Teoría de Tipos y *Coq* en la Enseñanza de Programación Funcional e Imperativa

Taller de Construcción Formal de Programas

Carlos Luna
cluna@fing.edu.uy, http://www.fing.edu.uy/~cluna
***InCo*, Fac. de Ingeniería, U. de la República, Uruguay**

Contexo

- Fundamentos de las Ciencias de la Computación.
  - Lógica matemática.
  - Matemática discreta.
- La propuesta:
  - Deducción lógica + Inducción matemática ⇒ Construcción formal de programas.
  - "Programar rigurosamente sobre la base de argumentos matemáticos".
  - ***Taller de construcción formal de programas.***

Contexo

- Demostración y Verificación de Corrección
  - Testing
  - Verificación de Corrección
  - Demostración de Corrección
    - *Coq* (entre otros)
      - Carácter unificador de la teoría que implementan.
      - El usuario es guiado en forma interactiva por el sistema en el proceso de construcción de un programa o una prueba.
      - El principal objetivo de estos sistemas es convertirse en sofisticadas herramientas que asistan en la tarea del desarrollo incremental de programas correctos.

Objetivos del curso

- Presentar a la teoría de tipos como lógica de programación y familiarizar al estudiante con ambientes de desarrollo de programas basados en este formalismo.
- Iniciar al estudiante en el uso de métodos formales para la especificación, producción, derivación y verificación de software correcto por construcción en los paradigmas de programación funcional e imperativo.

Objetivos del curso

- Iniciar al estudiante en el uso de métodos formales para la especificación y verificación de otras clases de sistemas. En particular, para sistemas críticos (sistemas reactivos y de tiempo real).
- Mostrar la utilidad de editores de pruebas basados en teoría de tipos para la especificación y verificación de aplicaciones industriales (no sólo académicas).

Contenidos del curso

1. Asistentes de pruebas para lógicos y matemáticos: Una presentación formal de la lógica proposicional y de primer orden.

2. Asistentes de Pruebas para programadores: El calculo lambda como lenguaje de programación funcional.

3. Identificación de pruebas y programas: Isomorfismo de Curry Howard

4. Recursión: Definiciones Inductivas, Principios de Inducción y Esquemas de Recursión.

Contenidos del curso

5. Extracción de programas a partir de pruebas. Construcción de Pruebas a partir de programas.

6. Construcción de programas certificados usando Coq: Programas Funcionales y Programas Imperativos.

7. Extensiones: una introducción al desarrollo de programas y pruebas con tipos recursivos que pueden contener objetos infinitos. Sistemas reactivos y de tiempo real.

Algunos datos del curso

- **Metodología de enseñanza:** taller
- **Conocimientos previos:**

lógica de primer orden y algún lenguaje de programación funcional

- **Contexto en un plan de estudios:**

curso de grado (electiva) o curso de postgrado

- **Bibliografía y material:**

http://coq.inria.fr/,

http://www.fing.edu.uy/inco/grupos/mf/TPPSF/

- **Modalidad de evaluación:**

entregas por unidad + parciales, proyecto y/o examen

- **Duración:** 160 hs

Coq

- Implementación del *CCI*
- Definiciones y demostraciones a la *DN*
- Interpretación computacional de las pruebas
  - "*t* es una demostración de la proposición *A*"
  - "el término t tiene tipo A"
  - "t es un programa de la especificación A"

  (*isomorfismo de Curry-Howard*)
- Prop y Set, Extracción de programas
- Tipos co-inductivos
- Uso de Tácticas

Coq

- ***Definición de tipos inductivos***

```
Inductive nat : Set := Co : nat | Sg : nat -> nat.

Inductive bintree [A:Set] : Set :=
   emptyb : (bintree A)
   | consb : A->(bintree A)->(bintree A)->(bintree A).
```

Coq

- ***Definición de relaciones inductivas***

```
Inductive mirror [A: Set]:
          (bintree A) -> (bintree A) -> Prop:=
empty: (mirror A (emptyb A) (emptyb A))
| notEmpty: ∀x∈A , ∀t1,t2,t3,t4∈(bintree A)
      (mirror A t1 t2) -> (mirror A t3 t4) ->
      (mirror A (consb A x t1 t3) (consb A x t4 t2)).
```

Coq

- ***Definición de funciones (recursivas)***

```
Fixpoint reverse [A:Set; b:(bintree A)]: (bintree A) :=
 Cases b of
    emptyb => (emptyb A)
    | (consb x b1 b2) =>
          (consb A x (reverse A b2) (reverse A b1))
 end.
```

