Clase 10- verificación

martes, 13 de mayo de 2025 19:00

Verificación--> formal
Validación--> informal

Es mejor ir verificando a la vez que programamos.

Prueba semántica:
Ir escribiendo las instrucciones del programa con como va quedando cada variabe

Forma sintáctica: no me importa que hace el programa, qué es cada instrucción.
Axioma de asignación. El último regla de secuencia

1) Por la vía **semántica**, usando la semántica de las instrucciones del lenguaje.

	variables	*programa*
1)	x = X, y = Y	**z := x** ; x := y ; y := z
2)	x = X, y = Y, **z = X**	**x := y** ; y := z
3)	**x = Y**, y = Y, z = X	**y := z**
4)	x = Y, **y = X**, z = X	

2) Por la vía **sintáctica**, usando axiomas y reglas de un método deductivo.

1) $\{x = X \wedge y = Y\}$ **z := x** $\{z = X \wedge y = Y\}$ ASI
2) $\{z = X \wedge y = Y\}$ **x := y** $\{z = X \wedge x = Y\}$ ASI
3) $\{z = X \wedge x = Y\}$ **y := z** $\{y = X \wedge x = Y\}$ ASI
4) $\{x = X \wedge y = Y\}$ **z := x ; x := y ; y := z** $\{y = X \wedge x = Y\}$ SEC 1,2,3

Método axiomático debe ser sensato: que lo que pruebo sintácticamente valga también semánticamente.. Que no diga boludeces
Completo: que pruebe todos los enunciados verdaderos.

- Volviendo al ejemplo del programa de swap:
  - $\{x = X \wedge y = Y\}$ S_{swap} $\{y = X \wedge x = Y\}$ es una **terna de Hoare** o **fórmula de correctitud**.
  - El predicado $x = X \wedge y = Y$ es la **precondición** de S_{swap}.
  - El predicado $y = X \wedge x = Y$ es la **postcondición** de S_{swap}.
  - El par $(x = X \wedge y = Y, y = X \wedge x = Y)$ es la **especificación** de S_{swap}.

Especificación es pre +post condición

Estado: en un determinado momento qué tiene cada variable.
Un estado saisface un predicado si al evaluar el preducado con el estado es verdadero.

- Un **estado** σ es una función que asigna a toda variable un valor. Por ejemplo: σ(x) = 1, σ(y) = 2, etc.
- Un estado σ **satisface** un predicado p, si p evaluado con σ es verdadero. Se expresa así: **σ |= p**.

Por ejemplo: si σ(x) = 1 y σ(y) = 2, entonces σ |= x < y.

Un programa es correcto respecto de la especificación si para todo estado sigma si al comienzo s vale p después vale q.

- Un programa S es **correcto con respecto a una especificación** (p, q), lo que se expresa con **{p} S {q}**, sii:

Para todo estado σ, si σ |= p entonces S ejecutado a partir de σ termina en un estado σ´ tal que σ´ |= q

|= significa satisface

¿Se cumple {p} S {q} si desde σ0 , S no alcanza un σ´0 dentro de q? Si: si σ |≠ p, {p} S {q} es trivialmente verdadera esto no lo entendi y el profe dijo que era importante c:

Un estado es un conjunto de variables con sus valores. Mi pre condicion puede no tomar todas mis variables, en cuyo caso me chuparia un huevo el valor. Onda, solamente me importa el reultado de las variables que estoy verifiando. No impongo necesaariamente una condición para todas las variables.

Método de verificación axiomatica
Lero lero

3. Axiomatica

1. Axioma de la asignación (ASI)

$\{p(e)\}\ x := e\ \{p(x)\}$

Si luego de x := e vale p para x, entonces antes de x := e valía p para e.
Por ejemplo: {y > 0} x := y {x > 0}

Si lo de la derecha vale despu´se de la asignación entonces lo de la izquierda valía
Ejercicio: {?} x := x + 1 {x > 0}

```
{ ? } x := x + 1 { x ≥ 0 }
```

Es decir: ¿qué condición debe cumplirse **antes** de ejecutar `x := x + 1` para que, **después**, `x ≥ 0`?

