Algoritmos y Estructuras de Datos · Escuela de Negocios, UTDT · Primer semestre 2024

Guía de Ejercicios 3: Verificación y especificación de ciclos

[Version: 1 de septiembre de 2024]

Objetivos:

- Entrenar la lectura y escritura de invariantes de ciclo.
- Entrenar la implementación de ciclos razonando sobre su invariante.
- Introducir la verificación de programas y ciclos.

Ejercicio 1. Demostrar la corrección de los siguientes programas usando verificación simbólica.

(a) `int calcular(int a)`
Pre: Verdadero

```
y = a;
x = y + 3;
res = x;
```

Post: $res = a + 3$

(b) `float maximo(float a, float b)`
Pre: Verdadero

```
if(a > b){
  res = a;
} else {
  res = b;
}
```

Post: $(res = a \lor res = b)$
$\land \ (res \geq a \land res \geq b)$

(c) `float invertir(bool b, float n)`
Pre: $n \neq 0$

```
res = n;
if(b == true){
  res = 1/res;
}
```

Post: $(b = true \implies res = 1/n)$
$\land \ (b = false \implies res = n)$

(d) `int duplicar(int n)`
Pre: Verdadero

```
if(n > 0){
  res = n*4;
  res = res/2;
} else{
  res = n*2;
}
```

Post: $res = n * 2$

Ejercicio 2. Sea la siguiente especificación del problema **Sumar Cuadrados Hasta** y una implementación en C++:

Calcula la suma de los cuadrados de los números enteros entre 1 y N.

`int sumar_cuadrados_hasta(int N)`
Pre: $N > 0$
Post: $res = \sum_{j=1}^{N} j^2$

```
int i = 1;
res = 0;
while(i <= N) {
    res = res + i*i;
    i = i + 1;
}
```

(a) ¿Cuál es el invariante de ciclo correcto?

- $I_1: \ 1 \leq i \leq N \wedge res = \sum_{j=0}^{i-1} j^2$
- $I_2: \ 1 \leq i \leq N+1 \wedge res = \sum_{j=0}^{i-1} j^2$
- $I_3: \ 1 \leq i \leq N+1 \wedge res = \sum_{j=0}^{N} j^2$
- $I_4: \ 1 \leq i \leq N+1 \wedge res = \sum_{j=0}^{i-1} j$

(b) Dar una precondición P_c y una postcondición Q_c adecuadas para el ciclo.

(c) Dar un ejemplo de valores de `res`, `i`, y `N` que cumplan el invariante y la guarda del ciclo.

(d) Mostrar los pasos de ejecución de una iteración del ciclo, comenzando por los valores del ítem anterior. Mostrar que el estado final de esa iteración satisface el invariante.

(e) Mostrar que la precondición del ciclo implica el invariante.

(f) Mostrar que la negación de la guarda junto con el invariante implican Q_c.

Ejercicio 3. Dar una implementación para las siguientes funciones. Escribir, para los ciclos, una precondicion P_c, una poscondición Q_c y un invariante I.

(a) `bool todos_positivos(vector<int> vec)`, que indica si todos los elementos de vec son positivos.

(b) `bool no_contiene_equis(vector<char> str)`, que indica si str no contiene la letra `'x'`.

Ejercicio 4. Dado el siguiente ciclo, escribir una precondicion P_c, una poscondición Q_c y un invariante I.

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

Ejercicio 5. Sea la siguiente especificación del problema **Suma de divisores** y una implementación en C++:

Calcula la suma de los divisores de N.

`int sumar_divisores(int N)`
Pre: $N > 0$
Post: $res = \sum_{j=1}^{N} j * \beta(N \mod j = 0)$

```
int i = 1; res = 2;
while(i < N) {
  i = i + 1;
  if (N % i == 0)
    res = res + 2*i;
}
res = res/2;
```

(a) Dar una precondición P_c y una postcondición Q_c adecuadas para el ciclo.

