Lógica Computacional
Práctica 06: Introducción a la Verificación Formal

Dicter Tadeo García Rosas Erik Rangel Limón Manuel Soto Romero

Semestre 2024-2
Facultad de Ciencias UNAM

Objetivos

1. Familiarizar al alumno con el asistente de pruebas *Coq* mediante la implementación de funciones y la realización de demostraciones formales.

Instrucciones

1. Para esta práctica necesitarán tener instalado *Coq*.

   - **Debian** y **Fedora**:
     El paquete `coq` contiene el asistente de pruebas y el editor `coqide`, investiguen si ésto difiere según su distribución

     ```
     $ sudo apt install coq
     ```

     ```
     $ sudo dnf install coq
     ```

   - **Windows** y **Mac**:
     Hay paquetes binarios en *Github*:
     `https://github.com/coq/platform/releases/tag/2023.11.0`
   - De preferencia intenten instalarlo, pero si no les funciona, hay un editor en línea:
     `https://coq.vercel.app`

2. Resuelvan todos los ejercicios del archivo `src/Practica6.v`

   Su calificación será decidida manualmente, pero la cantidad de ejercicios que Coq puede verificar (sin usar `Admitted`) será su calificación.

Ejercicios

1. Demuestra: $(p \to q \to r) \Leftrightarrow (p \wedge q) \to r$

```
Lemma imp_conj :
  forall P Q R : Prop, (P -> Q -> R) <-> (P /\ Q -> R).
```

2. Demuestra: $(p \to q) \to (q \to r) \to (p \to r)$

```
Lemma imp_trans :
  forall P Q R : Prop, (P -> Q) -> (Q -> R) -> P -> R.
```

3. Demuestra $(p \vee q) \wedge \neg p \to q$

```
Lemma or_and_not :
  forall P Q : Prop, (P \/ Q) /\ ~P -> Q.
```

4. Demuestra $\neg(p \vee q) \Leftrightarrow (\neg p \wedge \neg q)$

```
Lemma deMorgan :
  forall P Q : Prop, ~(P \/ Q) <-> (~P /\ ~Q).
```

5. Demuestra $\forall n \in \mathbb{N}. n + 1 = S(n)$

```
Lemma suma_suc :
  forall n : nat, n + 1 = S n.
```

6. Demuestra $\forall n, m \in \mathbb{N}. n + m = m + n$

```
Lemma suma_conm :
  forall n m : nat, n + m = m + n.
```

7. Demuestra $\forall n, m, p \in \mathbb{N}. p \cdot (n + m) = p \cdot n + p \cdot m$

```
Lemma mult_dist :
  forall n m p : nat, p * (n + m) = p * n + p * m.
```

8. Implementa de manera recursiva una función que calcule la suma de todos los números naturales hasta n.

```
Fixpoint gauss (n : nat) : nat.
```

9. Demuestra que tu implementación de gauss cumple que $\forall n \in \mathbb{N}. 2 \cdot gauss(n) = n \cdot (n + 1)$

```
Lemma gauss_eq :
  forall n : nat, 2 * gauss n = n * (n + 1).
```

10. Demuestra que la operación de concatenación es asociativa en listas

```
Lemma concat_assoc :
  forall (A : Type) (xs ys zs : list A), xs ++ ys ++ zs = (xs ++ ys) ++ zs.
```

11. Usando la siguiente definición de reversa en Coq:

```
Fixpoint reversa {A : Type} (l : list A) :=
  match l with
  | nil => nil
  | cons x xs => reversa xs ++ (x :: nil)
  end.
```

Demuestra:

```
Lemma rev_concat :
  forall (A : Type) (xs ys : list A), reversa (xs ++ ys) = reversa ys ++ reversa xs.
```

12. Define la función *take*, que toma una lista xs, un número natural n, y regresa la lista con los primeros n elementos de xs.

```
Fixpoint take {A : Type} (n : nat) (l : List A) : list A.
```

13. Define la función *drop*, que toma una lista xs, un número natural n, y regresa la lista sin los primeros n elementos de xs.

```
Fixpoint drop {A : Type} (n : nat) (l : List A) : list A.
```

14. Demuestra:

```
Lemma tdn:
  forall (A : Type) (xs : list A) (l : list A), xs = take n xs ++ drop n xs.
```

15. Demuestra:

```
Lemma ts:
  forall (A : Type) (xs : list A) (n m : nat),
    take m (drop n xs) = drop n (take (m + n) xs).
```

16. Demuestra:

```
Lemma min:
  forall (A : Type) (xs : list A) (n m : nat),
    take m (take n xs) = take (min m n) xs.
```

Entrega

1. La tarea se entrega en *Github Classroom*.
2. La tarea se entrega en equipos de hasta 4 personas.
3. Además de resolver los ejercicios, deben entregar un `README` (`.md` o `.org`) en la raiz de la carpeta de la práctica que contenga los datos de todos los integrantes, y por cada una de sus funciones dejar una breve explicación de su solución.

Consideraciones

1. Todas las prácticas con copias totales o parciales tanto en el código como en el `README` serán evaluadas con cero.
2. Las únicas funciones con soluciones iguales admisibles son todas aquellas que sean iguales a las resueltas por el grupo en el laboratorio, sin embargo la explicación de su solución en el `README` debe ser única para cada equipo.
3. No entregar el `README` o tenerlo incompleto se penalizará con hasta dos puntos menos sobre su calificación en la práctica.
4. Cada día de retraso se penalizará con un punto sobre la calificación de la práctica.
5. No pueden importar paqueterías adicionales, no pueden usar funciones ni teoremas que resuelvan el ejercicio. Pregunten al ayudante de laboratorio por el uso de funciones o teoremas predefinidos que quieran usar.
6. No se recibirán prácticas que arrojen errores. Si no resuelven alguno de los ejercicios déjenlos como `Admitted`, pero no eliminen el ejercicio.
7. No deben modificar las firmas de las funciones ni de los lemas. Si encuentran errores o tienen dudas, manden un correo al ayudante de laboratorio (si encuentran errores tendrán una participación).
8. Las participaciones en el laboratorio se aplicarán de manera individual sobre la calificación que tuvieron en la práctica.