Construcción Formal de Programas en Teoría de Tipos

Segundo Parcial de 2023

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

Problema 1.

a) Defina la función ***eliminar***, que elimina (eventualmente) la primera ocurrencia de un elemento dado en una lista de naturales.

b) Defina la función ***pertenece*** que, dada una lista de naturales y un elemento, retorna true si y solo si el elemento está en la lista.

c) Pruebe lemas para las siguientes propiedades:

1. *Lemma L1_1 : forall (A:Set) (l:list A) (x:A), l <> x::l.*

2. *Lemma L1_2: forall (l:list nat) (x:nat), pertenece x l = true -> eliminar x l <> l.*

Problema 2.

a) Defina inductivamente una relación binaria ***distintas*** entre listas de elementos de un tipo genérico, tal que una lista *l1* se relaciona con una lista *l2* si y sólo:

- Tienen igual largo (pudiendo ser 0).
- Todo elemento en la posición *i* de *l1* es distinto al elemento en la posición *i* de *l2*.

b) Pruebe el siguiente lema:

Lemma L2: forall (l1:list bool), {l2: list bool | distintas l1 l2}.

c) Extraiga del lema *L2* un programa Haskell que dada una lista de booleanos retorne la lista donde el valor de cada elemento sea el opuesto al de la lista parámetro.

Problema 3.

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

I : = Var ← Valor
| I **;** I
| **If** Var Valor I I

- (←) es la operación de asignación
- (**;**) es la composición secuencial
- (**If** *v val i1 i2*) ejecuta la instrucción *i1* si *n* = *val*, siendo *n* el valor de la variable *v*; en caso contrario, se ejecuta la instrucción *i2*.

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 *xIfT*: Si lookup (δ1, v) = val y (i1, δ1) » δ2 entonces (**If** v val i1 i2, δ1) » δ2.

Regla *xIfF*: Si lookup (δ1, v) ≠ val y (i2, δ1) » δ2 entonces (**If** v val i1 i2, δ1) » δ2.

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

- `lookup: Memoria -> Var -> Valor`, retorna el valor de una variable de la memoria.
- `update: Memoria -> Var -> Valor -> Memoria`, dada una memoria δ, una variable v y un valor val, actualiza δ asignando val a la variable v.

c) Defina en Coq la relación *Execute* que implemente el intérprete de instrucciones definido anteriormente.

d) Demuestre que:

- Si lookup (δ1,v1) ≠ val y (If v val i1 i2, δ1) » δ2 entonces (i2, δ1) » δ2,

  cualesquiera sean v, val, i1, i2, δ1 y δ2.

- Si v1 ≠ v2 y (v1 ← val; v2 ← val+1, δ1) » δ2 y (i2, δ2) » δ3 entonces

  (If v2 lookup(δ2, v1) i1 i2, δ2) » δ3, cualesquiera sean v1, v2, val, i1, i2, δ1, δ2 y δ3.