La importancia de la enseñanza del Invariante en la estructura iterativa

Lidia Marina López

Departamento de Informática y Estadística
Facultad de Economía y Administración
Universidad Nacional del Comahue
e-mail:llopez@uncoma.edu.ar

Keywords:invariante, bucle, aserto, educación

Abstract

La característica más importante de la programación imperativa es la estructura iterativa (while, until, for). Apuntar a mostrar y enseñar una técnica formal que permita construir dicha estructura asegurando su correctitud y aportando la forma de documentarla y verificarla es el objetivo de este trabajo.

Es conveniente expresar un problema en términos de un modelo formal, basado en reglas comprobables. Casi cualquier campo de la matemática, u otra ciencia, puede ser utilizado como ayuda para modelar el dominio de un problema. Una vez hallado un modelo adecuado, se trata de establecer una solución para el problema, ajustada a ese modelo.

El objetivo de este trabajo es presentar una manera formal para calcular y desarrollar una iteración cuidando hasta el último detalle, brindando una profunda comprensión de los programas y siendo una tarea mucho más fácil y rápida de verificación y corrección de la estructura mencionada.

La técnica consiste en imaginar congelar la computación en un punto crucial y dar una descripción estática del estado interno. De esta manera la técnica se completa especificando la situación de alguna forma lógica para obtener un *invariante de bucle*.

La importancia del mismo radica en que, con práctica, los alumnos podrán reconocer, diseñar e implementar bucles perfectamente documentados y verificados.

La importancia de la enseñanza del Invariante en la estructura iterativa

1 Introducción

Para expresar un problema en términos de un modelo formal, la ingeniería de software dispone de una variedad de técnicas. Existen los llamados métodos formales, sustentados en sistemas (modelos) formales, que utilizan semánticas precisas. Un lenguaje formal responde a una semántica precisa, distinta a los lenguajes naturales (inglés, francés, castellano, etc.) con el cual es posible construir expresiones no ambiguas. [Bis86]

Por ejemplo, en la presentación de lenguajes de programación, generalmente se describen simultáneamente la sintaxis de instrucciones y su semántica. Este detalle de significados puede ser informal (como en la mayoría de los manuales de introducción a distintos lenguajes de programación) o formal. En particular, un lenguaje formal es necesario como soporte de un método formal.

En la resolución de algoritmos, se apunta a identificar los conceptos esenciales mientras se ignoran los no esenciales. Un modelo así representa una **abstracción**.[LG86]

Primero debemos describir cuidadosamente el objetivo del bucle, es decir, la postcondición (o, generalizando, del algoritmo) y las condiciones iniciales y estado de las variables para ejecutar el bucle, es decir, la precondición, y luego podremos deducir instrucciones algorítmicas a partir de esta especificación. Este proceso se denomina **derivación**. [Gri81]

En la derivación de un algoritmo, al igual que cuando éste se desarrolla intuitivamente, los resultados que se desean obtener influyen sobre las decisiones de diseño mucho más que los datos de que se dispone.

Este hecho se traduce en que, la postcondición proporciona mucha más información que la precondición, sin que ésta sea irrelevante, por supuesto. El análisis de la postcondición resulta así un buen método para desarrollar algoritmos. Cuando la postcondición no brinde suficiente información se busca una expresión equivalente. Veamos algunas consideraciones naturales:[Gri81]

- Si en la postcondición aparecen igualdades entre variables del programa y expresiones, puede intentarse satisfacer éstas mediante asignaciones simples o múltiples.
- Si en la postcondición aparecen disyunciones, un criterio útil puede ser intentar diseñar una alternativa, cada una de cuyas ramas obtenga la postcondición a base de satisfacer una de las disyunciones.
- Si en la postcondición aparecen conjunciones, puede ser útil considerarlas de forma aislada y tratar se satisfacerlas por separado; otra posibilidad es crear alternativas.
- *Si en la postcondición aparecen operaciones que efectúan cálculos por repetición podemos intentar diseñar una iteración.*

En este trabajo se apunta a desarrollar algoritmos cuyas postcondiciones correspondan al último punto anterior (texto enfatizado).

A lo largo del diseño de un algoritmo es conveniente que todas las decisiones sean razonadas, explicando por qué se elige construir el programa de determinada manera, lo

que no significa que algunas veces debamos tomar decisiones arbitrarias que no puedan ser argumentadas. Esto no es tan malo, lo importante es ser concientes de ellas en todo momento porque en caso de fracaso, son las primeras que deberemos reconsiderar.

