Introducción a la Verificación Formal

Clase 9 – 20/11/2025

Lógica de Hoare: problemas

- $\{ x=X \}$ f(x) $\{ x=g(X) \}$ ==> $\{ x = X \land y = Y \}$ f(x) $\{ x = g(X) \land y = Y \}$
  - No… f podría modificar y.
  - Condición lateral: $x \notin Mod(f)$

- $\{ {*x} = 1 \land {*y} = 1 \}$ $*x = 2$ $\{ {*x} = 2 \land {*y} = 1\}$
  - No… x e y pueden ser el mismo puntero.

- { IsList(p1, [...]) ∧ IsList(p2, [...]) }
  freeList(p1)
  { IsList(p2, [...]) }
  - Necesitamos que **todos** los punteros de p1 no aparezcan en p2.

Lógica de Separación (Reynolds y O'Hearn 2002)

- *"Una lógica para estructuras mutables compartidas"*
- El ejemplo motivador es revertir una lista in-situ:

```
Reverse(A):
 B = nil
 while (A != nil):
  tmp = A->next
  A->next = B
  B = A
  A = tmp
```

- ¿Por qué es correcto?
  - Invariante: rev(A) ++ B == rev(A0)
  - Pero, A y B no pueden compartir punteros. Ni con ninguna otra lista que ande por ahí, si queremos demostrar que no cambia.
  - Es el caso usual… la mayoría de las estructuras en un programa no comparten punteros.

Conjunción separante

- En vez de ∧, Reynolds introduce la conjunción separante *
  - Un *heap* h satisface **p * q** cuando puede particionarse en **h1/h2** tal que **p(h1)** y **q(h2)**
  - { x -> 1 * y -> 1 } x = 2 { x -> 2 * y -> 1 } es trivial, dado que x <> y
  - { x -> 1 * x -> 2 } es imposible
  - Si vale { IsList(p, l) * IsList(q, m) }, las listas p y q son totalmente disjuntas.
- Las especificaciones son modulares, y *deben* mencionar los recursos accedidos
  - Esto implica que no hay condiciones como x ∉ Mod(C)

Frame rule

- Dado que las precondiciones mencionan todos los recursos accedidos, y la separación que provee la *, tenemos:

$$\{ p \} C \{ q \} \implies \{ p * r \} C \{ q * r \}$$

  - (Si la lógica también tiene /\, hace falta otra condición. Pulse no lo tiene.)
- La frame rule permite componer pruebas de correctitude de forma modular.
  - Es similar a la regla de Constancia en lógica de Hoare, pero sin condiciones laterales.

Reglas básicas

Modificación: { x -> _ } x := e { x -> [e] }

Alocación: { emp } x := alloc() { x -> _ }

Liberar: { x -> _ } free(x) { emp }

- Las aserciones sobre el heap son *recursos*
  - P **no** es lo mismo que P * P

Demo Pulse

Tareas

- Leer capítulos de Pulse en el libro de F* 31-37
- El paper de Reynolds está bueno: seplogic.pdf
- Más tarde subo algunos ejercicios