

Transformación de Estados

Algoritmos y Estructuras de Datos I

Transformación de estados

- ▶ Llamamos **estado** de un programa a los valores de todas sus variables en un punto de su ejecución:
 1. Antes de ejecutar la primera instrucción,
 2. entre dos instrucciones, y
 3. después de ejecutar la última instrucción.
- ▶ Podemos considerar la **ejecución** de un programa como una **sucesión de estados**.
- ▶ La asignación es la instrucción que permite pasar de un estado al siguiente en esta sucesión de estados.
- ▶ Las estructuras de control se limitan a especificar el flujo de ejecución (es decir, el orden de ejecución de las asignaciones).

Afirmaciones sobre estados

- ▶ Sea el siguiente programa que se ejecuta con estado inicial $\{True\}$.
- ▶ $\{True\}$

```
int x = 0;  
{x = 0}  
x = x + 3;  
{x = 3}  
x = 2 * x;  
{x = 6}
```
- ▶ ¿Finaliza siempre el programa? Sí, porque no hay ciclos
- ▶ ¿Cuál es el estado final al finalizar su ejecución? $\{x = 6\}$

Afirmaciones sobre estados

- ▶ Sea el siguiente programa que se ejecuta con estado inicial con una variable a ya definida ($\{a = A_0\}$).
- ▶ $\{a = A_0\}$

```
int b = a + 2;  
{a = A_0 ∧ b = A_0 + 2}  
int result = b - 1;  
{a = A_0 ∧ b = A_0 + 2 ∧ result = (A_0 + 2) - 1 = A_0 + 1}
```
- ▶ ¿Finaliza siempre el programa? Sí, porque no hay ciclos
- ▶ ¿Cuál es el estado final al finalizar su ejecución?
 $\{a = A_0 \wedge b = A_0 + 2 \wedge result = A_0 + 1\} \Rightarrow \{result = a + 1\}$

Corrección de un programa

- **Definición.** Decimos que un programa S es **correcto respecto de una especificación** dada por una precondición P y una postcondición Q , si siempre que el programa comienza en un estado que cumple P , el programa **termina su ejecución**, y en el estado final **se cumple** Q .
- **Notación.** Cuando S es correcto respecto de la especificación (P, Q) , lo denotamos con la siguiente **tripla de Hoare**:

$$\{P\} S \{Q\}.$$

Afirmaciones sobre estados

- Sea la siguiente especificación para incrementar en una unidad el valor de un entero.
- $\text{proc spec_incrementar}(\text{in } a : \mathbb{Z}, \text{out } \text{result} : \mathbb{Z})\{\$
 Pre $\{\text{True}\}$
 Post $\{\text{result} = a + 1\}$
 $\}$
- ¿Es el siguiente programa S **correcto** con respecto a su especificación?

```
1  int incrementar(int a) {  
2    int b = a + 2;  
3    int result = b - 1;  
4    return result;  
5  }
```

Ejemplo

- Sea el siguiente programa que se ejecuta con estado inicial con una variable $a = A_0$.
- $\{a = A_0\}$
 int $b = a + 2$;
 $\{a = A_0 \wedge b = A_0 + 2\}$
 int $\text{result} = b - 1$;
 $\{a = A_0 \wedge b = A_0 + 2 \wedge \text{result} = (A_0 + 2) - 1 = A_0 + 1\}$
 Por lo tanto, se deduce que:
 $\{\text{result} = a + 1\}$

Intercambiando los valores de dos variables enteras

- **Ejemplo:** Intercambiamos los valores de dos variables, pero sin una variable auxiliar!
- $\{a = A_0 \wedge b = B_0\}$
 $a = a + b$;
 $\{a = A_0 + B_0 \wedge b = B_0\}$
 $b = a - b$;
 $\{a = A_0 + B_0 \wedge b = (A_0 + B_0) - B_0\}$
 $\equiv \{a = A_0 + B_0 \wedge b = A_0\}$
 $a = a - b$;
 $\{a = A_0 + B_0 - A_0 \wedge b = A_0\}$
 $\equiv \{a = B_0 \wedge b = A_0\}$

Alternativas

- ▶ Sea el siguiente programa con una variable a de entrada cuyo valor no se modifica (i.e. podemos asumir $a = A_0$ como constante)
- ▶ Cuando tenemos una alternativa, debemos considerar las dos ramas por separado.
- ▶ Por ejemplo:

```
1 int modulo(int a) {  
2   int b = 0;  
3   if( a > 0 ) {  
4     b = a;  
5   } else {  
6     b = -a;  
7   }  
8   return b;  
9 }
```

- ▶ Verificamos ahora que $b = |a|$ después de la alternativa.

