Algoritmos y Estructuras de Datos I

Departamento de Computación - FCEyN - UBA

Corrección y Teorema del Invariante

Especificación, algoritmo, programa

1. **Especificación:** descripción del problema a resolver.
   - ¿Qué problema tenemos?
   - Habitualmente, dada en lenguaje formal.
   - Es un contrato que da las propiedades de los datos de entrada y las propiedades de la solución.
2. **Algoritmo:** descripción de la solución escrita para humanos.
   - ¿Cómo resolvemos el problema?
3. **Programa:** descripción de la solución para ser ejecutada en una computadora.
   - También, ¿cómo resolvemos el problema?
   - Pero descripto en un lenguaje de programación.

Un lenguaje imperativo simplificado

- SmallLang[1] es un lenguaje imperativo similar a C++ pero más sencillo.
- Instrucciones de SmallLang:
  1. Asignación: Instrucción **x := E**.
     - **x** es una variable. Por ej. **a**; **suma**; **acumulado**.
     - **E** es una expresión. Por ej. **7**; **2+3**; **x*4**.
  2. Nada: Instrucción **skip** que no hace nada.

  Una instrucción es un programa.
- Estructuras de control:
  1. Secuencia: **S1; S2** es un programa, si **S1** y **S2** son dos programas.
  2. Condicional: **if B then S1 else S2 endif** es un programa, si **B** es una expresión lógica y **S1** y **S2** son dos programas.
  3. Ciclo: **while B do S endwhile** es un programa, si **B** es una expresión lógica y **S** es un programa.

[1] *The Semantics of a Small Language* de David Gries

Un ejemplo de programa

- ```
  x := 0 ;
  x := x + 3 ;
  x := 2 * x
  ```

Al terminar este programa podemos decir que **x** tendra el valor **6**.
  ```

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).

Estados

- Si ejecutamos el siguiente programa con $\{True\}$ como estado inicial.
- $\{True\}$
  ```
  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? $\{x = 6\}$

Estados

- Sea el siguiente programa que se ejecuta con estado inicial con una variable a ya definida ($\{a = A_0\}$[2]).
  $\{a = A_0\}$
  ```
  c := a + 2;
  ```
  $\{a = A_0 \wedge c = A_0 + 2\}$
  ```
  result := c - 1
  ```
  $\{a = A_0 \wedge c = A_0 + 2 \wedge result = (A_0 + 2) - 1 = A_0 + 1\}$
- ¿Finaliza siempre el programa? Sí, porque no hay ciclos
- ¿Cuál es el estado final al finalizar su ejecución?
  $\{a = A_0 \wedge c = A_0 + 2 \wedge result = A_0 + 1\}$
  de lo que se deduce $\Rightarrow \{result = a + 1\}$

[2] Recordar que A_0 es una metavariable, que representa el valor inicial de la variable a

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.
- proc *incrementar*(inout $a : \mathbb{Z}$){
  Pre $\{a = A_0\}$
  Post $\{a = A_0 + 1\}$
  }
- ¿Es el siguiente programa S **correcto** con respecto a su especificación?

```
proc incrementar(inout a: Z) {
  c := a + 2;
  result := c - 1;
  a := result
}
```


Ejemplo

- proc *incrementar*(inout $a : \mathbb{Z}$){
  Pre $\{a = A_0\}$
  Post $\{a = A_0 + 1\}$
  }
- Sea el siguiente programa que se ejecuta con estado inicial con una variable $a = A_0$.
  $\{a = A_0\}$
  `c := a + 2;`
  $\{a = A_0 \wedge c = A_0 + 2\}$
  `result := c - 1;`
  $\{a = A_0 \wedge c = A_0 + 2 \wedge result = (A_0 + 2) - 1 = A_0 + 1\}$
  `a := result`
  $\{a = A_0 + 1 \wedge c = A_0 + 2 \wedge result = A_0 + 1\}$
  Por lo tanto, se deduce que:
  $\{a = A_0 + 1\}$
  así que es **correcto** con respecto a su especificación

Intercambiando los valores de dos variables enteras

- proc *swap*(inout $a : \mathbb{Z}$, inout $c : \mathbb{Z}$){
  Pre $\{a = A_0 \wedge c = C_0\}$
  Post $\{a = C_0 \wedge c = A_0\}$
  }
- **Ejemplo:** Intercambiamos los valores de dos variables, pero sin una variable auxiliar!

