coq* `[x:nat](le n x)`*). Esto último se ve porque la conclusión de nuestro esquema es* `(P n0)` *y debe coincidir con nuestro objetivo cuando* `n0` *es* `p`*.*

El tipo de `(le_ind m P)` *será*

```
(le n m))->(m0:nat)(m0:nat)(le m m0)->(le n m0)->(le n (S m0)
 -> (n0:nat)(le m n0)->(le n n0).
```

Nótese que lo que intentamos hacer al eliminar `H0` *es justamente ver que nuestro objetivo* `(le n p)` *se deduce de* `(le m p)`*. Pero esa implicación es la última linea del tipo anterior al tomar como* `n0` *precisamente* `p`*. Por lo tanto el sistema nos deja como subobjetivos los dos de la línea anteror de dicho tipo.*

iii) Interpreta el siguiente teorema:

```
Theorem  div_euclid:(a,b:nat)
{qr:(nat*nat) | (let (q,r)=qr in
           a=(plus r (mult q (S b)))/\(lt r  (S b)))}.
```

Este teorema dice:

$$\forall a, b : nat\ \exists q, r : nat \mid a = q * (b + 1) + r \wedge (r < (b + 1))$$

que corresponde con el Teorema de Euclides de la división entera relativo a la división de cualquier a entre $b + 1$ para cualquier b, obteniendose un cociente q y un resto r.

iv) Describe la naturaleza de `div_euclid`.

El término `div_euclid` *debe ser una función que para cada* `a,b:nat` *construye, a partir de un par* `(q,r):nat*nat` *y una prueba de la proposición:*

```
a=(plus r (mult q (S b)))/\(lt r (S b)))
```

mediante el constructor `exist` *un objeto del tipo:*

```
{qr:(nat*nat) | (let (q,r)=qr in a=(plus r (mult q (S b)))/\(lt r (S b)))}.
```

1. Encuentra las funciones generatrices y las sucesiones correspondientes, dadas por las convoluciones:

$$a_n = 1 \text{ si } 0 \le n \le 4 \text{ y } a_n = 0 \text{ si } n \ge 5 \quad \text{con} \quad b_n = n \ \forall n \tag{1}$$

$$a_n = (-1)^n \ \forall n \quad \text{con} \quad b_n = (-1)^n \ \forall n \tag{2}$$

La sucesión $< a_n >$ viene dada por los coeficientes del polinomio

$$1 + x + x^2 + x^3 + x^4 = \frac{1 - x^5}{1 - x}$$

y la $< b_n >$ por los coeficientes de la serie

$$0 + x + 2\,x^2 + 3\,x^3 + \cdots = \frac{x}{(1-x)^2}$$

por lo tanto, su convolución será el producto de ambas:

$$\frac{x(1-x^5)}{(1-x)^3}$$

Los términos de la sucesión correspondiente serán los coeficientes del desarrollo de esta función. Poniendo $\frac{x(1-x^5)}{(1-x)^3} = x(1-x)^{-3} - x^6(1-x)^{-3}$, el término general de la sucesión será:

$$\left(\begin{array}{c} 3 + (n-1) - 1 \\ n - 1 \end{array} \right) - \left(\begin{array}{c} 3 + (n-6) - 1 \\ n - 6 \end{array} \right) = 5\,n - 10$$

En el segundo caso la función generatriz asociada a $< a_n > =< b_n >$ es $\frac{1}{1+x}$. Por lo tanto, la convolución es su cuadrado

$$\frac{1}{(1+x)^2} = (1+x)^{-2}$$

Los términos de la sucesión correspondiente serán los coeficientes del desarrollo de esta función. O sea, el término general

$$\left(\begin{array}{c} -2 \\ n \end{array} \right) = (-1)^n \left(\begin{array}{c} 2 + n - 1 \\ n \end{array} \right) = (-1)^n\,(n+1)$$

que es la sucesión alternada

$$1, -2, 3, -4, 5, \cdots.$$

2. Calcula la función generatriz que nos da el número de funciones sobreyectivas entre un conjunto de 7 elementos y otro de 3.

Si llamamos A, B, C a los elementos del segundo conjunto, el problema consiste en contar el número de palabras de siete letras que se pueden formar con estas tres, de modo que, en cada palabra aparezca cada una de ellas, al menos una vez, (por ser la función, sobreyectiva). Por ejemplo $AABACBC$.

Nótese que, por ejemplo, en una palabra que aparezcan varias A's, éstas son indistinguibles entre sí, por lo que, la función que nos resuelve el problema debe ser la generatriz exponencial.

La función generatriz exponencial de estas disposiciones es

$$\left(x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \right)^3 = (e^x - 1)^3 = e^{3x} - 3e^{2x} + 3e^x - 1$$

[2]Calificación: 1:(4pt); 2:(3pt); 3:(3pt)

La respuesta es, entonces, el coeficiente de $\frac{x^7}{7!}$ en la función anterior, es decir

$$3^7 - 3\,(2^7) + 3\,(1^7) = \sum_{i=0}^{3} (-1)^i \left(\begin{array}{c} 3 \\ i \end{array} \right) (3-i)^7$$

Nótese que, cambiando 3 y 7 por n y m cualesquiera, se obtiene el número de aplicaciones sobreyectivas de un conjunto de m elementos sobre uno de n, siempre que $n \leq m$. Es decir

$$\sum_{i=0}^{n} (-1)^i \left(\begin{array}{c} n \\ i \end{array} \right) (n-i)^m$$

3. Resolver la relación de recurrencia:

$$n\,a_n + n\,a_{n-1} - a_{n-1} = 2^n \text{ donde } a_0 = 10$$

Para simplificar nuestra relación de recurrencia, hagamos $b_n = n\,a_n\ \forall n$. De aquí, la relación a resolver es

$$b_n + b_{n-1} = 2^n\ \forall n > 0$$

y $b_0 = 0$.

Resolviendo, por el método de las funciones generatrices, tenemos:

$$\sum_{n=1} b_n x^n + \sum_{n=1} b_{n-1} x^n = \sum_{n=1} 2^n x^n$$

y, llamando $f(x) = \sum_{n=0} b_n x^n$, tenemos:

$$\begin{aligned} \sum_{n=1} b_n x^n &= f(x) - b_0 \\ \sum_{n=1} b_{n-1} x^n &= x\,f(x) \\ \sum_{n=1} 2^n x^n &= \frac{1}{1-2x} - 1 \end{aligned}$$

de donde

$$(f(x) - b_0) + x f(x) = \frac{1}{1-2x} - 1$$

y, como $b_0 = 0$, queda:

$$f(x)(1+x) = \frac{1}{1-2x} - 1$$

y por lo tanto

$$f(x) = \frac{1}{(1-2x)(1+x)} - \frac{1}{1+x}$$

Descomponiendo en fracciones simples

$$\frac{1}{(1-2x)(1+x)} = \frac{A}{1-2x} + \frac{B}{1+x}$$

obteniendose $A = 2/3$ y $B = 1/3$. Por lo tanto:

$$f(x) = \frac{2/3}{1-2x} - \frac{2/3}{1+x}$$

y el coeficiente de x^n en el desarrollo de esta función es:

$$b_n = (2/3)2^n - (2/3)(-1)^n$$

y la solución buscada:

$$a_n = \frac{b_n}{n} = \frac{2}{3n}(2^n + (-1)^{n+1})\ \forall n \geq 1$$