TD3: Algoritmos y Estructuras de Datos

Prof. Agustín Garassino, Gervasio Pérez

Segundo Semestre de 2024

Clase Teórica 4
Corrección de programas y
especificación de ciclos

Resumen

En la clase de hoy veremos:

- Concepto de estado
- Triplas de Hoare
- Pruebas de corrección
- Especificación y verificación de ciclos
- Ejercicios

Corrección de un programa

Dado un contrato que especifica una **Pre**condición y una **Post**condición, nos interesa probar formalmente que un programa S propuesto es correcto con respecto a ese contrato.

Es decir, queremos probar que siempre que las entradas cumplan con la **Pre**, la ejecución de nuestro programa llegará a un *estado* que cumple la **Post**.
En esta clase veremos herramientas lógicas para poder demostrar esa propiedad.

Lenguaje: C++ simplificado

Por simplicidad, definiremos un subconjunto reducido de C++.
Demostraremos corrección de programas en este lenguaje.

- Tipos: **int**, **bool**, **char**, **float**, vector<T>.
  El programa se asume correctamente tipado.
- Operaciones de vector<T>:
  - **v.size()**, **v[i]**, **==**, **!=**.
  - Inicializar con tamaño N: **v = vector<T>(N)**
  - No se permite modificar el tamaño.
- Las variables se asumen todas definidas
- Condicionales: Sólo **if-then-else**; sin **switch** ni operador ternario **(?:)**
- Ciclos: Sólo **while** ; sin **break** ni **continue**
- No hay **return expr**, en su lugar vamos a asignar valores a una variable especial **res** y su valor en el estado final será el valor de retorno.

Triplas de Hoare

Una **Tripla de Hoare** consta de una **Precondición** P, una **Postcondición** Q y de un programa S, y se escribe:

$$\{P\} \quad S \quad \{Q\}$$

y será Verdadera si, para todo estado del programa que satisface P, la ejecución de S siempre lo deja en un estado que satisface Q.

Por ejemplo, las siguientes triplas de Hoare

$\{i = 3\}$ `j = i; j = j*i; j = j*j;` $\{i = 3 \wedge j = 3^4\}$

$\{i = a\}$ `j = i; j = j*i; j = j*j;` $\{i = a \wedge j = a^4\}$

son triplas de Hoare válidas.

Triplas de Hoare y corrección de programas

Un programa S con una precondición **Pre** y una postcondición **Post** será correcto con respecto a éstas si la tripla de Hoare

$$\{Pre\} \quad S \quad \{Post\}$$

es verdadera. Pero... ¿cómo hacemos esta prueba?

Una metodología es establecer un **camino** entre la **Pre** y la **Post** en base a la secuencia de instrucciones $S_1 \ldots S_n$ del programa:

$$\{E_0\} \quad S_1 \quad \{E_1\} \quad S_2 \quad \{E_2\} \quad \ldots \quad \{E_{n-1}\} \quad S_n \quad \{E_n\}$$

de manera que $Pre \Longrightarrow E_0, E_n \Longrightarrow Post$, y cada una de las triplas

$$\{E_i\} \ S_{i+1} \{E_{i+1}\}$$

sea verdadera.¡Ahora sólo necesitamos un mecanismo para calcular los E_i intermedios!

Estados concretos de un programa

Llamamos **estado concreto de un programa** al conjunto de valores de cada una de sus variables en algún punto de su ejecución.

$$\{i = 5,\ j = 11\}$$

Dado un estado concreto inicial, y una instrucción, podemos calcular el esado concreto final aplicando la definición de esa instrucción:

$$E_0 \equiv \{i = 5,\ j = 11\}$$

```
i = i + j;
```

$$E_1 \equiv \{i = 16,\ j = 11\}$$

Verificación concreta (similar a testing)
Calcular la secuencia de estados que atraviesa un programa desde un estado inicial que cumple con la **Pre** especificada, y verificar que el estado final cumple **Post** especificada.

Ejemplo

Dada la siguiente especificación:

Evalúa una cierta fórmula matemática

```
float f(float a, float b)
```

Pre: $a \neq b$

Post: $res = \frac{ab}{|a-b|}$

y dado el siguiente algoritmo que provee una solución:

```
res = a * b;
if (a < b) {
    res = res / (b-a);
}
else {
    res = res / (a-b);
}
```

Queremos validar su corrección.