(b) ¿Cuál es el invariante de ciclo correcto?

- $I_1: \ 1 \leq i \leq N \wedge res = \sum_{j=1}^{i} j * \beta(N \mod j = 0)$
- $I_2: \ 1 \leq i < N \wedge res = 2 * \sum_{j=1}^{i} j * \beta(N \mod j = 0)$
- $I_3: \ 1 \leq i \leq N \wedge res = 2 * \sum_{j=1}^{i} j * \beta(N \mod j = 0)$
- $I_4: \ 1 \leq i \leq N \wedge res = 2 * \sum_{j=1}^{i} j$

(c) Dar un ejemplo de valores de `res`, `i`, y `N` que cumplan el invariante y la guarda ciclo.

(d) Mostrar los pasos de ejecución de una iteración del ciclo, comenzando por los valores del ítem anterior. Mostrar que el estado final de esa iteración satisface el invariante.

(e) Dada la función variante $fv = N - i$, probar que se cumplen las condiciones de terminación del ciclo dado. (Para probar que fv es decreciente, puede utilizar <u>verificación concreta</u> para un valor de N y i concreto.)

Ejercicio 6. Dar una implementación para las siguientes funciones. Escribir, para los ciclos, una precondicion P_c, una poscondición Q_c y un invariante I.

(a) `void sumar_n(vector<int> & ls, int n)`. Dada una lista de enteros `ls` y un entero `n`, modificar `ls` sumando `n` a cada uno de sus elementos. Por ejemplo, para `ls={1,9,7,7}` y `n=10`, el estado final de `ls` debe ser `{11,19,17,17}`.

(b) `int suma_y_reset(vector<int> v)`. Dado un `vector<int>` v, devuelve la suma de sus elementos y modifica el vector poniendo ceros en todas sus posiciones.

Ejercicio 7. Dado el siguiente ciclo, escribir una precondicion P_c, una poscondición Q_c y un invariante I.

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

Ejercicio 8. Dado el siguiente ciclo con su respectiva P_c, Q_c e invariante I:

```
int i = 0; res = 0;
while (i < v.size()) {
  res = res + v[i];
  i = i + 1;
}
res = res + 100;
```

$P_c: \ i = 0 \land res = 0 \land |v| \mod 2 = 0$
$Q_c: \ res = \sum_{i=0}^{|v|-1} v[i]$
$I: \ 0 \leq i \leq |v|/2 \land res = \sum_{j=0}^{i-1} v[2j] + v[2j+1]$

(a) El código no respeta el invariante dado, explicar por qué.

(b) Corregir el código para respetar el invariante.

(c) Dar un ejemplo de valores de `i`, `res` y `v` que cumplan el invariante, la guarda del ciclo, y **que comiencen la última iteración del ciclo**.

(d) Mostrar los pasos de ejecución de una iteración del ciclo, comenzando por los valores del ítem anterior.

(e) Mostrar que el estado final de esa iteración satisface el invariante, no satisface la guarda y sí satisface la poscondición del ciclo.

(f) Mostrar que la negación de la guarda junto con el invariante implican Q_c.

Ejercicio 9. Sea la siguiente especificación del problema **Mínima Posición** y una implementación en C++:

Devuelve una posición del mínimo elemento de una secuencia, y aumenta el valor del elemento en esa posición.

int minima_posicion(vector<**int**>& v)
Pre: $v = v_0 \wedge |v| > 0$
Post: $0 \leq res < |v| \wedge$
$\big((\forall i : \mathbf{int})\ 0 \leq i < |v| \implies v[i] \geq v[res]\big) \wedge$
$\big((\forall i : \mathbf{int})\ 0 \leq i < |v| \wedge i \neq res \implies v[i] = v_0[i]\big) \wedge$
$v[res] = v_0[res] + 1$

```
int i = 0; res = 0;
while(i < v.size()) {
    if (v[i] < v[res])
        res = i;
    i = i + 1;
}
v[res] = v[res] + 1;
```

(a) Dar un invariante I, una precondición P_c y una postcondición Q_c adecuadas para el ciclo.