Coq

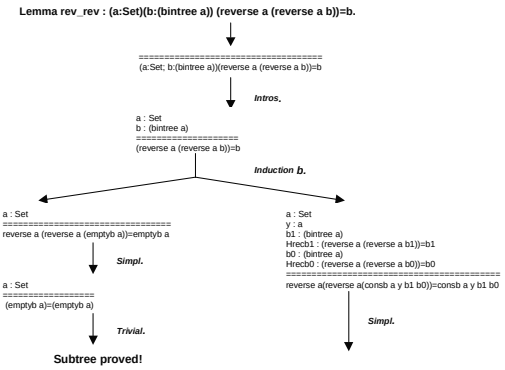
- ***Definición y prueba de propiedades***

Lemma rev_rev : (∀A∈Set) (∀b∈(bintree A))
(reverse A (reverse A b)) = b.

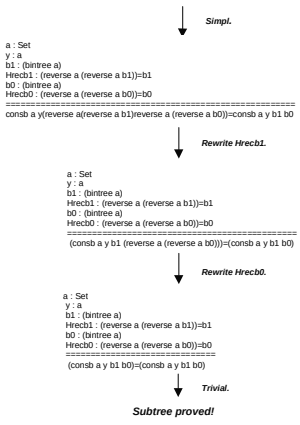
Una prueba del lema *rev_rev* en el Asistente *Coq* es la siguiente:

```
Proof.
   Intros. Induction b. Simpl. Trivial.
   Simpl. Rewrite Hrecb1. Rewrite Hrecb0. Trivial.
Qed.
```

Coq



Coq



Coq

- ***Construcción y verificación de programas funcionales***

Lemma ***mirror_inverse****: (t:bintree) { t':bintree | (mirror t t') }.*

…..

Coq > Write Haskell File "mirror_function" [mirror_inverse].

```
data Tree a = Nil | Cons a (Tree a) (Tree a)
inverse t =  case t of
                Nil -> Nil
                Cons a0 t1 t2 -> Cons a0 (inverse t2) (inverse t1)
mirror_inverse = inverse
```

Coq

- ***Especificación y verificación de programas imperativos***

```
Global Variable f,i,n:Z ref.
Correctness imperative_program
{`n>=0`}
begin
    f:=1; i:=1;
    while (!i < !n+1) do
        {invariant  (RFact `(i-1)` f) /\ `i<= n+1`
         variant `n-i+1`}
       f:=!f*!i; i:=!i+1
    done
end {(RFact n@0 f)}.
```

Coq

- ***Especificación y verificación de programas imperativos***

Dónde:

```
Inductive RFact : Z -> Z -> Prop :=
    f0: (RFact `0` `1`)
    | fs: ∀z,f∈Z (RFact z f) -> (RFact `z+1`  `f*(z+1)`).
```

Experiencias

- **InCo (Uruguay): 2000 a la actualidad**
- **UNRC (Argentina): Año 2001**
- **UNR: Años 2000, 2001 y 2005**
- **Escuelas de Ciencias Informáticas:**

  Río'2000, ECI'2001, Río'2004

- Más de 300 estudiantes (de grado/postgrado de distintos grupos).

- Cuatro tesis de maestría (y otras en curso) y varias tesinas (proyectos) de grado.

- Desde el año 2000 a la actualidad tres proyectos de investigación han sido llevados a cabo en temas relacionados y varias colaboraciones con instituciones regionales e internacionales han sido establecidas.

Conclusiones

- Propuesta para apoyar la enseñanza de métodos formales en una currícula de grado, usando el asistente de pruebas *Coq* y conceptos del área de *Teoría de Tipos*
- El taller permite fortalecer la noción de que junto con la construcción de los algoritmos existe la obligación de la verificación rigurosa (formal) de su corrección y que los programas son objetos matemáticos plausibles de ser tratados con argumentos lógico-matemáticos. *Coq* es una herramienta adecuada para asistir a los estudiantes en este proceso de aprendizaje.

Conclusiones

- El taller abarca la especificación, derivación y verificación de sistemas en los paradigmas de programación funcional e imperativo (reactivos y de tiempo real)
- Experiencias:

El taller nos permite integrar a docentes y ayudantes interesados en trabajar en la construcción formal y la verificación de sistemas de software.

El taller ofrece un marco adecuado para la realización de trabajos de investigación, trabajos finales en carreras de grado y trabajos de postgrado.

Conclusiones

- Posibles extensiones:
  - Análisis de sistemas críticos: sistemas reactivos y de tiempo real (tipos co-inductivos).
  - Desarrollo de casos de estudio.
  - Análisis de sistemas en el paradigma POO.