Alternativas

- ▶ Rama positiva:

Se cumple la condición

$$\{a = A_0 \wedge b = 0 \wedge B\} \equiv \{a = A_0 \wedge b = 0 \wedge A_0 > 0\}$$

$b = a$;

$$\{a = A_0 \wedge b = A_0 \wedge A_0 > 0\}$$

$$\Rightarrow \{b = |a|\}$$

- ▶ Rama negativa:

No se cumple la condición

$$\{a = A_0 \wedge b = 0 \wedge \neg B\} \equiv \{a = A_0 \wedge b = 0 \wedge A_0 \leq 0\}$$

$b = -a$;

$$\{a = A_0 \wedge b = -A_0 \wedge A_0 \leq 0\}$$

$$\Rightarrow \{b = |a|\}$$

- ▶ En ambos casos vale $b = |a|$
- ▶ Por lo tanto, esta condición vale al salir de la instrucción alternativa.

Ciclos (repaso)

- ▶ Recordemos la **sintaxis** de un ciclo:

```
while (B) {  
    cuerpo del ciclo  
}
```

- ▶ Se repite el cuerpo del ciclo mientras la **guarda** B se cumpla, cero o más veces. Cada repetición se llama una **iteración**.
- ▶ La ejecución del ciclo **termina** si no se cumple la guarda al comienzo de su ejecución o bien luego de ejecutar una iteración.
- ▶ Si/cuando el ciclo termina, el estado resultante es el estado posterior a la última instrucción del cuerpo del ciclo.

Ejemplo

- ▶ $\text{proc } \text{sumar}(\text{in } n : \mathbb{Z}, \text{out } \text{result} : \mathbb{Z}) \{$
 Pre $\{n \geq 0\}$
 Post $\{\text{result} = \sum_{i=1}^n i\}$
}

```
1 int sumar(int n) {  
2   int i = 1;  
3   int s = 0;  
4  
5   while( i <= n ) {  
6     s = s + i;  
7     i = i + 1;  
8   }  
9   return s;  
10 }
```

Ejemplo

- ▶ Estados al finalizar cada iteración del ciclo, para $n = 6$:

Iteración	i	s
0	1	$0 = 0$
1	2	$1 = 0 + 1$
2	3	$3 = 0 + 1 + 2$
▶ 3	4	$6 = 0 + 1 + 2 + 3$
4	5	$10 = 0 + 1 + 2 + 3 + 4$
5	6	$15 = 0 + 1 + 2 + 3 + 4 + 5$
6	7	$21 = 0 + 1 + 2 + 3 + 4 + 5 + 6$

- ▶ **Observación:** Luego de cada iteración vale que:

$$s = \sum_{k=1}^{i-1} k$$

- ▶ A este tipo de afirmación se denomina un **invariante** del ciclo.

Invariante de un ciclo

- ▶ **Definición.** Un predicado I es un **invariante** de un ciclo si:

1. I vale antes de comenzar el ciclo, y
2. si vale $I \wedge B$ al comenzar una iteración arbitraria, entonces sigue valiendo I al finalizar la ejecución del cuerpo del ciclo.

- ▶ Un invariante describe un estado que se satisface cada vez que comienza la ejecución del cuerpo de un ciclo y también se cumple cuando la ejecución del cuerpo del ciclo concluye.

