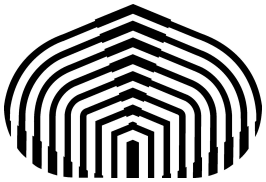

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la información
CI-2691- Laboratorio de algoritmos I

Laboratorio 3

El objetivo de este laboratorio es traducir algoritmos dados en GCL a Python, estudio de iteraciones y sus correspondientes invariantes y cota.

Contenido: Iteraciones, invariantes, cotas, ejemplos, ejercicios

Iteraciones

En GCL las iteraciones tienen la siguiente sintaxis:

```
do B0 → S0
[] B1 → S1
[] ...
[] Bn → Sn
Od
```

donde, **do** y **od** son palabras reservadas, B_i, $0 \leq i \leq n$, es una expresión booleana (o guardia) y su evaluación resulta en verdad o en falso, y B_i, $0 \leq i \leq n$, es una instrucción. Cada $B_i \rightarrow S_i$ es un comando con guardia, cuya guardia es B_i. La interpretación operacional de la instrucción iterativa es la siguiente: Se evalúan todas las guardias. Si todas las guardias son “falso” (esto equivale a la condición de terminación), entonces se ejecuta la instrucción `skip`. En caso contrario, se escoge una guardia con valor “verdad” y se procede a ejecutar la instrucción correspondiente a esa guardia; una vez ejecutada la instrucción, se procede a ejecutar la instrucción iterativa nuevamente.

Es importante recordar que GCL verifica en cada iteración que el invariante sea verdadero, y que la cota siempre sea positiva y decreciente. Los invariantes son aserciones intermedias que se escriben en GCL entre llaves ({}) precedida de la palabra reservada **inv**. En el caso de las cotas también se escriben entre llaves precedidas de la palabra reservada **bound**.

Invariantes

Son expresiones lógicas que generalmente contienen un cuantificador. Dicha expresión debe cumplirse al entrar al ciclo, y luego de cada iteración. En particular, el invariante se satisface luego de salir del ciclo. Un ejemplo de invariante en GCL, que verifica el cálculo de una suma, es el siguiente

```
{inv 0<=k<=N /\ suma =(%sigma i : 0<=i<k /\ (i mod 2 = 0): i}
```

El ciclo que satisface el invariante, tiene una variable **k**, cuyos valores están entre **0** y **N-1** inclusive. Cuando el ciclo termina **k** tiene el valor **N** por ello aparece en el rango del invariante de manera que se satisfaga esta condición a la salida del ciclo. La igualdad que aparece en el invariante a continuación del rango de valores de **k**, verifica que la variable **suma** tenga el valor de la sumatoria de números pares entre **k** y **N**.

Cotas

La cota es una función acotada que permite garantizar que el ciclo termina. Puede ser una función acotada inferiormente (con límite inferior) y que decrece estrictamente en cada iteración o puede ser acotada superiormente (con límite superior) y que crece estrictamente.

En cada iteración se verifica el valor de la cota, el cual debe ser menor (o mayor) al valor de la cota en el ciclo anterior. Además se verifica que dicho valor sea mayor o igual (o menor o igual) al límite inferior (o superior). Una función de cota para el invariante anterior es por ejemplo:

```
{bound N-k}
```

El valor inicial de esta cota es **N**, pues el valor inicial de **k** es **0**. Luego la cota decrece, a medida que **k** va creciendo en cada iteración. Al finalizar el ciclo, **k** tiene valor **N**, por lo que la cota tiene valor **N - N = 0**. Por ello se cumple que la cota es decreciente y el valor final es mayor o igual a cero.

