Práctico 5

Definiciones Inductivas

- Segunda Parte -

Objetivos: Trabajar con tipos inductivos. Realizar pruebas por inducción y análisis de casos. Familiarizarse con los lemas de inversión y las tácticas asociadas.

Principales tácticas a utilizar en estos ejercicios:

`inversion` id, `inversion_clear` id, `Derive Inversion ... with ...`, `inversion <id> using <invlemma>`, `inversion <id> using <invlemma> in...`

Ejercicio 5.1. Considere las siguientes relaciones inductivas:

```
Inductive LE : nat -> nat -> Prop :=
  | Le_O : forall n : nat, LE 0 n
  | Le_S : forall n m : nat, LE n m -> LE (S n) (S m).


Inductive Mem (A : Set) (a : A) : list A -> Prop :=
  | here : forall x : list A, Mem A a (cons A a x)
  | there : forall x : list A, Mem A a x ->
      forall b : A, Mem A a (cons A b x).
```

Demuestre los siguientes resultados usando las tácticas de inversión:

1. `Theorem not_sn_le_o: forall n:nat, ~ LE (S n) O.`
2. `Theorem  nil_empty: forall a:A, ~ Mem a (nil A).`
3. 4 no es menor o igual que 3.
4. Para todo natural n, el sucesor de n no es menor o igual que n.
5. La relación menor o igual (LE) es transitiva.
6. Si un elemento pertenece a una lista, entonces pertenece a la concatenación de esta lista con cualquier otra.

Ejercicio 5.2. Considere las definiciones de árboles binarios, isomorfismo entre árboles binarios y la función que retorna el árbol espejo de uno dado del práctico anterior. Pruebe las siguientes propiedades:

1. El árbol vacío no es isomorfo a ningún árbol no vacío (con al menos un nodo).
2. Si dos árboles no vacíos son isomorfos, entonces también lo son sus subárboles izquierdos y derechos, respectivamente.
3. La relación de isomorfismo entre árboles binarios es transitiva.
4. Si dos árboles binarios son isomorfos, también los son sus árboles espejos.

Ejercicio 5.3. Se quiere definir conjuntos en Coq como una restricción de listas.

1. Defina, a partir de las siguientes declaraciones, la relación inductiva unaria `isSet` que contiene a las listas sin elementos repetidos.

```
Section Ej5_3.

  Variable A : Set.

  Parameter equal : A -> A -> bool.
  Axiom equal1 : forall x y : A, equal x y = true -> x = y.
  Axiom equal2 : forall x : A, equal x x <> false.

  Inductive List : Set :=
    | nullL : List
    | consL : A -> List -> List.

  Inductive MemL (a : A) : List -> Prop :=
    | hereL : forall x : List, MemL a (consL a x)
    | thereL : forall x : List, MemL a x -> forall b : A, MemL a (consL b x).

  Inductive isSet : List -> Prop := ...
```

2. Defina una función recursiva `deleteAll` que dado un elemento y una lista, elimine todas las ocurrencias del elemento de la lista.

3. Pruebe la siguiente propiedad:

```
Lemma DeleteAllNotMember : forall (l : List) (x : A),
   ~ MemL x (deleteAll x l).
```

4. Defina una función recursiva `delete` que dado un elemento y una lista, elimine la primera ocurrencia del elemento de la lista.

5. Pruebe la siguiente propiedad inherente a los conjuntos:

```
Lemma DeleteNotMember : forall (l : List) (x : A), isSet l ->
   ~ MemL x (delete x l).
```

```
End Ej5_3.
```

Ejercicio 5.4. Considere la siguiente definición de árboles binarios de elementos de un tipo genérico:

```
Section Ej5_4.

 Variable A : Set.

 Inductive AB: Set :=
   null : AB
   | cons: A -> AB-> AB -> AB.
```

a) Defina inductivamente la relación `Pertenece` de pertenencia de un elemento a un árbol binario de tipo `AB`.

b) Defina una función recursiva `Borrar` que dado un árbol binario `ab` de tipo `AB` y un elemento `x de tipo A` retorne el árbol sin aquellos subárboles de `ab` que tienen a x como raíz.