- ▶ Por ejemplo, otros invariantes para este ciclo son:

- ▶ $I' \equiv i \neq 0$
- ▶ $I'' \equiv s \geq 0$
- ▶ $i \geq 1$
- ▶ ...etc

Ejemplo

- ▶ ¿La ejecución del cuerpo del ciclo preserva $I \equiv s = \sum_{k=1}^{i-1} k$?
- ▶ Para chequear esto, asumimos que vale $I \wedge B$ ya que se cumplió la condición para ejecutar el cuerpo del ciclo.
- ▶ Agregamos metavariables para las variables que cambian.
- ▶ $\{i = i_0 \wedge s = S_0 \wedge S_0 = \sum_{k=1}^{i_0-1} k \wedge (i_0 \leq n)\}$

$s = s + i;$

$$\{i = i_0 \wedge s = S_0 + i_0 \wedge S_0 = \sum_{k=1}^{i_0-1} k \wedge (i_0 \leq n)\}$$

$$\Rightarrow \{s = \sum_{k=1}^{i_0-1} k + i_0\}$$

$$\not\Rightarrow \{s = \sum_{k=1}^{i_0} k\}$$

¿Qué pasa si $i_0 = -1, -2, \text{etc..?}$

Sólo vale la implicación si $i_0 \geq 0$

$i = i + 1;$

Ejemplo

- ▶ El predicado $I \equiv s = \sum_{k=1}^{i-1} k$ (por sí solo) **no es un invariante de ciclo** ya que la ejecución del ciclo no lo preserva.

Iteración	i	s
0	1	$0 = 0$
1	2	$1 = 0 + 1$
2	3	$3 = 0 + 1 + 2$
▶ 3	4	$6 = 0 + 1 + 2 + 3$
4	5	$10 = 0 + 1 + 2 + 3 + 4$
5	6	$15 = 0 + 1 + 2 + 3 + 4 + 5$
6	7	$21 = 0 + 1 + 2 + 3 + 4 + 5 + 6$

- ▶ ¿Cómo podemos **reforzar** I para obtener un auténtico invariante para el ciclo?
- ▶ Nueva propuesta de invariante I :

$$i \geq 1 \wedge s = \sum_{k=1}^{i-1} k$$

Ejemplo

- ¿Vale $I \equiv i \geq 1 \wedge s = \sum_{k=1}^{i-1} k$ al principio del ciclo (i.e., antes de la instrucción while)?

```

1  int i = 1;
2  int s = 0;
3
4  while( i <= n ) {
5      s = s + i;
6      i = i + 1;
7  }
```

- Antes de ejecutar el ciclo el estado de la ejecución cumple que $n \geq 0 \wedge i = 1 \wedge s = 0$.
- Esto implica que $I \equiv i \geq 1 \wedge s = \sum_{k=1}^{i-1} k$.
- Por lo tanto, se cumple I al principio del ciclo.

Ejemplo

- ¿La ejecución del cuerpo del ciclo preserva $I \equiv i \geq 1 \wedge s = \sum_{k=1}^{i-1} k$?
- Para chequear esto, asumimos que vale $I \wedge B$ ya que se cumplió la condición para ejecutar el cuerpo del ciclo.
- Agregamos metavariabes para las variables que cambian.
- $\{i = l_0 \wedge s = S_0 \wedge l_0 \geq 1 \wedge S_0 = \sum_{k=1}^{l_0-1} k \wedge (l_0 \leq n)\}$
 $s = s + i;$
 $\{i = l_0 \wedge s = S_0 + l_0 \wedge l_0 \geq 1 \wedge S_0 = \sum_{k=1}^{l_0-1} k \wedge (l_0 \leq n)\}$
 $\Rightarrow \{s = \sum_{k=1}^{l_0-1} k + l_0\}$
 $\Rightarrow \{s = \sum_{k=1}^{l_0} k\}$ Este paso se puede aplicar ya que $i_0 \geq 0$
 $\{i = l_0 \wedge s = \sum_{k=1}^{l_0} k \wedge l_0 \geq 1 \wedge S_0 = \sum_{k=1}^{l_0-1} k \wedge (l_0 \leq n)\}$
 $i = i + 1;$
 $\{i = l_0 + 1 \wedge s = \sum_{k=1}^{l_0} k \wedge l_0 \geq 1 \wedge S_0 = \sum_{k=1}^{l_0-1} k \wedge (l_0 \leq n)\}$
 $\Rightarrow \{i \geq 1 \wedge s = \sum_{k=1}^{i-1} k\} \equiv \{I\}$

