Verificación de Programas con F*

Clase 6 – 23/04/2024

Clase pasada: BSTs

- Los definimos mediante un simple tipo inductivo
- Pudimos demostrar algunas propiedades como:
  - forall x t. member x (insert x t)
  - Otras sobre tamaño y altura
- La propiedad `member x (insert y (insert x t))` puede demostrarse… pero requiere razonar sobre la forma del árbol
- La propiedad `delete x (insert x t) == t` no vale, dado que la *forma* del árbol puede cambiar. Dos BST pueden ser equivalentes sin ser iguales.

```
type bst =
  | L
  | N of bst & int & bst
```

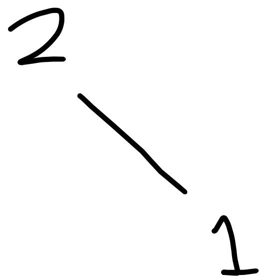
¿BSTs?

```
type bst =
  | L
  | N of bst & int & bst
```

- ¿Realmente estábamos verificando BSTs?
- ¿Se puede demostrar la propiedad siguiente?

```
let delete_not_mem (x y : int) (t : bst)
 : Lemma (requires not (member x t))
         (ensures (not (member x (delete y t)))) = …

let falso () : Lemma False =
  delete_not_mem 1 2 (N (L, 2, N (L, 1, L)))
```



- Todo lo que demostramos hasta ahora no hace uso del *invariante* de los BST (o no podríamos haberlo demostrado)

Refinando la estructura BSTs

- Una forma usual de trabajar con estructuras con invariantes es definir primero la versión “base”, y luego refinarla.
- Las funciones relevantes suelen definirse sobre la versión base, y luego se demuestra que preservan los invariantes.
- Las versiones refinadas de las funciones son operacionalmente iguales a las versiones base
  - Las pruebas y los refinamientos se borran

```
type bst0 =
  | L
  | N of bst0 & int & bst0

let rec all_lt (x: int) (t: bst0) : bool =
  match t with
  | L -> true
  | N (l, y, r) -> all_lt x l && y < x && all_lt x r

let rec all_gt (x: int) (t: bst0) : bool =
  match t with
  | L -> true
  | N (l, y, r) -> all_gt x l && y > x && all_gt x r

let rec is_bst (t: bst0) : bool =
  match t with
  | L -> true
  | N (l, x, r) -> is_bst l && is_bst r && all_lt x l && all_gt x r
```

```
type bst = b:bst0{is_bst b}
```

```
let rec insert0 (x: int) (t: bst0) : bst0 =
  ...

let rec insert0_bst (x:int) (t:bst0)
  : Lemma (requires is_bst t)
          (ensures is_bst (insert0 x t)) =
  ...

let insert (x:int) (t:bst) : bst =
  insert0_bst x t;
  insert0 x t
```

Tareas

- Clase6.*.fst: completar removiendo los admits/assume.