Validación concreta: Ejemplo

```
float f(float a, float b)
```
Pre: $a \neq b$

Post: $res = \frac{ab}{|a-b|}$

$E_0 : \{a = -2,\ b = 3\}$ consistente con *Pre* ya que $-2 \neq 3$
```
res = a * b;
```
$E_1 : \{a = -2,\ b = 3,\ res = -6\}$
```
if (a < b)
```
$E_2 : \{a = -2,\ b = 3,\ res = -6\}$
```
    res = res / (b-a);
```
$E_3 : \{a = -2,\ b = 3,\ res = -\frac{6}{5}\}$
```
else
    res = res / (a-b);
```
$E_4 : \{a = -2,\ b = 3,\ res = -\frac{6}{5}\}$ implica *Post* (justificar)

Ejercicio: hacer el seguimiento para $\{a = 3,\ b = 2\}$

Estado simbólico de un programa

Para razonar sobre *múltiples estados concretos* escribimos **predicados** que expresan propiedades que cumple un **conjunto de estados concretos** del programa:

$$E : \{a \neq b\}$$

E representa al **conjunto de estados concretos** donde vale que $a \neq b$:

$$\{a = -2, b = 3\}, \{a = 3, b = 2\}, \{a = 57, b = 42\}, ...etc$$

A este conjunto lo llamamos **estado simbólico**.

También podemos calcular cambios de estado simbólicos:

$$E_0 \equiv \{a > b\}$$

```
a = a - b;
```

$$E_1 \equiv \{a > 0\}$$

Validación simbólica: Ejemplo

```
float f(float a, float b)
```
Pre: $a \neq b$

Post: $res = \frac{ab}{|a-b|}$

$E_0 : \{a \neq b\}$ (equivalente a *Pre*)
```
 res = a * b;
```
$E_1 : \{a \neq b \wedge res = ab\}$
```
 if (a < b)
```
$E_{2t} : \{a < b \wedge res = ab\}$
```
      res = res / (b-a);
```
$E_{3t} : \{a < b \wedge res = \frac{ab}{b-a}\}$
```
 else
```
$E_{2f} : \{a > b \wedge res = ab\}$
```
      res = res / (a-b);
```
$E_{3f} : \{a > b \wedge res = \frac{ab}{a-b}\}$

$E_4 \equiv E_{3t} \vee E_{3f}$ *[uso x>y $\implies$ x-y = |x-y|, |x-y| = |y-x|]* $\implies$ *Post* ✓

Metodología de verificación TD3

Podemos verificar que, partiendo de un input que cumple con la **Pre**, llegamos a un output que satisface la **Post**:

1. Verificación concreta: si partimos de un estado **concreto**, estamos haciendo algo análogo a correr un caso de test.
2. Verificación simbólica: si partimos de un estado **simbólico** estamos probando la corrección general del programa.

Existen herramientas teóricas para realizar (2) y así demostrar la corrección teórica de un programa.

Ejercicio

Dada la siguiente especificación:

Intercambia dos posiciones si el contenido de la primera es mayor que el de la segunda

```
void intercambiar(vector<int>& v, int i, int j)
```

Pre: $v = v_0 \wedge 0 \leq i < j < |v|$

Post: $v[i] = min(v_0[i], v_0[j]) \wedge v[j] = max(v_0[i], v_0[j]) \wedge$
$(\forall k : \mathtt{int})\ 0 \leq k < |v| \wedge k \neq i \wedge k \neq j \implies v[k] = v_0[k]$

Verificar si las siguiente implementación es correcta según la especificación dada arriba.

```
if (v[i] > v[j]) {
    tmp = v[j];
    v[j] = v[i];
    v[i] = tmp;
}
```

Ejercicio – concreto

$E_0 \equiv \{v = \{1,5,3,4,5\},\ i = 1,\ j = 3\}$

```
if (v[i] > v[j]) {
```

$E_1 \equiv \{v = \{1,5,3,4,5\},\ i = 1,\ j = 3\}$

```
  tmp = v[j];
```

$E_2 \equiv \{v = \{1,5,3,4,5\},\ i = 1,\ j = 3,\ tmp = 4\}$

```
  v[j] = v[i];
```

$E_3 \equiv \{v = \{1,5,3,5,5\},\ i = 1,\ j = 3,\ tmp = 4\}$

```
  v[i] = tmp;
```

