Práctico 3

Asistentes de prueba para programadores

- Cálculo de construcciones -

Objetivos Manejar conceptos básicos del cálculo λ tipado (tipos polimórficos y tipos dependientes de objetos).

Familiarizarse con las nociones de δ, β y η reducción. Se verá el manejo de esas nociones en Coq a través de las estrategias de reducción implementadas en las tácticas de simplificación.

Cálculo de construcciones

`Set` es el tipo de los conjuntos

`Type` es el tipo de los constructores de tipos

A `->` B denota el tipo de las funciones de A en B

`(forall x:T, U)` denota la familia de tipos U indexada por T

`(fun (x:T) => U)` denota la función (λ abstracción) de x de tipo de T cuyo cuerpo es U ([x:T] U)

`(`T U `)` denota la aplicación del término U a T

Principales tácticas a utilizar en estos ejercicios

`cbv` $flag_1$ $flag_2$

- se utiliza para reescribir un término usando las reducciones indicadas en $flag_1$ $flag_2$. Por el momento, cada $flag_i$ debe ser `beta` o `delta`. Puede acompañarse de una lista de nombres de constantes para ser utilizadas en aplicaciones de la regla δ (unfold). Ver manual.

`lazy` $flag_1$ $flag_2$

- se utiliza para reescribir un término usando las reducciones indicadas en $flag_1$ $flag_2$. La estrategia de reducción utilizada es "lazy", o sea, las reducciones de los argumentos de las funciones se hacen sólo cuando es necesario para poder seguir reduciendo. Por el momento, cada $flag_i$ debe ser `beta` o `delta`. Puede acompañarse de una lista de nombres de constantes para ser utilizadas en aplicaciones de la regla δ (unfold). Ver manual.

`compute`

- es un alias para `Cbv beta delta iota`.

`simpl`

- efectúa β-reducciones y expande constantes transparentes (aquellas que no son lemas).

Comandos de extensión de sintaxis:

`Infix`.

Tipos simples y paramétricos

Ejercicio 3.1. Considere los siguientes tipos en el contexto A,B,C: Set

- $(A \to A \to A) \to A \to A$
- $(A \to B) \to (B \to C) \to A \to B \to C$
- $(A \to B) \to (A \to B \to C) \to A \to C$
- $(A \to B) \to ((A \to B) \to C) \to C$

1. Encuentre al menos un habitante (un elemento) de cada tipo, indicando para cada uno de ellos si está en β forma normal.
2. Verifique en Coq que los términos propuestos tienen los tipos esperados.
3. Demuestre en Coq que dichos tipos son tautologías. Para ello utilice `tauto` y luego rehaga las pruebas utilizando la táctica `exact`.

Ejercicio 3.2. Complete los términos siguientes reemplazando cada "?" por un tipo de forma que el término esté bien tipado. Calcule el tipo de cada término y verifique que éste es correcto utilizando `Check`.

1. [x:?][y:?][z:?]((x y) z)
2. [x:?][y:?][z:?](x (y z))
3. [z:?] ([x:?][y:?](x y) z)
4. ([x:?]x [y:?]y)

Ejercicio 3.3. Sean A,B,C de tipo Set. Escriba en Coq los siguientes operadores:

1. el operador **apply** de tipo $(A \to B) \to A \to B$ que aplique una función f a su argumento.
2. el operador **o** de composición, de tipo $(A \to B) \to (B \to C) \to (A \to C)$.
3. el operador **twice**, que aplica dos veces una función a su argumento.
4. Redefina los operadores de las partes anteriores de este ejercicio de forma que sean aplicables cualesquiera sean los conjuntos A, B y C.

Ejercicio 3.4. Considere el operador de composición en el ejercicio 3.3. Defina en Coq la función de identidad id: $A \to A$ y demuestre que para todo x:A, se cumple que (id ° id) x = id x. Escriba tres pruebas de esta igualdad utilizando diferentes tácticas para δ y β reducción (`unfold`, `cbv beta`, `cbv delta`, `cbv beta delta`, `compute` y `simpl`).