  $\{a = A_0 \wedge c = C_0\}$
  `a = a + c;`
  $\{a = A_0 + C_0 \wedge c = C_0\}$
  `c = a - c;`
  $\{a = A_0 + C_0 \wedge c = (A_0 + C_0) - C_0\}$
  $\equiv \{a = A_0 + C_0 \wedge c = A_0\}$
  `a = a - c;`
  $\{a = A_0 + C_0 - A_0 \wedge c = A_0\}$
  $\equiv \{a = C_0 \wedge c = A_0\}$ que es la Post especificada.

Alternativas

- Recordemos: **if B then S1 else S2 endif**
  el valor de la condición (**B**) da lugar a que se ejecute una de las 2 posibles ramas (**S1** o **S2**)
- En este caso, debemos considerar las dos ramas por separado.
- Sea el siguiente programa con una variable a de entrada (i.e. utilizaremos la metavariable A_0 para referirnos a su valor inicial)
  $\{a = A_0\}$

```
if( a > 0 ) {
  c = a;
} else {
  c = -a;
}
```

  ¿$\{c = |a|\}$?
- Verifiquemos ahora que $c = |a|$ después de la alternativa.

Alternativas

```
if( a > 0 ) {
  c = a;
} else {
  c = -a;
}
```

- Rama positiva:
  - Se cumple la condición B (i.e. $a > 0$)
  - $\{a = A_0 \wedge B\} \equiv \{a = A_0 \wedge A_0 > 0\}$
  - `c = a;`
  - $\{a = A_0 \wedge c = A_0 \wedge A_0 > 0\}$
  - $\Rightarrow \{c = |a|\}$
- Rama negativa:
  - No se cumple la condición B (i.e. $a <= 0$)
  - $\{a = A_0 \wedge \neg B\} \equiv \{a = A_0 \wedge A_0 \leq 0\}$
  - `c = -a;`
  - $\{a = A_0 \wedge c = -A_0 \wedge A_0 \leq 0\}$
  - $\Rightarrow \{c = |a|\}$
- En ambos casos vale $c = |a|$
- Por lo tanto, esta condición vale al salir de la instrucción alternativa.

Ciclos

- Recordemos la sintaxis de un ciclo:

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

- Se repite el cuerpo del ciclo S 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.
- Cuando el ciclo termina (si lo hace), el estado resultante es el estado posterior a la última instrucción del cuerpo del ciclo.

Ejemplo

$\{n \geq 0 \wedge j = 1 \wedge s = 0\}$

```
while( j ≤ n ) {
  s = s + j;
  j = j + 1
}
```

¿$\{s = \sum_{k=1}^{n} k\}$?

Ejemplo con n=6

```
while( j ≤ n ) {
  s = s + j;
  j = j + 1
}
```

- Estados de cada iteración del ciclo:

Antes del ciclo vale: $\{n \geq 0 \wedge j = 1 \wedge s = 0\}$

Iteración	j	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:** En las condiciones que estamos probando, luego de cada iteración vale que:

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

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, son posibles candidatos a invariantes para este ciclo:
  - $I' \equiv True$
  - $I'' \equiv j \neq 0$
  - $I''' \equiv s \geq 0$
  - $j \geq 1$
  - $j \leq n + 1$
  - ...*etc*

```
while( j ≤ n ) {
  s = s + j;
  j = j + 1
}
```


