Construcción Formal de Programas en Teoría de Tipos

Segundo Parcial de 2015

NOTA: en el parcial pueden usarse tácticas automáticas y todo lo visto en el curso.

Problema 1.

a) Defina la función ***Elim***, que borre todas las ocurrencias de un elemento dado en una lista de naturales. Si el elemento no pertenece a la lista, la función deberá retornar la lista sin cambios.

b) Pruebe lemas para las siguientes propiedades (las variables están cuantificadas universalmente):

1. *Elim x (Elim x l) = Elim x l.*

2. *length (Elim x l) <= length l.*

Problema 2.

a) Defina inductivamente una relación binaria ***Prefijo*** entre listas de elementos de un tipo genérico, tal que una lista l1 se relaciona con una lista l2 si y sólo si l1 es un prefijo (segmento inicial) de l2. Toda lista es prefijo de si misma.

b) Pruebe lemas para las siguientes propiedades (las variables están cuantificadas universalmente):

1. Si l1 es un *Prefijo* de l2 y l2 es un *Prefijo* de l1 entonces l1 = l2.

2. La relación *Prefijo* es transitiva.

Problema 3.

Se quiere definir la sintaxis y semántica de un mini-lenguaje imperativo cuya gramática es:

I : = Var ← Valor

| I **;** I

| **IfEq** Var Var I

| **Repeat** nat I

(←) es la operación de asignación, (**;**) es la composición secuencial, (**If** *v1 v2 i*) ejecuta la instrucción *i* sii *n1=n2*, siendo *n1* el valor de la variable *v1* y *n2* el valor de *v2*, y finalmente (**Repeat** *n i*) es la operación que ejecuta *n* veces la instrucción *i*.

a) Defina inductivamente el tipo `Instr:Set` que representa la sintaxis abstracta de los programas (I), dónde:

```
Definition Var := nat.
Definition Valor := nat.
```

Considere la siguiente especificación de un intérprete de instrucciones para el mini-lenguaje imperativo definido en la parte a). El resultado de la ejecución de un programa en un estado de la memoria devuelve un nuevo estado de la memoria.

Regla *xAss*: (var ← val , δ) » (update δ var val)

Regla *xSeq*: Si (i_1, δ1) » δ2 y (i_2, δ2) » δ3 entonces (i_1;i_2, δ1 » δ3)

Regla *xIfF*: Si lookup (δ, v1) ≠ lookup (δ, v2) entonces (If v1v2 i, δ) » δ.

Regla *xIfT*: Si lookup (δ1, v1) = lookup (δ1, v2) y (i, δ1) » δ2 entonces (If v1v2 i, δ1) » δ2.

Regla *xRepO*: (Repeat 0 i, δ) » δ

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

b) Considerando, `Definition Memoria := Var -> Valor`, defina:

- `update: Memoria -> Var -> Valor -> Memoria`, dada una memoria m, una variable v y un valor w, `(update m v w)` representa la actualización de la memoria `m`, donde se asigna `w` a la variable `v`.
- `lookup: Memoria -> Var -> Valor`, retorna el valor de una variable de la memoria.

c) Defina en Coq la relación *Execute*, que especifique la ejecución del intérprete de instrucciones definido anteriormente.

d) Demuestre que:

- lookup (update (δ, var, val), var) = val., cualesquiera sean var, val y δ.
- Si var ≠ var' entonces lookup (update (δ, var, val), var') = lookup (δ, var'), cualesquiera sean var, var', val y δ.
- si lookup (v1, δ1) = lookup (v2, δ1) y (If v1v2 i, δ1) » δ2 entonces (Repeat 1 i, δ1) » δ2, cualesquiera sean v1, v2, i, δ1 y δ2.
- si (i;(Repeat n i), δ1) » δ2 entonces (Repeat n+1 i, δ1) » δ2, cualesquiera sean n, i, δ1 y δ2.
- 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.