Ejercicio 3.5.

1. Escriba en Coq la versión paramétrica de los operadores I, K, S

   opI $=_{def}$ [x] x

   opK $=_{def}$ [x][y] x

   opS $=_{def}$ [f][g][x] ((f x) (g x))

2. Escriba en Coq el enunciado de un lema que establezca la igualdad opS opK opK = opI (para ello deberá asignar tipos adecuados a los operadores de modo que las aplicaciones sean posibles). Pruebe dicho lema utilizando las tácticas para β y δ reducción.

Ejercicio 3.6. Considere la siguiente codificación de los naturales en cálculo lambda propuesta por Church:

```
Definition N := forall X : Set, X -> (X -> X) -> X.
Definition Zero (X : Set) (o : X) (f : X -> X) := o.
Definition Uno (X : Set) (o : X) (f : X -> X) := f (Zero X o f).
```

1. Defina la constante `Dos` que representa el natural 2.
2. Defina la función sucesor de tipo `N->N` y pruebe que el sucesor de 1 es igual a 2.
3. Considere la siguiente definición de la suma para los naturales de Church con la correspondiente notación infija "++":

   ```
   Definition Plus (n m : N) : N :=
     fun (X : Set) (o : X) (f : X -> X) => n X (m X o f) f.
   Infix "++" := Plus (left associativity, at level 94).
   ```

   Demuestre los siguientes lemas:

   a. `Lemma suma1: (Uno ++ Zero) = Uno.`

   b. `Lemma suma2: (Uno ++ Uno) = Dos.`

4. Defina la función `Prod:N->N->N` que calcula el producto de dos naturales de Church. Asocie la notación infija ** para dicha función.
5. Demuestre los siguientes lemas:

   a. `Lemma prod1 : (Uno ** Zero) = Zero.`

   b. `Lemma prod2: (Dos ** Uno) = Dos.`

Ejercicio 3.7.

1. Defina en Coq una codificación para el tipo de los booleanos y asóciele el nombre `Bool`. Defina los símbolos `t` y `f` para denotarlos valores booleanos true y false respectivamente.
2. Defina un operador `If: Bool->Bool->Bool->Bool` tal que dados tres valores de tipo `Bool` `b, then` y `else`, el término `(if b then else)` reduzca a `then` si el valor de `b` es `t` y a `else` si el valor de `b` es `f`.
3. Defina la función de negación `Not: Bool->Bool` con la semántica usual y pruebe el siguiente lema de corrección:
   `Lemma CorrecNot : (Not t) = f /\ (Not f) = t.`
4. Defina dos implementaciones diferentes para la función `And: Bool->Bool->Bool`.
5. A una de las implementaciones de la parte anterior asóciele el símbolo infijo `&` y demuestre el siguiente lema de corrección:
   `Lemma CorrecAnd : (t & t) = t /\ (f & t) = f /\ (t & f) = f.`

Tipos dependientes

Ejercicio 3.8. Considere la siguiente definición del constructor de tipos `Array` y los siguientes operadores:

```
Section ArrayNat.
Parameter ArrayNat : forall n : nat, Set.
Parameter empty : ArrayNat 0.
Parameter add : forall n : nat, nat -> ArrayNat n -> ArrayNat (S n).
```

1. Calcule el tipo del siguiente vector `(add 0 (S 0) empty)`.
2. Construya utilizando los operadores `empty` y `add` un vector de ceros de largo 2, y un vector de largo 4 que contenga [0, 1, 0, 1]. Utilice `Check` para verificar que el tipo es correcto.
3. Defina un parámetro `concat` que permita concatenar dos vectores de largo arbitrario. (La operación de suma de naturales en Coq se llama `plus`).
4. Defina un parámetro `zip` que dados dos vectores de igual largo, y una función `f:nat->nat->nat` devuelva el vector resultado de aplicar la función `f` a cada pareja de elementos de igual posición.
5. Calcule con `Check` el tipo de `ArrayNat`.
6. Generalice en Coq la definición de `ArrayNat` para que sea paramétrica en el tipo de los elementos del vector. Redefina los parámetros `empty`, `add` y `zip`.
7. Utilizando la definición anterior defina el tipo `ArrayBool` de vectores de booleanos, de largo arbitrario.