Teorema del Invariante

- Los invariantes de un ciclo permiten razonar sobre su corrección. Llamamos...
  - P_C: Precondición del ciclo,
  - Q_C: Postcondición del ciclo,
  - I: Un invariante del ciclo.
  - B: Guarda del ciclo,
  - S: El cuerpo del ciclo.

```
while( B ) {
  S
}
```

- **Teorema del invariante**: si se cumplen que
  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
- En otras palabras, 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.
- 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

```
while( j ≤ n ) {
  s = s + j;
  j = j + 1
}
```

- Volvamos a mirar el seguimiento

  Antes del ciclo vale: $\{n \geq 0 \wedge j = 1 \wedge s = 0\}$

Iteración	j	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	$21 = 0 + 1 + 2 + 3 + 4 + 5 + 6$

- Propuesta de invariante I:

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

Ejemplo

- Sabiendo que:
  - $P_C \equiv n \geq 0 \wedge j = 1 \wedge s = 0$
  - $Q_C \equiv n \geq 0 \wedge s = \sum_{k=1}^{n} k$
  - $B \equiv j \leq n$
- Con este invariante:

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

- Si se cumplen que:
  1. $P_C \Rightarrow I$,
  2. $\{I \wedge B\}$ S $\{I\}$,
  3. $I \wedge \neg B \Rightarrow Q_C$,
- por el Teorema del Invariante podemos decir que es parcialmente correcto.

¿Al llegar al ciclo vale I?

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

$$\begin{array}{ccc} P_C \Rightarrow & I \\ \left(n \geq 0 \wedge j = 1 \wedge s = 0\right) \Rightarrow & 1 \leq j \leq n+1 \wedge s = \sum_{k=1}^{j-1} k \end{array}$$

- Si vale $n \geq 0 \wedge j = 1$ podemos decir que vale $1 \leq j \leq n+1$
- Por $j = 1 \wedge s = 0$ podemos decir que vale $s = 0 = \sum_{k=1}^{0} k = \sum_{k=1}^{1-1} k = \sum_{k=1}^{j-1} k$
- Por lo tanto, se cumple que $P_C \Rightarrow I$

¿El cuerpo del ciclo preserva I?

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

$$\{j = J_0 \wedge s = S_0 \wedge \overbrace{1 \leq J_0 \leq n+1 \wedge S_0 = \sum_{k=1}^{J_0-1} k}^{I} \wedge \overbrace{(J_0 \leq n)}^{B}\}$$

```
s = s + j;
```

$\{j = J_0 \wedge s = S_0 + J_0 \wedge 1 \leq J_0 \leq n+1 \wedge S_0 = \sum_{k=1}^{J_0-1} k \wedge (J_0 \leq n)\}$
$\Rightarrow \{s = \sum_{k=1}^{J_0-1} k + J_0\}$
$\Rightarrow \{s = \sum_{k=1}^{J_0} k\}$ Esto vale porque $J_0 \geq 0$

$\{j = J_0 \wedge s = \sum_{k=1}^{J_0} k \wedge 1 \leq J_0 \leq n+1 \wedge S_0 = \sum_{k=1}^{J_0-1} k \wedge (J_0 \leq n)\}$

```
j = j + 1
```

$\{j = J_0+1 \wedge s = \sum_{k=1}^{J_0} k \wedge 1 \leq J_0 \leq n+1 \wedge S_0 = \sum_{k=1}^{J_0-1} k \wedge (J_0 \leq n)\}$
$\Rightarrow \{1 \leq j \leq n+1 \wedge s = \sum_{k=1}^{j-1} k\} \equiv \{I\}$ Esto vale ya que $J_0 \leq n$

Al salir del ciclo, ¿vale Q_C?