🔧 Axioma de asignación (ASI)

Este axioma dice:

```
{ q[x | e] } x := e { q }
```

Se lee así:

> Si queremos que **después** de `x := e` se cumpla `q`, entonces **antes** de la asignación tiene que cumplirse `q` con `x` reemplazado por `e`.

3. Entonces, la precondición que necesitamos es:

```
{ x + 1 ≥ 0 } x := x + 1 { x ≥ 0 }
```

Va x+1

2. Regla de la secuencia (SEC)

$$\frac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1\ ;\ S_2\ \{q\}}$$

Descartar el puente

3. Regla del condicional (COND)

$$\frac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi}\ \{q\}}$$

No entendi este para nada :D

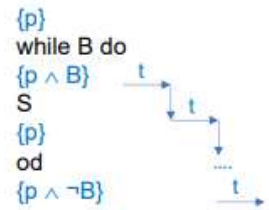
4. Regla de la repetición (REP)

$$\frac{\{p \wedge B\}\ S\ \{p\},\ \{p \wedge B \wedge t = Z\}\ S\ \{t < Z\},\ p \rightarrow t \geq 0}{\{p\}\ \text{while } B \text{ do } S \text{ od}\ \{p \wedge \neg B\}}$$

Es un while
P invariante, t te asegura que el while termine
t es la condición de corte del while
Pre condicion es vale p, vale b y t vale z.
La función tiene que ir decrementando ese t que vale z.

Si S que es un programa a partir de p y b mantiene p (invariante), y s termina a partir de p y b y decrementa

algo y además p nunca pasa de 0



P vale siempre, cuando termina el while es cuando no B

Ejemplos 4 y 5 revisar

Composicionalidad:
Método composicional si no importa el contenido. Es una caja negra.

Por ejemplo, dado el programa S :: S_1 ; S_2, si se cumplen las fórmulas: **{p} S_1 {r} y {r} S_2 {q}**,
también se cumple la fórmula: **{p} S_1 ; S_2 {q}**,
independientemente del contenido de S_1 y S_2.

Más aún, si en lugar de **S_2** utilizamos un subprograma **S_3** que también satisface la fórmula: **{r} S_3 {q}**,
entonces también se cumple la fórmula: **{p} S_1 ; S_3 {q}**,
lo que significa que S_2 y S_3 son **intercambiables** (son funcionalmente equivalentes respecto de (r, q)).

Siempre y cuando la precondición sea la misma, podes intercambiar los bloques
En los programas concurrentes perdes la composicionalidad

especificaciones

Especificaciones

- Hemos especificado un programa para calcular el factorial de la siguiente forma:

(x > 0, y = x!)

¿Pero es correcta la especificación?

No. Por ejemplo, el programa **S :: x := 1 ; y := 1** satisface (x > 0, y = x!) pero no es el programa pedido:
Por ejemplo, si al inicio x = 5, entonces al final debe ser y = 5! = 120.

- Lo que sucede es que **las variables de la precondición pueden modificarse a lo largo del programa**.

- Lo que se hace es utilizar **variables lógicas**, para **congelar valores**. En ej ejemplo considerado haríamos:

(x = X ∧ X > 0, y = X!)

¿Y se puede agregar a la especificación que la variable x no se modifique nunca?

No, la lógica de predicados no lo permite. Una lógica que sí lo permite es la **lógica temporal**.

Esto parece importante y no lo entendi

Sensatez:
Si se cumple sintácticamente, se cumple semánticamente también? Esto es obligatorio

Completitud: la inversa, si se cumple semánticamente que se cumpla sintácticamente tambien. Esto es deseable.
Que se cumpla depende de como expreso eso (creo, chequear esto)

La verificación de programas es indecidible, no se puede automatizar.

Lógica temporal: para espeificar propiedades a lo largo del tiempo.

1. $\sigma_0 \models Xp$ significa que en el estado siguiente de σ_0 vale p.
2. $\sigma_0 \models Gp$ significa que a partir de σ_0 siempre vale p.
3. $\sigma_0 \models Fp$ significa que en algún estado siguiente de σ_0 vale p.
4. $\sigma_0 \models p\ U\ q$ significa que a partir de σ_0 vale repetidamente p y en algún momento vale q (p puede o no seguir valiendo).