El objetivo de este trabajo es presentar una manera formal para calcular y desarrollar una iteración cuidando hasta el último detalle, brindando una profunda comprensión de los programas y siendo una tarea mucho más fácil y rápida de verificación y corrección de la estructura mencionada.

Este trabajo está organizado de la siguiente manera: la sección 2 *Conceptos básicos* revisa definiciones algebraicas y propone una notación, la sección 3 *Derivación de bucle* muestra cómo introducir la técnica de derivación, la sección 4 *Ejemplos de derivación* completa con dos ejemplos las ideas de la sección anterior y la sección 5 *Conclusión* resume la propuesta y concluye con las ventajas que produciría la aplicación de la técnica.

2 Conceptos básicos

> *Los conceptos que se pueden manejar dependen de la preparación lógica y matemática del alumno. Si no se cuenta con un curso de introducción al álgebra previo hay que adaptar las siguientes definiciones de acuerdo al conocimiento de los alumnos.*

Se propone trabajar con las siguientes definiciones: [Bal93]

Alfabeto: Conjunto de símbolos permitidos para construir expresiones y asertos. Las letras minúsculas $(i, j, x, y, z, \dots)$ denotan las variables y las letras mayúsculas denotan valores constantes $(X, Y, A, B, \dots)$.

Expresión: Es una fórmula lógica.

Aserto o Predicado: Es una relación lógica entre un conjunto de expresiones que resulta en VERDADERO o FALSO en función de los valores de verdad asignado a esas expresiones. Se escriben entre llaves.

Estado: Es la aplicación de valores de verdad en un conjunto de variables, de manera que se asocie, en un momento dado, a cada variable un valor coherente con su tipo, tales que describa correctamente una situación particular.

Proceso: Es una sucesión de estados posiblemente infinita. Usualmente, el análisis de un programa puede hacerse en términos de un conjunto de estados iniciales admisibles y un conjunto de estados finales apropiados.

Como convención se fija:

Operadores relacionales:

$>$	mayor
$<$	menor
$=$	igual
$\geq$	mayor o igual
$\leq$	menor o igual
$\neq$	distinto

Operadores lógicos:

$\wedge$	conjunción
$\vee$	disyunción
$\neg$	negación

Cuantificadores:

$\sum$	suma
$\prod$	multiplicación
$\forall$	universal
$\exists$	existencial
$\sharp$	recuento

Un cuantificador es una abreviatura de una secuencia de operaciones análogas; requiere ir asociado a una variable ligada, que es simplemente un identificador, y a un dominio, que indica el conjunto de valores que permitimos tomar a la variable ligada y que frecuentemente es un intervalo de naturales. La expresión así obtenida denota la *repetición de la operación* sobre todos los valores del dominio.

Así pues, si $E(x)$ es una expresión natural o entera, que depende de la variable x,

$$\sum x : Dominio : E(x)$$

$$\prod x : Dominio : E(x)$$

denotan, respectivamente, la suma y el producto de todos los valores tomados de la expresión E, cuando x recorre todos los valores indicados en el dominio. La variable ligada es x.

Si $E(x)$ es una expresión booleana,

$$\forall x : Dominio : E(x)$$

$$\exists x : Dominio : E(x)$$

denotan, respectivamente, la conjunción y disyunción de todos los valores tomados de la expresión E, cuando x recorre todos los valores indicados en el dominio.La variable ligada es x.

$$\sharp x : Dominio : E(x)$$

denota el natural que indica el número de valores en el dominio de x para los cuales E es cierta.

Si $E(x)$ es una expresión natural,

$$Min\ x : Dominio : E(x)$$

$$Max\ x : Dominio : E(x)$$

denotan, respectivamente, el menor y el mayor de todos los valores tomados de la expresión E, cuando x recorre todos los valores indicados en el dominio.

Además son aplicables los cuantificadores, sobre un dominio vacío (o nulo). Por convención el resultado de un cuantificador de recuento ($\sharp$) sobre un dominio vacío es cero, y los resultados de los cuantificadores aritméticos y booleanos sobre los dominios nulos son los elementos neutros de las correspondientes operaciones binarias.