Ejercicio 3.9.

1. Defina en Coq como parámetro el constructor de tipos `MatrizNat` que permite construir matrices de naturales.

2. Defina en Coq los siguientes operadores como parámetros:

   a. `prod`: calcula el producto de dos matrices

   b. `es_id`: es una función booleana que determina si una matriz es la matriz de identidad.

   c. `ins_fila`: que dada una matriz matriz `n×m` y un array de `m` elementos devuelve la matriz `(n+1)×m`

   d. `ins_columna` (análogo a `ins_fila`)

3. Calcule con `Check` el tipo de `MatrizNat`.

Ejercicio 3.10. Considere la siguiente definición del constructor de tipos Array (paramétrico en el largo, y de elementos de un tipo genérico) y los siguientes operadores:

```
Parameter Array: Set -> nat -> Set.
Parameter empty : forall X : Set, Array X 0.
Parameter addA: forall (X : Set) (n : nat), X -> Array X n -> Array X (S n).
```

Considere asimismo el constructor de tipos `Matrix` de matrices *escalonadas* de *n* columnas (de elementos de un tipo genérico), donde la primera columna contiene un sólo elemento, la segunda columna 2 elementos, la tercera 3 y así sucesivamente, de tal manera que la columna *n* posee *n* elementos.

```
Parameter Matrix : Set -> nat -> Set.
```

1. Defina el tipo del operador `emptyM` que permita representar una matriz escalonada vacía (sin elementos, de 0 filas).

2. Defina el tipo del operador `addM` que permita generar matrices escalonadas no vacías de *n* columnas. Este operador junto con `emptyM` deberían permitir crear cualquier matriz escalonada de *n* columnas.

3. Construya utilizando los operadores `emptyM` y `addM` una matriz escalonada de 3 columnas de números naturales, cuyos elementos cumplan que: los elementos en la columna *i* sean todos iguales a *i*. En este parte use el tipo `nat` predefinido de Coq.

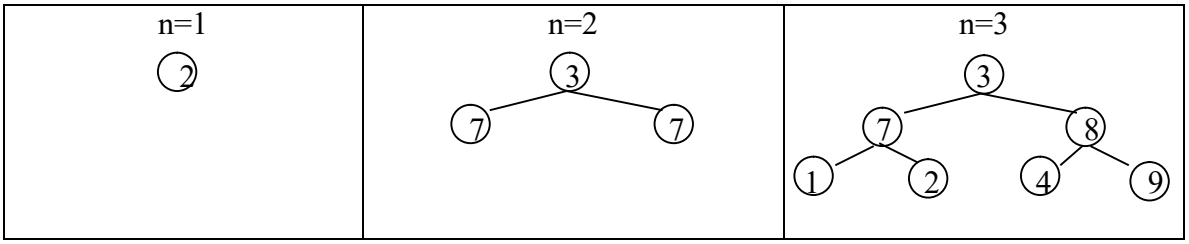
Ejercicio 3.11. Considere el constructor de tipos `ABNat` de árboles binarios completos de altura `n`, de números naturales:

```
Parameter ABNat : forall n : nat, Set.
```

1. Defina el tipo del operador `emptyAB` que permita representar un árbol binario de naturales vacío de altura 0.

2. Defina el tipo del operador `addAB` que permita generar árboles binarios completos no vacíos de altura mayor que cero, de números naturales.

   Tenga en cuenta que un árbol binario completo tiene todos sus niveles llenos. Los siguientes son ejemplos de árboles binarios completos de naturales de altura n:



3. Construya utilizando los operadores emptyAB y addAB el árbol binario completo de altura 2 del ejemplo previo. Verifique que el árbol construido tiene el tipo adecuado.

4. Generalice las definiciones `ABNat`, `emptyAB` y `addAB` para trabajar con elementos de un tipo genérico (en vez de `nat`).

