Introducción a la Verificación Formal

Clase 4 – 18/09/2025

- Repaso Clase 3

Tipos indexados

- Al matchear, *conseguimos* algunas igualdades en el contexto
- Si `e : expr ty`, al matchear e con `EAdd l r` debemos tener necesariamente `ty = Int`.
- En general no se define por completo, pero conseguimos ecuaciones útiles (cómo el caso del `EIf`).

```
type expr : l_ty -> Type =
  | EInt : int -> expr Int
  | EBool : bool -> expr Bool
  | EAdd : expr Int -> expr Int -> expr Int
  | EEq : expr Int -> expr Int -> expr Bool
  | EIf :
    #ty:_ ->
    expr Bool -> expr ty -> expr ty -> expr ty
```

```
let cast (#ty:l_ty) (e : expr ty{EAdd? e}) : expr Int =
  e
let eadd_es_int (#ty:l_ty) (e : expr ty{EAdd? e})
  : Lemma (ensures ty == Int) =
  ()
```

Igualdad

- Al igual que en otras teorías de tipos dependientes, la "igualdad" es un tema complicado. Hay al menos 3 versiones:
  - (=) Igualdad decidable/computable. Una función de tipo `a -> a -> bool` para "algunos" `a`, los tipos que soportan igualdad. F* usa un predicado hasEq para marcar cuáles tipos la soportan.

```
type eqtype = t:Type{hasEq t}
val (=) : #a:eqtype -> a -> a -> bool
```

    - Este diseño está motivado por la extracción a OCaml.
  - (==) Igualdad proposicional. Una *prueba* de que dos elementos `x,y` de un mismo tipo son iguales. La prueba se construye, en general, con ayuda del SMT solver. Al haber demostrado una igualdad `x==y`, estos dos términos son mutuamente reemplazables en cualquier contexto.
    - F* es *extensional*. Si tenemos `e : t1` y podemos *demostrar* `t1 == t2`, entonces `e : t2` (sin cast/coercion).
  - (===) Igualdad heterogénea, por ahora no la vemos

Lemas y pruebas extrínsecas

- Volvamos a esta definición de factorial (`int -> int`). ¿Podemos demostrar que el resultado es siempre positivo?
- Podemos usar un *Lema*, una función que retorna una prueba de algo. La forma general de un lema es:

  `Lemma (requires pre) (ensures post) (decreases …)`

  Si solo hay postcondición, se puede abreviar a:

  `Lemma post`
- Un lema garantiza la postcondición dada la precondición. Si la precondición no vale, el lema no puede ser “llamado”

```
let rec fac (x:int) : int =
  if x <= 0 then 1 else x * fac (x - 1)
```

```
let rec fac_is_pos (x:int) : Lemma (fac x > 0) =
  ..
```

```
let suma_fac (x y : int) : pos =
  fac_is_pos x;
  fac_is_pos y;
  fac x + fac y
```

Buscando la hipótesis de inducción

- Esto debería sonar familiar a EDyA2
  - (demo)
- Curry-Howard:
  - Prueba por inducción = función recursiva
  - Buena fundación = terminación/totalidad
  - Razonamiento circular = divergencia

Tareas

- Completar Clase04.*.fst
  - Vamos a seguir trabajando sobre el modulo de árboles binarios. Es importante que lo vayan completando a medida que avanzamos.
  - (por cierto: pueden usar `make` para verificar todo el directorio)
- Leer capítulo 8