Ejemplo

- Tenemos entonces:
 1. I vale justo antes de comenzar el ciclo.
 2. Si valía la condición B e I , I sigue valiendo luego de la ejecución del cuerpo del ciclo.
- Esto implica que I también vale cuando el ciclo termina.
- Si se salió del ciclo fue porque no se cumplió $i \leq n$, y entonces estamos en un estado que satisface:

$$n \geq 0 \wedge I \wedge i > n \equiv n \geq 0 \wedge i \geq 1 \wedge s = \sum_{k=1}^{i-1} k \wedge i > n$$

Invariante de un ciclo

- Los invariantes de un ciclo permiten razonar sobre su corrección. Llamamos ...
 1. P_C : Precondición del ciclo,
 2. Q_C : Postcondición del ciclo,
 3. B : Guarda del ciclo,
 4. I : Un invariante del ciclo.
- Si se cumplen estas condiciones ...
 1. $P_C \Rightarrow I$,
 2. $\{I \wedge B\}$ cuerpo del ciclo $\{I\}$,
 3. $I \wedge \neg B \Rightarrow Q_C$,
- ... entonces el ciclo es parcialmente correcto respecto de su especificación (si termina, termina en Q_C).

Teorema del invariante

- **Teorema del invariante.** Si existe un predicado I tal que ...

1. $P_C \Rightarrow I$,
2. $\{I \wedge B\} S \{I\}$,
3. $I \wedge \neg B \Rightarrow Q_C$,

entonces el ciclo **while(B) S** es parcialmente correcto respecto de la especificación (P_C, Q_C) .

- Este teorema es la **herramienta principal** para argumentar la corrección de ciclos.
- El teorema del invariante se puede demostrar formalmente (más detalle en las próximas teóricas!).

Ejemplo

- Verifiquemos estas tres condiciones con el ejemplo anterior, y con ...

1. $P_C \equiv n \geq 0 \wedge i = 1 \wedge s = 0$
2. $Q_C \equiv n \geq 0 \wedge s = \sum_{k=1}^n k$
3. $B_C \equiv i \leq n$
4. $I \equiv i \geq 1 \wedge s = \sum_{k=1}^{i-1} k$

- En primer lugar, debemos verificar que $P_C \Rightarrow I$:
- Ya probamos anteriormente que:

$$(n \geq 0 \wedge i = 1 \wedge s = 0) \Rightarrow i \geq 1 \wedge s = \sum_{k=1}^{i-1} k.$$

- Por lo tanto, podemos concluir que se cumple la condición $P_C \Rightarrow I$

Ejemplo

- ¿Es cierto que $\{I \wedge B\} S \{I\}$?

$$\begin{aligned} I \wedge B : \{i \leq n \wedge i \geq 1 \wedge s = \sum_{k=1}^{i-1} k\} \\ s = s + i; \\ i = i + 1; \\ I : \{i \geq 1 \wedge s = \sum_{k=1}^{i-1} k\} \end{aligned}$$

- Esto también lo probamos cuando demostramos que I era un invariante para el ciclo.

Ejemplo

- Finalmente, ¿es cierto que $I \wedge \neg B \Rightarrow Q_C$?

$$i \geq 1 \wedge s = \sum_{k=1}^{i-1} k \wedge i > n \Rightarrow s = \sum_{k=1}^n k ?$$

- **No!** Contraejemplo: Si $i = n + 2$, entonces la implicación no vale!
- Sin embargo, **sabemos** que esto no puede pasar, puesto que $i \leq n + 1$ a lo largo del ciclo.
- ¿Qué hacemos?
- ⇒ Reforzamos el invariante!

Ejemplo

- Proponemos el nuevo invariante de ciclo reforzado (i.e. mas restrictivo):

$$I \equiv 1 \leq i \leq n+1 \wedge s = \sum_{k=1}^{i-1} k$$

- ¿Vale ahora que tenemos que $I \wedge \neg B \Rightarrow Q_C$?

$$1 \leq i \leq n+1 \wedge s = \sum_{k=1}^{i-1} k \wedge i > n$$

$$\Rightarrow i = n+1 \wedge s = \sum_{k=1}^{i-1} k$$

$$\Rightarrow s = \sum_{k=1}^n k \equiv Q_C$$

Ejemplo

- ¿Qué pasa con los dos primeros puntos del teorema del invariante?
 - $P_C \Rightarrow I$
 - $\{I \wedge B\}$ cuerpo del ciclo $\{I\}$
- ¿Se siguen verificando estas condiciones con el nuevo invariante?
- Hay que demostrarlo nuevamente! Si $I' \Rightarrow I$ no podemos concluir que $P_C \Rightarrow I'$.