Asuma el siguiente parámetro para la igualdad de elementos de tipo `A`:

```
Parameter eqGen: A->A->bool.
```

c) Pruebe el lema:

```
Lemma BorrarNoPertenece: forall (x:AB) (a:A), ~(Pertenece a (Borrar a x)).
```

Asuma la siguiente propiedad referente a la igualdad de elementos de tipos `A`:

```
Axiom eqGen1: (x:A) ~(eqGen x x)=false.
```

d) Defina inductivamente la relación `SinRepetidos` que contiene a los árboles binarios sin elementos repetidos. El árbol vacío pertenece a la relación.

```
End Ej5_4.
```

Ejercicio 5.5. Considere la siguiente gramática para expresiones booleanas:

$$
\begin{aligned}
\text{Var} &:= 0 \mid 1 \mid 2 \mid \ldots \\
\text{bool} &:= \text{true} \mid \text{false} \\
\text{BE} &:= \text{Var} \\
&\mid \text{bool} \\
&\mid (\ \text{BE} \wedge \text{BE}\ ) \\
&\mid (\neg\ \text{BE}\ )
\end{aligned}
$$

1. Defina en Coq los tipos `Var` como el conjunto de naturales y `BoolExpr` para representar el lenguaje *BE*.

2. La memoria es una función que asocia un valor booleano a cada variable y se dispone de una función lookup que dada una memoria y una variable, devuelve el valor de la variable en la memoria; lookup: Memoria $\rightarrow$Var $\rightarrow$Valor. La notación (e, δ) $\succ$ w se lee "la expresión booleana e se evalúa al valor booleano w en la memoria δ".

   Definimos la relación de evaluación de expresiones booleanas en un estado de la memoria δ, donde el conjunto de valores es bool, de acuerdo a las siguientes reglas:

   Regla *evar*: (v, δ) $\succ$ (lookup δ v)

   Regla *eboolt*: (true, δ) $\succ$ true

   Regla *eboolf*: (false, δ) $\succ$ false

Regla *eandl*: Si $(e_1, \delta) \succ$ false entonces $((e_1 \wedge e_2), \delta) \succ$ false

Regla *eandr*: Si $(e_2, \delta) \succ$ false entonces $((e_1 \wedge e_2), \delta) \succ$ false

Regla *eandrl*: Si $(e_1, \delta) \succ$ true y $(e_2, \delta) \succ$ true entonces $((e_1 \wedge e_2), \delta) \succ$ true

Regla *enott*: Si $(e, \delta) \succ$ true entonces $(\neg e, \delta) \succ$ false

Regla *enotf*: Si $(e, \delta) \succ$ false entonces $(\neg e, \delta) \succ$ true

Defina en Coq los tipos `Valor`, `Memoria` y la relación `BEval` definida por las reglas anteriores.

3. Demuestre en Coq las siguientes propiedades:

   a. Para toda memoria δ la expresión true evaluada en la memoria δ nunca da el valor false.

   b. Si $(e_1, \delta) \succ$ true y si $(e_2, \delta) \succ w$ entonces $(e_1 \wedge e_2, \delta) \succ w$ cualesquiera sean δ, e_1, e_2 y w.

   c. Si $(e, \delta) \succ w_1$ y $(e, \delta) \succ w_2$ entonces $w_1 = w_2$, cualesquiera sean w_1 y w_2.

   d. Si $(e_1, \delta) \succ$ false entonces no se da que $(\neg(e_1 \wedge e_2), \delta) \succ$ true, cualesquiera sean e_1 y e_2.

4. Escriba en Coq una función `beval:Memoria->BoolExpr->Valor` que evalúe las expresiones booleanas por valor (o sea que evalúe todas las subexpresiones de una expresión).

5. Demuestre que la función definida en la parte anterior es una estrategia correcta de evaluación con respecto a la relación $\succ$. Es decir que $(e, \delta) \succ$ (beval δ e).

Ejercicio 5.6. Considere el siguiente mini-lenguaje imperativo definido por la gramática:

I : = **Skip**
| Var : = BE
| **IF** BE **Then** I **Else** I
| **While** BE **Do** I
| **Repeat** nat I
| **Begin** LI **End**

LI : = ε
| I **;** LI

1. Defina en Coq los tipos `Instr`, `LInstr` para representar la sintaxis abstracta de los programas (*I* y *LI* respectivamente).

