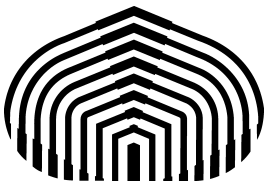

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la información
CI-2691- Laboratorio de algoritmos I

Prelaboratorio 5

El objetivo de este laboratorio es la verificación y prueba de las aserciones correspondientes a las precondiciones, postcondiciones y a las instrucciones de asignación, secuenciación y selección, los invariantes y función de cota de una iteración.

Verificación de las aserciones correspondientes a las precondiciones y postcondiciones

La verificación de la corrección de un programa sólo puede hacerse por métodos de pruebas formales tal como se enseña en el curso teórico “Algoritmos y Estructuras 1” [1]. Hay estrategias para el diseño de programas basados en las reglas de estas pruebas que permiten derivar programas correctos a partir de sus especificaciones formales. La limitación de dichas estrategias es que existen problemas cuyas soluciones algorítmicas no pueden derivarse con estas técnicas.

En la práctica profesional lo que se desea es el poder escribir programas que cumplan cabalmente con sus precondiciones y postcondiciones, a esto se le llama “programación basada en contrato”. Se desea asimismo que los programas sean escritos de forma tal que se eviten condiciones de error que de antemano se sabe que pueden ocurrir durante la ejecución, a esto se le llama "programación robusta". El propósito del presente prelaboratorio es aprender a hacer programación basada en contrato y programación robusta.

Esto se logra añadiendo al código del programa acciones que verifiquen el cumplimiento de las aserciones de corrección durante la ejecución (mediante la instrucción "try/except"). En caso de violación de alguna aserción, la acción a tomar es dar un mensaje informativo al usuario del error ocurrido y parar la ejecución del programa (mediante la instrucción sys.exit()), de manera que no vaya a propagarse el error.

En el caso que el error sea por el incumplimiento de alguna especificación de entrada, la programación robusta indica que, de ser posible, en lugar de parar la ejecución del

programa, se le da al usuario la opción de corregir el valor de entrada erróneo (se sugiere usar la instrucción "while").

Try/except

En el caso de Python, utilizaremos la instrucción `try/except` para tomar el control luego que un aserción falle. La idea básicamente es colocar las aserciones dentro de un bloque try/except, y en caso de fallar, el programa automáticamente comenzará a ejecutar las instrucciones indicadas en el except. La sintaxis es de la siguiente forma:

```
try:
  a = -1 # se coloca esta asignación a propósito a manera de
         # ejemplo
  assert( a >= 0 )
except:
  print("El numero no es positivo")
```

El código mostrado imprimirá el siguiente mensaje: "`El numero no es positivo`".

A continuación se coloca un ejemplo utilizando `sys.exit()`. Para utilizar `sys.exit()` es necesario importar la librería `sys` usando la instrucción `import` al inicio del programa, de la siguiente forma:

```
import sys

try:
  a = int(input("Introduzca un numero entero: "))
  # el usuario coloca 1.5
  assert( a >= 0 )
except:
  print("El numero no es valido, debe ser un entero positivo")
  print("El programa terminara")
  sys.exit()
```

En este caso, cuando el usuario no introduce un valor apropiado, el programa termina, de manera similar a como lo hace el `assert`, sólo que se puede dar un mensaje más entendible para el usuario. A continuación se coloca un ejemplo de programación robusta, donde el usuario tiene la posibilidad de volver a introducir el valor.

```
While True:
  try:
    a = int(input("Coloque un entero positivo: "))
    assert( a > 0 )
```

```
      break
    except:
      print("Hubo un error en los datos de entrada")
      print("Vuelva a intentar")
```

La instrucción **`break`** se utiliza para salir del `while`, es decir, el programa continúa su ejecución luego del bloque de instrucciones internas al `while`.

En el caso de las postcondiciones, se puede hacer un tratamiento similar para indicarle al usuario de manera más "amigable" que el programa no logró verificar la postcondición, indicando los valores actuales de las variables involucradas en dicha expresión. En ese caso tiene más sentido que el programa termine la ejecución porque luego de la postcondición sólo queda la salida de los resultado. Veamos un ejemplo:

```
try:
  assert( r == (suma <= x + y))
except:
  print("Hubo un error en los calculos")
  print("La expresion r == (suma <= x + y) no es correcta")
  print("r="+str(r)+" suma="+str(suma)+" x="+str(x)+" y="+str(y))
  sys.exit()
```

Ejemplo de programas en Python

Al realizar una compra en efectivo, es necesario dar el vuelto al comprador. Para dar el vuelto se dispone de billetes de 10, 5 y 2, así como monedas de 1. Dado el monto de una compra y del pago, se debe calcular el vuelto y entregarlo con el número mínimo posible de billetes y monedas. Los montos de compra y de pago son redondos (sin céntimos). Se desea solicitar al usuario los montos de una compra y del pago, y calcular el vuelto, en su desglose óptimo. Para mostrar el tipo de verificaciones que se deben hacer en este prelaboratorio, usted debe estudiar los programas que se encuentran en los archivos vueltoVersionContrato.py y vueltoVersionRobusta.py. Preste atención a las verificaciones que se hacen. Pruebe los programas introduciendo datos erróneos y datos correctos. Ahora introduzca errores adrede en las instrucciones de cálculo de cualquiera de estos programas y observe lo que pasa al ejecutarlo.

