

Verificación de Programas con F^*

Clase 7 – 30/04/2024

Repaso clase pasada: BSTs

Lógica de Hoare

- La [Lógica de Hoare \(1969\)](#) es una lógica axiomática para la verificación de programas.
 - La versión original es para un pequeño lenguaje imperativo (variables, asignación, if, while)
 - Muchas versiones para lenguajes más expresivos (incluso en tipos dependientes: HTT, F*)
- La noción principal es la *tripleta*: $\{P\} S \{Q\}$, donde
 - P y Q son condiciones o propiedades sobre el estado global ($St \rightarrow Bool$)
 - P es la “pre”condición, Q la “post”condición
 - S es un programa
- La lógica de Hoare trae reglas para verificar los programas, en general una por conectivo

Lógica de Hoare – Un adelantado

At present, the method which a programmer uses to convince himself of the correctness of his program is to try it out in particular cases and to modify it if the results produced do not correspond to his intentions. After he has found a reasonably wide variety of example cases on which the program seems to work, he believes that it will always work. The time spent in this program testing is often more than half the time spent on the entire programming project; and with a realistic costing of machine time, two thirds (or more) of the cost of the project is involved in removing errors during this phase.


- C.A.R. Hoare, ~~la semana pasada~~ 1969

Lógica de Hoare - Reglas

$e ::= \text{var} \mid c \mid e + e \mid e - e \mid e * e \mid e = e \mid \dots$

$s ::= \text{skip} \mid x := e \mid s; s \mid \text{if } e \text{ then } s \text{ else } s \mid \text{while } e \text{ } s$

La propiedad P debe valer en el estado actual, modificado para que x tenga el valor de la expression e.



H-SKIP

$$\frac{}{\{P\} \text{ skip } \{P\}}$$

H-ASSIGN

$$\frac{}{\{\lambda s. P(s \oplus (x, \llbracket e \rrbracket))\} x := e \{P\}}$$

H-SEQ

$$\frac{\{P\} S_1 \{Q\} \quad \{Q\} S_2 \{R\}}{\{P\} S_1; S_2 \{R\}}$$

$(f \oplus (x,y)) = \text{fun } z \rightarrow \text{if } z = x \text{ then } y \text{ else } f z$

Lógica de Hoare - Reglas

$e ::= \text{var} \mid c \mid e + e \mid e - e \mid e * e \mid e = e \mid \dots$

$s ::= \text{skip} \mid x := e \mid s; s \mid \text{if } e \text{ then } s \text{ else } s \mid \text{while } e \text{ s}$

H-IF

$$\frac{\{P \wedge [e] = \text{true}\} T \{Q\} \quad \{P \wedge [e] = \text{false}\} E \{Q\}}{\{P\} \text{ if } C \text{ then } T \text{ else } E \{Q\}}$$

$(f \oplus (x,y)) = \text{fun } z \rightarrow \text{if } z = x \text{ then } y \text{ else } f z$

Lógica de Hoare – Bucles e invariantes

$e ::= \text{var} \mid c \mid e + e \mid e - e \mid e * e \mid e = e \mid \dots$

$s ::= \text{skip} \mid x := e \mid s; s \mid \text{if } e \text{ then } s \text{ else } s \mid \text{while } e \text{ s}$

H-WHILE

$$\frac{\{Inv \wedge \llbracket e \rrbracket = \text{true}\} B \{Inv\}}{\{Inv\} \text{ while } (C) \{B\} \{Inv \text{ s} \wedge \llbracket e \rrbracket = \text{false}\}}$$

Las reglas anteriores son (casi) completamente mecánicas. Pero, para verificar un bucle, debemos encontrar un invariante adecuado para conseguir la poscondición que nos interesa. Conseguir un invariante adecuado es indecidible en general.

$(f \oplus (x,y)) = \text{fun } z \rightarrow \text{if } z = x \text{ then } y \text{ else } f z$

Lógica de Hoare – En F*

Vamos a trabajar con un lenguaje Imp un poco más interesante que en prácticas anteriores.

Vamos a definir una lógica de Hoare y **demostrar que es correcta**.

En particular, tiene bucles posiblemente infinitos.

La evaluación **no** puede dars como una función total, ahora es una relación.

```
type var = string

type expr =
  | Var    : var -> expr
  | Const : int  -> expr
  | Plus  : expr -> expr -> expr
  | Minus : expr -> expr -> expr
  | Times : expr -> expr -> expr
  | Eq    : expr -> expr -> expr

type stmt =
  | Assign : var -> expr -> stmt
  | IfZ    : expr -> stmt -> stmt -> stmt
  | Seq    : stmt -> stmt -> stmt
  | Skip   : stmt
  | While  : expr -> stmt -> stmt
```


Lógica de Hoare – En F*

```
(* Relación de evaluación big step. *)
noeq
type runsto : (p:stmt) -> (s0:state) -> (s1:state) -> Type0 =
  | R_Skip : s:state -> runsto Skip s s
  | R_Assign : s:state -> #x:var -> #e:expr -> runsto (Assign x e) s (override s x (eval_expr s e))

  | R_Seq :
    #p:stmt -> #q:stmt ->
    #s:state -> #t:state -> #u:state ->
    runsto p s t -> runsto q t u -> runsto (Seq p q) s u
```

Tareas

Completar Clase7.Imp.fst

- Completar reglas de Lógica de Hoare
- Demostrar su correctitud.