Así, si $N = 0$ entonces

$$(\sum x : 1 \leq x \leq N : a(x)) = 0$$

$$(\prod x : 1 \leq x \leq N : a(x)) = 1$$

$$(\forall x : 1 \leq x \leq N : a(x)) = verdadero$$

$$(\exists x : 1 \leq x \leq N : a(x)) = falso$$

$$(\sharp x : 1 \leq x \leq N : a(x)) = 0$$

3 Derivación del bucle

Una buena regla para introducir conceptos nuevos a los estudiantes es hacerlo paulatinamente, tan pronto como sea posible y solamente si están capacitados para manejarlos. En el caso de los invariantes de bucle, es posible introducir parte de la idea al mismo tiempo que la estructura iterativa WHILE es mostrada permitiendo exponer la idea y asociar inmediatamente el invariante con la sentencia WHILE.

Cuando la sentencia WHILE es introducida por primera vez se puede:

- Dar una definición preliminar de un invariante: es una expresión lógica (aserto) que es verdadero justo antes del WHILE y justo después del cuerpo del bucle

Este concepto debe ser introducido con suficientes y variados ejemplos como los siguientes, que permiten reconocer invariantes fácilmente:

```
BUCLE 1              BUCLE 2
x:=0;                n:=m;
while x < 20 do      t:=0;
   begin             while t < 10 do
   x:=x+5               begin
   end                  t:=t+1;
                        n:=n+1
                        end
```

En el BUCLE 1 se puede ver que el invariante incluye $x \geq 0$ y que x es múltiplo de 5 y, en el BUCLE 2, el invariante incluye $n \geq m$.

Se introduce así la idea de un invariante como un medio para el análisis del bucle. El próximo paso es presentarlo como una herramienta para construirlo.

Los invariantes deben ser usados para construir bucles y no sólo como herramienta una vez que han sido diseñados.

La técnica para desarrollar un bucle consite en:

1. Determinar las variables necesarias para el bucle
2. Expresar la condición que es deseada después del bucle (objetivo del bucle)
3. Extraer, de alguna manera, de esta condición, el invariante y la condición de corte del bucle.
4. Usar la información anterior para construir el bucle.

Enseñar un bucle de esta manera requiere de tiempo para discutir la estrategia del bucle y, derivar el invariante y la condición de corte del bucle a partir de la estrategia misma. Esto se puede hacer presentando varios ejemplos numéricos. Un buen ejemplo, que es el favorito de la mayoría de los autores, es el cálculo de la división por restas repetidas:

- Calcular 13 / 4. Tenemos 13 elementos y debemos calcular cuántos grupos de cuatro podemos hacer.

Cantidad de elementos	Tamaño del grupo	Cantidad de grupos	Lo que queda
13	4	1	9
13	4	2	5
13	4	3	1

- ¿Dónde está el invariante? Se puede ver que en cada fila la cantidad de elementos, que es constante, es igual a la cantidad de grupos por el tamaño del grupo más lo que queda.
- ¿Cuándo parar? Podemos parar cuando lo que queda es menor que el tamaño del grupo.
- Elijamos nombres de variable para cada columna:

Cantidad de elementos	Tamaño del grupo	Cantidad de grupos	Lo que queda
a: integer	b: integer	q: integer	r: integer

- Ahora estamos en condiciones de establecer el invariante y la condición de corte del bucle:
  - Invariante: $a = q * b + r$
  - Condición de corte: $r < q$
  - Lo que progresa hacia la condición: $q := q + 1$
  - Construyamos el código:
    {Invariante: $a := q * b + r$}
    while $r \geq q$ do
      begin
        $r := r - b$
        $q := q + 1$
      end

– Finalmente podemos comprobar que el invariante es satisfecho antes del bucle y luego al final del cuerpo del bucle. Podemos establecer los valores de las variables antes del bucle necesarios: $q := 0; r := a$

Una vez incorporado estos conceptos intuitivos podemos formalizar la técnica estableciendo una notación matemática y un método.

4 Ejemplos de derivación

En esta sección se presentan dos ejemplos de cómo derivar un bucle.[Sch86]

4.1 Suma de los elementos de un vector entero

Dado el vector $V[0 \ldots N-1]$ de números enteros, se desea calcular su suma.

Establecemos la postcondición, es decir, el resultado final del cálculo:

$$Post : suma = \sum i : 0 \leq i < N : V[i]$$

En la definición anterior ya hemos tomado algunas decisiones:
La variable *suma* será el acumulador donde se almacenará el resultado final.
Como precondición se puede adelantar que $suma = 0$ y que $i = 0$.