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

$$\overbrace{1 \leq j \leq n+1 \wedge s = \sum_{k=1}^{j-1} k}^{I} \wedge \overbrace{\neg(j \leq n)}^{\neg B} \Rightarrow \overbrace{n \geq 0 \wedge s = \sum_{k=1}^{n} k}^{Q_C} \ ?$$

- Como $1 \leq j \leq n+1$, podemos decir que $1 \leq n+1$, o $0 \leq n$, es decir, vale $n \geq 0$
- Como $1 \leq j \leq n+1 \ \wedge \neg(j \leq n)$, sabemos que $j \leq n+1$ y por la segunda $j > n$, con lo cual $j = n+1$, entonces $s = \sum_{k=1}^{(n+1)-1} k$, es decir $s = \sum_{k=1}^{n} k$
- Vale Q_C al salir del ciclo.

Resultado final

- Dados:
  1. $P_C \equiv n \geq 0 \wedge j = 1 \wedge s = 0$
  2. $Q_C \equiv n \geq 0 \wedge s = \sum_{k=1}^{n} k$
  3. $B \equiv j \leq n$
  4. $I \equiv 1 \leq j \leq (n+1) \wedge s = \sum_{k=1}^{j-1} k$
- Y 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) S` es parcialmente correcto respecto de la especificación P_C, Q_C.

¿Y si $I = s = \sum_{k=1}^{j-1} k$?

```
while( j ≤ n ) {
  s = s + j;
  j = j + 1
}
```

- Volvamos a mirar el seguimiento

  Antes del ciclo vale: $\{n \geq 0 \wedge j = 1 \wedge s = 0\}$

Iteración	j	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	$21 = 0 + 1 + 2 + 3 + 4 + 5 + 6$

- ¿Por qué no I?:

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

$I = s = \sum_{k=1}^{j-1} k$?

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

- Al igual que antes asumimos que vale $I \wedge B$ ya que se cumplió la condición para ejecutar el cuerpo del ciclo. Es decir, vale:

$$(s = \sum_{k=1}^{j-1} k) \wedge (j <= n)$$

- Veamos que pasa al ejecutar el cuerpo del ciclo.
- $\{j = J_0 \wedge s = S_0 \wedge S_0 = \sum_{k=1}^{J_0-1} k \wedge (J_0 \leq n)\}$

  ```
  s = s + j;
  ```

  $\{j = J_0 \wedge s = S_0 + J_0 \wedge S_0 = \sum_{k=1}^{J_0-1} k \wedge (J_0 \leq n)\}$

  $\Rightarrow \ \{s = \sum_{k=1}^{J_0-1} k + J_0\}$

  $\not\Rightarrow \ \{s = \sum_{k=1}^{J_0} k\}$ ¿Qué pasa si $J_0 = -1, -2, etc.$.?

  Sólo vale la implicación si $J_0 \geq 0$

  ```
  j = j + 1;
  ```

- Con este invariante no podemos probar los 3 puntos del teorema

Tarea

- Si planteamos que

$$I \equiv j \geq 1 \wedge s = \sum_{k=1}^{j-1} k$$

- ¿Podemos probar que el ciclo es parcialmente correcto respecto a la especificación?
- Spoiler: No, no se puede. Lo que dice este invariante no alcanza, ¿por qué?

Algunas observaciones

- $I \equiv 1 \leq j \leq n+1 \land s = \sum_{k=1}^{j-1} k$.
  1. El invariante refleja la hipótesis inductiva del ciclo.
  2. En general, un buen invariante debe incluir el rango de la(s) variable(s) de control del ciclo.
  3. 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 las asunciones y propiedades que hace nuestro algoritmo durante el ciclo.
- En general, es sencillo argumentar informalmente la terminación del ciclo (más detalles en las próximas teóricas).

Para concluir...

- Ojo: Para probar esto:

  $\{n \geq 0 \land j = 1 \land s = 0\}$

  ```
  while( j ≤ n ) {
    s = s + j;
    j = j + 1
  }
  ```

  $\{s = \sum_{k=1}^{n} k\}$
- Nos falta demostrar que si vale P_C el ciclo siempre termina.
- Por ahora, solo probamos que es parcialmente correcto[3]
- Vamos a ver como demostrar terminación en las próximas teóricas.

[3] Cuando termina, cumple Q_C, pero no sabemos si siempre termina

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)