Ejercicio 3.12. Considere el constructor de tipos `AVLNat` de árboles binarios de altura `n` y elementos de un tipo genérico `A`, que cumplen la propiedad de estructura de los árboles binarios balanceados AVL. Esto es, que para cada nodo del árbol, la altura de sus subárboles puede diferir en a lo sumo una unidad.

```
Parameter AVLNat : forall n : nat, Set.
```

1. Defina el tipo del operador `emptyAVL` que permita representar el AVL de naturales vacío de altura 0.

2. Defina tipos para operadores $addAVL_i$ (i=1..k) que permitan generar, conjuntamente, todos los árboles binarios balanceados AVL no vacíos de altura mayor que cero, de números naturales.

3. Construya utilizando los operadores emptyAVL y $addAVL_i$ un árbol binario AVL de altura 2. Verifique que el árbol construido tiene el tipo adecuado.

4. Generalice las definiciones `AVLNat`, `emptyAVL` y $addAVL_i$ para trabajar con elementos de un tipo genérico (en vez de `nat`).

Programación de Pruebas – Isomorfismo de Curry Howard

Objetivo: Trabajar en la programación de pruebas utilizando el isomorfismo de Curry Howard.

Ejercicio 3.13. Demuestre en Deducción Natural las siguientes tautologías y encuentre los términos correspondientes a las mismas. Escriba en Coq las pruebas de las tautologías en un sólo renglón utilizando la táctica `exact` con el término asociado.

1. $(A \to B \to C) \to B \to A \to C$
2. $(A \to B) \to (B \to C) \to A \to C$
3. $(A \to B \to C) \to (B \to A) \to B \to C$

Ejercicio 3.14. Considere las siguientes pruebas incompletas en Coq, y complételas en un sólo renglón utilizando la táctica `exact` con el término asociado.

1.
```
Lemma Ej314_1 : (A -> B -> C) -> A -> (A -> C) -> B -> C.
Proof.
      intros f a g b.
      ...
```
2.
```
Lemma Ej314_2 : A -> ~ ~ A.
Proof.
      unfold not.
      intros.
      ...
```
3.
```
Lemma Ej314_3 : (A -> B -> C) -> A -> B -> C.
Proof.
      ...
```
4.
```
Lemma Ej314_4 : (A -> B) -> ~ (A /\ ~ B).
Proof.
      unfold not.
      intros.
      elim H0; intros.
      ...
```

Ejercicio 3.15. Considere las siguientes declaraciones:

```
Reset Initial.

Variable U : Set.
Variable e : U.
Variable A B : U -> Prop.
Variable P : Prop.
Variable R : U -> U -> Prop.
```

Complete las siguientes pruebas en Coq en un sólo renglón utilizando la táctica `exact` con el término asociado.

1.
```
Lemma Ej315_1 : (forall x : U, A x -> B x) -> (forall x : U, A x) ->
      forall x : U, B x.
Proof.
      intros.
      ...
```
2.
```
Lemma Ej315_2 : forall x : U, A x -> ~ (forall x : U, ~ A x).
```

```
Proof.
      unfold not.
      intros.
      ...
```

3.

```
Lemma Ej315_3 : (forall x : U, P -> A x) -> P -> forall x : U, A x.
Proof.
      ...
```

4.

```
Lemma Ej315_4 : (forall x y : U, R x y) -> forall x : U, R x x.
Proof.
      ...
```

5.

```
Lemma Ej315_5 : (forall x y: U, R x y -> R y x) ->
      forall z : U, R e z -> R z e.
Proof.
      ...
```

Ejercicios a entregar:

Ver la fecha límite y los ejercicios requeridos en el sitio EVA del curso.

El archivo a entregar debe cargar correctamente en Coq. Si deja ejercicios sin resolver, debe delimitarlos como comentarios: (... *).*

Al inicio del archivo deben estar los datos de cada integrante; se admiten entregas individuales o de a dos estudiantes.

Usar la plantilla publicada junto con el práctico para el desarrollo de los ejercicios requeridos; no es necesario entregar los ejercicios no solicitados.