Para concluir...

- ¿ $P_C \Rightarrow I$?

$$P_C \equiv (n \geq 0 \wedge i = 1 \wedge s = 0) \Rightarrow 1 \leq i \leq n+1 \wedge s = \sum_{k=1}^{i-1} k$$

- Por lo tanto, se cumple que $P_C \Rightarrow I$

Para concluir...

- ¿La ejecución del cuerpo del ciclo preserva $I \equiv 1 \leq i \leq n+1 \wedge s = \sum_{k=1}^{i-1} k$?
- $\{i = l_0 \wedge s = S_0 \wedge 1 \leq l_0 \leq n+1 \wedge S_0 = \sum_{k=1}^{l_0-1} k \wedge (l_0 \leq n)\}$
 $s = s + i;$
 $\{i = l_0 \wedge s = S_0 + l_0 \wedge 1 \leq l_0 \leq n+1 \wedge S_0 = \sum_{k=1}^{l_0-1} k \wedge (l_0 \leq n)\}$
 $\Rightarrow \{s = \sum_{k=1}^{l_0-1} k + l_0\}$
 $\Rightarrow \{s = \sum_{k=1}^{l_0} k\}$ Este paso sólo se puede aplicar si $l_0 \geq 0$
 $\{i = l_0 \wedge s = \sum_{k=1}^{l_0} k \wedge 1 \leq l_0 \leq n+1 \wedge S_0 = \sum_{k=1}^{l_0-1} k \wedge (l_0 \leq n)\}$
 $i = i + 1;$
 $\{i = l_0 + 1 \wedge s = \sum_{k=1}^{l_0} k \wedge 1 \leq l_0 \leq n+1 \wedge S_0 = \sum_{k=1}^{l_0-1} k \wedge (l_0 \leq n)\}$
 $\Rightarrow \{1 \leq i \leq n+1 \wedge s = \sum_{k=1}^{i-1} k\} \equiv \{I\}$ Esto lo podemos hacer ya que $l_0 \geq n$

Resultado final

- ▶ Finalmente, Sean:
 1. $P_C \equiv n \geq 0 \wedge i = 1 \wedge s = 0$
 2. $Q_C \equiv n \geq 0 \wedge s = \sum_{k=1}^n k$
 3. $B_C \equiv i \leq n$
 4. $I \equiv i \geq 1 \wedge s = \sum_{k=1}^{i-1} k$
- ▶ Ya que demostramos que se cumplen las siguientes condiciones:
 1. $P_C \Rightarrow I$
 2. $\{I \wedge B\}$ cuerpo del ciclo $\{I\}$
 3. $I \wedge \neg B \Rightarrow Q_C$
- ▶ Entonces, por el Teorema del Invariante podemos concluir que el ciclo `while(B)` es parcialmente correcto respecto de la especificación P_C, Q_C .

Algunas observaciones

- ▶ $I \equiv 1 \leq i \leq n+1 \wedge s = \sum_{k=1}^{i-1} k$.
 1. En general, un buen invariante debe incluir el **rango** de la(s) **variable(s) de control** del ciclo.
 2. Además, debe incluir alguna afirmación sobre el **acumulador** del ciclo.
- ▶ Cuando tenemos un invariante I que permite demostrar la corrección parcial del ciclo, nos referimos a I como **el invariante** del ciclo.
 1. El invariante de un ciclo **caracteriza** las acciones del ciclo, y representa al **algoritmo** que el ciclo implementa.
- ▶ En general, es sencillo argumentar **informalmente** la terminación del ciclo (más detalles en las próximas teóricas).

Bibliografía

- ▶ David Gries - The Science of Programming
 - ▶ Chapter 6 - Using Assertions to Document Programs
 - ▶ Chapter 6.1 - Program Specifications
 - ▶ Chapter 6.2 - Representing Initial and Final Values of Variables
 - ▶ Chapter 6.3 - Proof Outlines (transformación de estados, alternativas)