También se pudo haber seleccionado la cota **k**, la cual es creciente y que está acotada superiormente por **N** (o su límite superior es **N**)

Ejemplo de Iteración en GCL y traducción a Python

A continuación se muestra el ciclo completo de una iteración en GCL que calcula la suma de los número pares positivos menores que N:

```
k,suma:=0,0;
{inv 0<=k<=N /\ suma =(%sigma i : 0<=i<k /\ (i mod 2 = 0): i}
{bound N-k}
do k < N ->
   if (k mod 2 = 0) ->
      suma:=suma + k
   [] (k mod 2 !=0) ->
      skip
   fi;
   k:=k+1
od
```

La traducción a Python de este ciclo sería:

```
k,suma=0,0
cota = N-k

# Verificacion de invariante y cota al inicio
assert( 0<=k<=N and suma == sum ( x for x in range(0,k) if (x % 2 ==0) ) )
assert( cota >= 0 )

while ( k < N ):
   if (k % 2 == 0):
      suma=suma+k
   else:
      pass
   k=k+1

   # Verificacion de invariante y cota en cada iteracion
   assert( 0<=k<=N and suma == sum ( x for x in range(0,k) if (x % 2 ==0) ) )
   assert( cota > N-k )
   cota = N-k
   assert( cota >= 0 )
```

Como en Python no es obligatorio el uso del `else`, es decir, es opcional, el código puede simplificarse de la siguiente forma:

```
k,suma=0,0
cota = N-k

# Verificacion de invariante y cota al inicio
assert( 0<=k<=N and suma == sum ( x for x in range(0,k) if (x % 2 ==0) ) )
assert( cota >= 0 )
```

```
while ( k < N ):
   if (k % 2 == 0):
      suma=suma+k
   k=k+1

   # Verificacion de invariante y cota en cada iteracion
   assert( 0<=k<=N and suma == sum ( x for x in range(0,k) if (x % 2 ==0) ) )
   assert( cota > N-k )
   cota = N-k
   assert( cota >= 0 )
```

El ejemplo completo puede verlo y probarlo en la carpeta del Laboratorio 3 del aula virtual con el nombre PreLab3Ejemplo1.py.

Ejercicios: Dadas los siguientes algoritmos en GCL, escriba un programa equivalente en Python. Guarde los programas en su espacio del aula virtual con los nombres sugeridos.

1. Prelab3Ejercicio1.py: algoritmo que calcula la suma de los factoriales desde 0 hasta N. Recuerde que 0!=1.

```
[
       const N: int;
       var suma: int;
       var fact: int;
       var k: int;

       {N >= 0}

       suma,fact,k := 0,1,0;

       {inv 0<=k<=N+1 /\ suma = (%sigma i: 0<=i<k: (%pi j: 1<=j<=i: j)) /\
            fact = (%pi j: 1<=j<=k: j) }
       {bound N+1-k}
       do ( k<=N ) ->
              if ( k > 0 ) ->
                     fact := fact * k
              []
                     skip;
              fi;
              suma := suma + fact;
              k := k + 1
       od

       { suma = (%sigma i: 0<=i<=N: (%pi j: 1<=j<=i: j)) }
]
```

Dado que Python no tiene definido el cuantificador productoria, usted puede definirlo incluyendo al comienzo de su código la siguiente definición de dicho cuantificador.

```
def prod( iterable ):
   p= 1
   for n in iterable:
      p *= n
   return p
```

El cual se usa igual que el resto de los cuantificadores. Por ejemplo, la productoria en GCL `(%pi j: 1<=j<i: j)` se traduce como `prod (j for j in range(1,i))`.

2. Prelab3Ejercicio2.py: algoritmo que cuenta los divisores de un número entero N.

```
[
      const N: int;
      var cuenta: int;
      var i: int;

      {N > 0}

      cuenta,i := 0,1;
      {inv 0<i<=N+1 /\ cuenta=(%count j: 0<j<i: N mod j = 0 )}
      {bound N-i+2 }

      do ( i<=N ) ->
            if ( N mod i = 0 ) ->
                  cuenta := cuenta + 1;
            [] ( N mod i != 0 )
                  skip
            fi;
            i = i+1
      od

      {cuenta = (%count j: 0<j<= N : N mod j = 0 )}
]
```

Sugerencia: puede utilizar el cuantificador sum de Python para representar el cuantificador de conteo. Note que `(%oount j: 0<j<= N: N mod j=0)=(%sigma j: 0<j<= N /\ N mod j = 0: 1 ).`