Ejercicio a entregar: Modifique el ejercicio denominado como "**Prelab2ejercicio2.py"** del prelaboratorio 2 para incluir las acciones de verificación antes explicadas. Las aserciones a verificar son las precondiciones y postcondiciones de los programas. Guarde el programa en su espacio del aula virtual con el nombre PreLab5ejercicio1.py antes del martes 12 de mayo del 2015 a las 9:00 am.

Verificación de las aserciones correspondientes a los invariantes y función de cota de una iteración

La verificación de la corrección de un ciclo requiere conocer el invariante que se satisface durante toda la ejecución del mismo, así como la función de cota que garantiza que dicho ciclo termina. Las pruebas formales que garantizan que un ciclo es correcto se estudian en el curso teórico "Algoritmos y Estructuras 1" [1]. Usualmente el invariante involucra obtener el valor de un cuantificador o de una función de agregación. Hasta ahora lo hemos verificado usando la instrucción `assert`.

Para encontrar errores en invariantes o función de cota planteada de manera práctica, se puede usar como estrategia la visualización, en cada iteración, de las variables que intervienen en el cuerpo del ciclo. Aquí también es útil el uso de la instrucción `try/except`. Asimismo, podemos verificar a través de una instrucción `try/except` si la función de cota es mayor o igual que cero y si es estrictamente decreciente. Observe que en caso de usar un ciclo `FOR`, esto no es necesario pues el `FOR` de Python siempre termina.

El propósito del presente prelaboratorio es aprender a verificar que las instrucciones iterativas son robustas mediante la inclusión en el código de acciones que permiten visualizar los valores de las variables y el chequeo de la finitud del ciclo, de manera “amigable” para el usuario.

En caso de violación de alguna de las aserciones correspondientes a invariantes o cotas, la acción a tomar es dar un mensaje informativo al usuario del error ocurrido y parar la ejecución del programa (mediante la instrucción `sys.exit()`), de manera tal que se garantice que la ejecución del ciclo es correcta y que no se cae en un ciclo infinito. Note que esto garantiza que los valores de las variables involucradas en el ciclo son correctos, así como el valor de la cota, a menos que ocurra una violación de un `assert`.

Veamos un ejemplo estudiado en el prelaboratorio 3. El siguiente programa calcula la suma de los números pares entre 0 y N. El programa tiene las aserciones correspondientes al invariante y la cota. Se incluye la verificación usando `try/except`.

```
k,suma=0,0
cota = N-k+1

# Verificacion de invariante al inicio
try:
   assert( 0<=k<=N and suma == sum ( x for x in range(0,k) if (x % 2 ==0) ) )
except:
   print("Hubo un error en el invariante para los siguientes valores")
   print("k="+str(k)+" suma="+str(suma))
   sys.exit()

# Verificacion de cota al inicio
```

```
try:
   assert( cota >= 0 )
except:
   print("Error cota no positiva: ")
   print("cota="+str(cota))
   sys.exit()

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

   # Verificacion de invariante en cada iteracion
   try:
      assert( 0<=k<=N and suma == sum ( x for x in range(0,k) if (x % 2 ==0) ) )
   except:
      print("Hubo un error en el invariante para los siguientes valores")
      print("k="+str(k)+" suma="+str(suma))
      sys.exit()

   # Verificacion de cota decreciente en cada iteración
   try:
      assert( cota > N-k+1 )
   except:
      print("Error cota no decreciente : ")
      print("cota anterior ="+str(cota)+" nueva cota ="+str(N-k+1))
      sys.exit()

   cota = N-k+1
   # Verificacion de cota positiva en cada iteración
   try:
      assert( cota >= 0 )
   except:
      print("Error cota no positiva: ")
      print("cota="+str(cota))
      sys.exit()
```

Ejercicio a entregar: Modifique el ejercicio denominado como "**Prelab3Ejercicio1.py**" del prelaboratorio 3 para incluir las verificaciones de invariantes y cotas con instrucción `try/except`. Guarde el programa en su espacio del aula virtual con el nombre de PreLab5ejercicio2.py antes del martes 12 de mayo del 2015 a las 9:00 am.

Referencias

[1] Oscar Meza y Jesús Ravelo, "Introducción a la Programación". Universidad Simón Bolívar, 2012. Disponible en la web: http://ldc.usb.ve/~meza/ci-2611/IntroduccionALaProgramacion.pdf