- Invariante: Podemos deducir, usando un ejemplo o no, que en cada iteración la variable *suma* tiene almacenado la suma parcial del vector hasta la posición actual. Para representar esto utilizamos en el invariante otra variable (j) que refleja esta situación:
$$suma = \sum j : 0 \leq j < i : V[j]$$
- Condición de corte: $i = n$
- Lo que progresa hacia la condición: $i := i + 1$
- Construyamos el código:

  $suma := 0$

  $i := 0$

  $\{Invariante : suma = \sum j : 0 \leq j < i : V[j]\}$

  while $i < n$ do

  begin

  $suma := suma + V[i]$

  $i := i + 1$

  end

4.2 Máximo elemento de un vector

Dado el vector $V[0 \ldots N-1]$ de números enteros, se pide buscar el máximo elemento.
Establecemos la postcondición, es decir, el resultado final del cálculo:

$$Post : maximo = Max\ i : 0 \leq i < N : V[i]$$

Guardaremos en *maximo* el resultado final.
Ahora debemos analizar el algoritmo:

- Debemos guardar (en *maximo*) el primer elemento del vector para poder compararlo con el siguiente elemento.
- Si el siguiente elemento se mayor, se debe actualizar el valor de *maximo* y seguir con el siguiente elemento

Establecemos como precondición del bucle que: $maximo = V[0]$ y que $i = 1$

- Invariante: Otra vez, para representar el resultado parcial utilizamos en el invariante otra variable (j) que refleja esta situación:

$$maximo = Max\ j : 0 \leq j < i : V[j]$$

- Condición de corte: $i = n$
- Lo que progresa hacia la condición: $i := i + 1$
- Construyamos el código:

  $maximo := V[0]$

  $i := 1$

  $\{Invariante : maximo = Max\ j : 0 \leq j < i : V[j]\}$

  while $i < n$ do

  begin

  if $maximo < V[i]$ then $maximo := V[i]$

  $i := i + 1$

  end

5 Conclusión

Presentar el bucle acompañado por su invariante permite asociar dos conceptos fundamentales en la programación imperativa: la repetición y su análisis. La propuesta que aquí se presenta consiste en:

- Introducir el Invariante simultáneamente con la sentencia WHILE y usarlo inicialmente como un medio para analizar la iteración.
- Mostrar que al establecer claramente el objetivo del bucle (su Invariante) se puede aplicar la técnica de derivarlo.

Utilizar invariantes de bucle no solamente autodocumenta el código y permite verificarlo, sino que nos independiza del análisis de la primera iteración y de la última que son los momentos críticos del bucle.

Al programar en forma modular se realizan refinamientos sucesivos y se respetan las estructuras de control básicas: base de la programación estructurada. Esto es:

- Dividir el programa en submódulos independientes que se agrupan en un módulo principal
- Subdividir dichos submódulos mientras así lo requiera su complejidad, en otros submódulos hasta que tengan una tarea específica para ejecutar.
- Cada módulo y submódulo se codifican utilizando las tres estructuras de control básicas: asignación, selección y repetición.

Un programa así diseñado y, habiendo pasado la compilación satisfactoriamente, debe ser ejecutado con datos de prueba que contemplen "todas" las posibles entradas para poder decir que "funciona". Pero es difícil poder fundamentarlo.

Enseñar a diseñar un bucle aplicando el método que se ha descripto es mostrar cómo también es posible calcular los programas cuidando hasta el último detalle, porque en cualquiera de éstos puede esconderse el error. No se puede asegurar que siguiendo este método, todo programa será correcto pero brindará una profunda comprensión de los programas y corregirlos será una tarea mucho más fácil y rápida.

Referencias

[Bal93] José Luis Balcázar. *Programación Metódica.* Mc Graw Hill, 1993.

[Bis86] Judy Bishop. *Data Abstraction in Programming Languages.* International Computer Science Series-Addison Wesley, Great Britain, 1986.

[Gri81] David Gries. *The Science of Programming.* Springer Verlag, 1981.

[LG86] Barbara Liskov and John Guttag. *Abstraction and Specification in Program Development.* The MIT Press, Mc Graw Hill, London, England, 1986.

[Sch86] P. C. Scholl. *Algorítmica y Representación de Datos*, volume 2. Masson S.A., Barcelona, 1986.