$E_4 \equiv \{v = \{1,4,3,5,5\},\ i = 1,\ j = 3,\ tmp = 4\}$

```
}
```

$E_5 \equiv \{v = \{1,4,3,5,5\},\ i = 1,\ j = 3,\ tmp = 4\}$

El estado E_5 cumple con la **Post** pues

- $v[i] = 4 = min(5,4) = min(v_0[i], v_0[j])$
- $v[j] = 5 = max(5,4) = max(v_0[i], v_0[j])$
- para $i = 0,2,4$ $v[i] = v_0[i]$

Ejercicio – simbólico

$E_0 \equiv \{v = v_0 \land 0 \leq i < j < |v|\}$

```
if (v[i] > v[j]) {
```

$E_1 \equiv \{v = v_0 \land 0 \leq i < j < |v| \land v_0[i] > v_0[j]\}$

```
  tmp = v[j];
```

$E_2 \equiv \{v = v_0 \land 0 \leq i < j < |v| \land v_0[i] > v_0[j] \land tmp = v_0[j]\}$

```
  v[j] = v[i];
```

$E_3 \equiv \{0 \leq i < j < |v| \land v_0[i] > v_0[j] \land tmp = v_0[j] \land v[j] = v_0[i]$
$\land |v| = |v_0| \land (\forall k : \texttt{int})\ 0 \leq k < |v| \land k \neq j \implies v[k] = v_0[k]\}$

```
  v[i] = tmp;
```

$E_4 \equiv \{0 \leq i < j < |v| \land v_0[i] > v_0[j] \land tmp = v_0[j] \land v[j] = v_0[i]$
$\land v[i] = v_0[j] \land |v| = |v_0| \land (\forall k : \texttt{int}) 0 \leq k < |v| \land k \neq j \land k \neq i \implies v[k] = v_0[k]\}$

```
 } { // else vacio
```

$E_{1\texttt{false}} \equiv \{v = v_0 \land 0 \leq i < j < |v| \land v_0[i] \leq v_0[j]\}$ **}**

$E_5 \equiv E_4 \lor E_{1F}$ ...que cumple con la Post porque:

- $v[i] = min(v_0[i], v_0[j]) \land v[j] = max(v_0[i], v_0[j])$ porque o bien vale
  - Por $E_{1\texttt{false}}$: $v_0[i] < v_0[j] \land v[i] = v_0[i] \land v[j] = v_0[j]$
  - Por E_4: $v_0[i] > v_0[j] \land v[i] = v_0[j] \land v[j] = v_0[i]$
- además, por E_4 y por $E_{1\texttt{false}}$, para $k \neq i, j$ $v[k] = v_0[k]$

Corrección de programas con ciclos

```
int suma_y_duplica(vector<int> v, int k)
```

Pre: Verdadero

S_1

```
res = k;
i = 0;
```

Pc (precondición del ciclo)

S_2

```
while(i < v.size()){
    res = res + v[i];
    i = i + 1;
}
```

Qc (poscondición del ciclo)

S_3

```
res = res * 2;
```

Post: $res = (k + \sum_{j=0}^{|v|-1} v[j]) * 2$

Para demostrar $\{Pre\}\ \ S\ \ \{Post\}$ necesitamos probar que valen las triplas:

- $\{Pre\}\ \ S_1\ \ \{Pc\}$,
- $\{Pc\}\ \ S_2\ \ \{Qc\}$,
- $\{Qc\}\ \ S_3\ \ \{Post\}$

Corrección de programas con ciclos

Queremos demostrar que vale

$$\{P_c\} \;\; while(B)\; S_c; \;\; \{Q_c\}$$

Donde:

- B es la guarda (condición) del ciclo, y
- S_c es el cuerpo del ciclo.

Debemos demostrar:

- **Corrección parcial**
  Si el ciclo termina, el estado final cumple Q_c.
- **Terminación**
  Luego de una cantidad finita de iteraciones la condición B se vuelve falsa y el ciclo termina.