(b) Mostrar que el invariante y la negación de la guarda implican Q_c

Ejercicio 10. Sea el siguiente ciclo con su pre y pos condición:

```
while (i < v.size()) {
  if(v[i] % 2 = 0){
    v[i] = v[i]/2;
  }
  i = i + 1;
}
```

$P_c:\ i = 0 \wedge v = v_0$
$Q_c:\ |v| = |v_0|$
$\wedge (\forall k : \mathbf{int})(0 \leq k < |v| \wedge v_0[k] \mod 2 \neq 0 \implies v[k] = v_0[k])$
$\wedge (\forall k : \mathbf{int})(0 \leq k < |v| \wedge v_0[k] \mod 2 = 0 \implies v[k] = v_0[k]/2)$

(a) Proponer un invariante de ciclo.

(b) Dar un ejemplo de ejecución de una iteración del ciclo y mostrar, para esa iteración, que el invariante se preserva.

(c) Proponer una función variante y demostrar que si la función variante llega a cero, entonces la condición del ciclo se vuelve falsa..

Ejercicio 11. Sea la siguiente especificación del problema **Ubicar Máximo** y una implementación en C++:

Ubica en la posición i del vector de entrada al máximo elemento de sus primeras $i + 1$ posiciones, intercambiándolo con el elemento original de la posición i. No modifica al resto de las posiciones del vector.

void ubicar_maximo(vector<**int**>& vec, **int** i)
Pre: $0 \leq i < |vec| \wedge vec = vec_0$
Post: $(\exists j : \mathbf{int})\ esMaximaPos(vec_0, j, 0, i) \wedge$
$\wedge\ esSwap(vec, vec_0, i, j)$

$\bullet esSwap(v_1, v_2, i, j):\ |v_1| = |v_2| \wedge v_1[i] = v_2[j] \wedge v_1[j] = v_2[i]$
$\wedge (\forall k : tint)\ 0 \leq k < |v_1| \wedge k \neq i \wedge k \neq j \implies v_1[k] = v_2[k]$
$\bullet esMaximaPos(v, k, i, j):\ i \leq k \leq j \wedge$
$(\forall n : \mathbf{int})\ 0 \leq n \leq j \implies v[n] \leq v[k]$

```
int j = 0; max = 0; int temp;
while(j <= i) {
    if (vec[j] > vec[max])
        max = j;
    j = j + 1;
}
temp = vec[max];
vec[max] = vec[i];
vec[i] = temp;
```

(a) Dar una precondición P_c, una postcondición Q_c y un invariante I adecuados para el ciclo.

Ejercicio 12. Sea la siguiente especificación del problema **Es Palíndromo**:

Dado un vector de caracteres, verifica si éste es palíndromo (capicúa). Por ejemplo, el `vector<char>` `"arribalabirra"` *es palíndromo, pero* `"holahola"` *no lo es.*

`bool es_palindromo(const vector<char>& s)`
Pre: $Verdadero$
Post: $res = \texttt{true} \iff (\forall i : \texttt{int})\ 0 \leq i < |s| \implies s[i] = s[|v| - i - 1]$

Y sea el siguiente invariante de ciclo:

$$I = \{0 \leq j \leq |s| \land res = \texttt{true} \iff (\forall i : \texttt{int})\ 0 \leq i < j \implies s[i] = s[|s| - i - 1]\}$$

(a) Escribir un programa con un único ciclo que resuelva el problema y sea coherente con el invariante de ciclo dado.

(b) Si el invariante dado fuera, en cambio,

$$I = \{0 \leq j \leq \frac{|s|}{2} \land res = \texttt{true} \iff (\forall i : \texttt{int})\ j < i < \frac{|s|}{2} \implies s[i] = s[|s| - i - 1]\}$$

analizar y describir cómo debería cambiar su implementación para ser consistente el nuevo invariante.