2. Asocie una notación infija al operador de constructor de listas no vacías, de forma que sea asociativo a la derecha.

a. Considere el siguiente programa *PP* en el cual *v1* y *v2* son variables.

Begin

v1 := true ;

v2 := ¬v1

End

Escriba el término Coq correspondiente a la sintaxis abstracta del programa anterior utilizando la notación infija del constructor de listas definida en la parte anterior.

b. Considere el siguiente programa *swap* en el cual *v1*, *v2* y *aux* son variables.

Begin

aux := v1 ;

v1 := v2 ;

v2 := aux

End

Escriba el término Coq correspondiente a la sintaxis abstracta de *swap* utilizando la notación infija del constructor de listas definida en la parte anterior.

3. Defina una función updtate:Memoria→Var→Valor→Memoria tal que el resultado de (update δ v w) es una memoria igual a δ excepto por el valor de la variable v que ahora es w.

4. Pruebe que: lookup (update (δ, var, val), var) = val, cualesquiera sean var, val y δ.

5. Pruebe que: si var <> var' entonces lookup (update (δ, var, val), var') = lookup (δ, var'), cualesquiera sean var, var', val y δ.

Ejercicio 5.7. Considere la siguiente especificación de un intérprete de instrucciones para el minilenguaje imperativo definido en el ejercicio anterior. El resultado de la ejecución de un programa en un estado de la memoria, devuelve un nuevo estado de la memoria.

Regla *xAss*: Si (e, δ) ≻ w entonces (v: = e , δ) » (update δ v w)

Regla *xSkip*: (Skip, δ) » δ

Regla *xIFthen*: Si (c,δ) ≻ true y (p_1,δ) » δ_1 entonces (If c Then p_1 Else p_2, δ) » δ_1

Regla *xIFelse*: Si (c,δ) ≻ false y (p_2,δ) » δ_2 entonces (If c Then p_1 Else p_2, δ) » δ_2

Regla *xWhileTrue*: Si (c,δ) ≻ true y (p,δ » δ_1) y (While c Do p, δ_1) » δ_2 entonces (While c Do p, δ) » δ_2

Regla *xWhileFalse*: Si (c,δ) ≻ false entonces (While c Do p, δ) » δ

Regla *xRepeat0*: (Repeat 0 i, δ) » δ

Regla *xRepeatS*: Si (i, δ1) » δ2 y (Repeat n i, δ2) » δ3, (Repeat n+1 i, δ1) » δ3

Regla *xBeginEnd*: Si (p, δ) $»_L$ $δ_1$ entonces (Begin p End, δ) » $δ_1$

Regla *xEmptyblock*: (ε, δ) $»_L$ δ

Regla *xNext*: Si (i, δ) » $δ_1$ y (li, $δ_1$) $»_L$ $δ_2$ entonces (i; li, δ $»_L$ $δ_2$)

1. Defina en Coq la relación *Execute* que implemente la definición anterior.
2. Demuestre que si (If (¬ c) Then e_1 Else e_2, δ) » δ′ entonces (If c Then e_2 Else e_1, δ) » δ′, cualesquiera sean c, e_1, e_2, δ y δ′.
3. Demuestre que si (While false Do p, δ) » δ′ entonces δ = δ′, cualesquiera sean p, δ y δ′.
4. Demuestre que si ((If c Then p Else Skip); While c Do p, δ) » δ′ entonces (While c Do p, δ) » δ′, cualesquiera sean p, c, δ y δ′.
5. Demuestre que si (i;(Repeat n i), δ1) » δ2 entonces (Repeat n+1 i, δ1) » δ2, cualesquiera sean n, i, δ1 y δ2.
6. Demuestre que si (Repeat n1 i, δ1) » δ2 y (Repeat n2 i, δ2) » δ3 entonces (Repeat n1+n2 i, δ1) » δ3, cualesquiera sean n1, n2, i, δ1, δ2 y δ3.
7. Pruebe que para toda memoria δ, si *v1* y *v2* son variables distintas, la ejecución del programa *PP* del ejercicio anterior da como resultado una memoria en la cual *v2* tiene el valor *false* y *v1* tiene el valor *true*.

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.