Corrección parcial

Para demostrar **corrección parcial** de

$$\{P_c\} \;\; while(B)\; S_c; \;\; \{Q_c\}$$

necesitamos proponer un <u>invariante del ciclo</u> $\mathscr{I}$ y demostrar que:

- $P_c \implies \mathscr{I}$
  El invariante vale antes de entrar al ciclo
- $\{\mathscr{I} \wedge B = \texttt{true}\} \;\; S_c \;\; \{\mathscr{I}\}$
  El cuerpo del ciclo preserva la validez del invariante
- $(\mathscr{I} \wedge B = \texttt{false}) \implies Q_c$
  Al salir del ciclo vale la poscondición Q_c.

Invariante de ciclo

$P_c \equiv (i = 0 \land res = 0 \land n \geq 0)$

```
while(i <= n){
    res = res + i;
    i = i + 1;
}
```

$Q_c \equiv (res = \sum_{j=0}^{n} j)$

Ejemplo con $n = 6$:

i	res	n
0	0	6
1	0 + 0	6
2	0 + 0 + 1	6
3	0 + 0 + 1 + 2	6
4	0 + 0 + 1 + 2 + 3	6
5	0 + 0 + 1 + 2 + 3 + 4	6
6	0 + 0 + 1 + 2 + 3 + 4 + 5	6
7	0 + 0 + 1 + 2 + 3 + 4 + 5 + 6	6

Invariante del ciclo:

$$(res = \sum_{j=0}^{i-1} j) \land 0 \leq i \leq n+1$$

"**res** almacena la suma parcial hasta **i** sin incluir, y **i** está en rango"

Invariante de ciclo

$$\mathscr{I} \equiv (res = \sum_{j=0}^{i-1} j) \wedge 0 \leq i \leq n+1$$

- El invariante del ciclo expresa cuál es el trabajo computacional que realiza el ciclo en cada iteración.
- Expresa el cómputo "parcial" realizado por el ciclo luego de cada iteración incluso antes de finalizar todas las iteraciones.

Un buen invariante de ciclo suele tener:

- El valor de las variables que se modifican en cada iteración.
  - Un caso particular importante es una variable en la que vamos acumulando un resultado.
- El rango de valores entre los que se mueve la variable que itera.

Corrección parcial - Ejemplo

$P_c \equiv (i = 0 \land res = 0 \land n \geq 0)$

```
while(i <= n){
    res = res + i;
    i = i + 1;
}
```

$Q_c \equiv (res = \sum_{j=0}^{n} j)$

Proponemos el siguiente invariante:

$$\mathscr{I} \equiv (res = \sum_{j=0}^{i-1} j) \land 0 \leq i \leq n+1$$

Terminación

Para demostrar **terminación** de

$$\{P_c\} \;\; while(B)\ S_c; \;\; \{Q_c\}$$

necesitamos proponer una función variante fv y demostrar que:

- $\{\mathscr{I} \wedge B = \mathtt{\textbf{true}} \wedge fv = fv_0\} \;\; S_c \;\; \{fv < fv_0\}$
  La función variante es estrictamente decreciente al ejecutar una iteración del ciclo.
- $\mathscr{I} \wedge fv \leq 0 \implies B = \mathtt{\textbf{false}}$
  Si la función variante llega a cero, entonces la condición del ciclo se vuelve falsa.

Función variante

$P_c \equiv (i = 0 \wedge res = 0 \wedge n \geq 0)$

```
while(i <= n){
    res = res + i;
    i = i + 1;
}
```

$Q_c \equiv (res = \sum_{j=0}^{n} j)$

Ejemplo con $n = 6$:

i	n	n + 1 - i
0	6	7
1	6	6
2	6	5
3	6	4
4	6	3
5	6	2
6	6	1
7	6	0

Función variante:

$$fv = n + 1 - i$$

Terminación - Ejemplo

Si nuestra función variante es

$$fv = n + 1 - i$$

tenemos que mostrar que

- $\{\mathscr{I} \land B = \mathtt{\textbf{true}} \land n + 1 - i = fv_0\} \quad S_c \quad \{n + 1 - i < fv_0\}$
- $\mathscr{I} \land n + 1 - i \leq 0 \implies B = \mathtt{\textbf{false}}$

Repaso de la clase

Hoy vimos

- Estados de un programa
- Triplas de Hoare y su relación con las pruebas de corrección de programas
- Especificación de ciclos

Guía de ejercicios

- Ya pueden hacer toda la Guía 3.

Bibliografía:

- Estados y Triplas de Hoare: **Gries**, capítulo 6.
- Teorema del invariante: **Gries**, capítulo 11.