Ejercicio 13. Sea la siguiente especificación del problema **Invertir Rango**:

Dado un vector y dos posiciones cualesquiera de éste, se invierte el orden de los elementos de ese rango en el vector original. Por ejemplo, invertir el rango 1-5 del vector `{0,1,2,3,4,5,6}` *da como resultado* `{0,5,4,3,2,1,6}`

`void invertir_rango(vector<int>& v, int i, int j)`
Pre: $0 \leq i < |v| \land 0 \leq j < |v| \land v = v_0$
Post: $(\forall k : \texttt{int})\big(\ (0 \leq k < min(i,j) \lor max(i,j) < k < |v| \implies (v[k] = v_0[k])) \land$
$(min(i,j) \leq k \leq max(i,j) \implies v[k] = v_0[min(i,j) + max(i,j) - k])\ \big)$

• *Notar que no puede asumirse ninguna relación entre i y j aparte de que pertenecen al mismo rango.*

(a) Escribir un algoritmo que use un único ciclo y resuelva el problema dado.

(b) Proponer P_c, Q_c y I consistentes con el código escrito.

Ejercicio 14.

`int contar_iguales(const vector<int> & v1, const vector<int> & v2)`
Pre: $|v1| = |v2|$
Post: $res = \sum_{k=0}^{|v1|-1} \beta(v1[k] = v2[k])$

```
int res = 0;
int i = v1.size() - 1;
while (i >= 0){
    if(v1[i] == v2[i]){
        res = res + 1;
    }
    i = i - 1;
}
```

(a) Dar una precondición P_c, una postcondición Q_c y un invariante I adecuados para el ciclo.

(b) Dar un ejemplo de valores de `i`, `res`, `v1` y `v2` que cumplan el invariante, la guarda del ciclo, y que comiencen la última iteración del ciclo.

(c) Mostrar los pasos de ejecución de una iteración del ciclo, comenzando por los valores del ítem anterior.

(d) Mostrar que el estado final de esa iteración satisface el invariante, no satisface la guarda y sí satisface la poscondición del ciclo.

Ejercicio 15. Para los siguientes conjuntos de P_c, I, y Q_c, implementar el ciclo correspondiente.

(a) $P_c: \ i = 0 \land k > 0$
$I: \ res = \sum_{j=0}^{i-1} \beta(v[i] > k) * v[i] \land 0 \leq i \leq |v|$
$Q_c: \ res = \sum_{j=0}^{|v|-1} \beta(v[i] > k) * v[i]$

(b) $P_c: \ i = |v|$
$I: \ 0 \leq i \leq |v| \land res = \texttt{true} \iff (\exists j : \texttt{int})(i \leq j < |v| \land v[j] \mod 2 = 0)$
$Q_c: \ res = \texttt{true} \iff (\exists j : \texttt{int})(0 \leq j < |v| \land v[j] \mod 2 = 0)$

(c) $P_c: \ i = 0 \land v = v_0$
$I: \ |v| = |v_0| \land 0 \leq i \leq |v| \land (\forall j : \texttt{int})(0 \leq j < i \implies v[j] = v_0[j+1])$
$Q_c: \ |v| = |v_0| \land (\forall j : \texttt{int})(0 \leq j < |v| - 1 \implies v[j] = v_0[j+1])$

(d) $P_c: \ v = v_0 \land |v_0| = |w| \land i = 0$
$I: \ |v| = |v_0| \land 0 \leq i \leq |v| \land (\forall j : \texttt{int})(|v| - i \leq j < |v| \implies v[j] = v_0[j] + w[j])$
$Q_c: \ |v| = |v_0| \land (\forall j : \texttt{int})(0 \leq j < |v| \implies v[j] = v_0[j] + w[j])$

(El ciclo es parte de una función `void sumar_vector(vector<int> & v, vector<int> w)` que toma `v` por referencia y `w` por copia)