Unidad 3 / Escenario 6

Lectura fundamental

Corrección de algoritmos

Contenido

Palabras claves:
Especificación de un algoritmo, tripla de hoare, correcci ón asignación, corrección ciclo.

Introducción

Cuando se diseña un algoritmo para ser ejecutado en un computador, una pregunta que surge es: ¿cómo garantizar que el programa realiza la tarea deseada? Es posible, que el lector considere que una solución sea *correr* el programa con varios ejemplos y observar que la respuesta sea la adecuada, pero este proceso no es totalmente confiable, dado que solo se está validando algunos casos y en la práctica el programa deberá abordar muchos más.

En términos generales, para determinar si un algoritmo realiza de forma *correcta* una tarea es necesario:

1. Especificar la tarea que debe realizar.
2. Verificar que el conjunto de instrucciones del algoritmo ejecuten la tarea.

Para realizar lo descrito en el item 2), se pretende visualizar un algoritmo como una "teorema" y construir una "demostración" de su validez. Para lo cual es necesario introducir nuevos conceptos, reglas lógicas de inferencia y emplear temas desarrollados en lecturas anteriores de este módulo.

1. Especificación de algoritmos

Desde un punto de vista abstracto y simple, un algoritmo se puede analizar como una función que transforma el estado de ciertas variables. Por ejemplo, observe el siguiente programa:

```
1 con A,B : Int;
2 var x,y,z : Int;
3 x:=A;
4 y:=B;
5 z:= x + y;
```

La instrucción **var** y **con** declaran constantes y variables del programa, el símbolo :Int indica que son datos de tipo entero (por lo tanto, el espacio de estados del programa es el conjunto $\mathbb{Z}^5$), las instrucciones de las líneas 3 y 4 realizan una asignación (cambian el estado) a las variables x, y, la instrucción de la línea 5 asigna el resultado de $x + y$ a la variable z (de nuevo es un cambio de estado).

Ante lo expuesto anterirmente, es natural preguntar ¿cuál es el estado inicial de las variables del programa?, es decir, antes de ejecutar el algoritmo. O ¿cuál es el estado final de las variables? luego de ejecutar el programa. En la práctica, el estado inicial y final de las variables se describe por medio de predicados sobre ellas. Regresando al ejemplo anterior, un predicado sobre el estado inicial de las variables es:

$$\{x = 0 \wedge y = 0 \wedge z = 0\}$$

y un predicado sobre el estado final de las variables es:

$$\{z = A + B\}$$

Ubicando esta información en el programa se obtiene la siguiente *especificación* del algoritmo presentado inicialmente:

$$
\begin{array}{l}
[\![\textbf{con } A, B : Int; \\
\quad \textbf{var } x, y, z : Int; \\
\quad \{x = 0 \wedge y = 0 \wedge z = 0\} \\
\quad x := A; \\
\quad y := B; \\
\quad z := x + y; \\
\quad \{z = A + B\} \\
]\!]
\end{array}
$$

Se han agregado los símbolos "|[" y "]|" para indicar que esta es una especificación de un programa. En resumen:

En resumen, la **especificación** de un algoritmo consiste en establecer:

- El espacio de estados: variables del programa (incluye las constantes).
- Un predicado sobre los valores iniciales de algunas variables del programa, que se denota por $\{P\}$ y se denomina la **precondición** del algoritmo.
- Un conjunto de instrucciones, en un lenguaje de programación, que se denota por S y corresponde al algoritmo.
- Un predicado sobre los valores finales de algunas variables del programa, que se denota por $\{Q\}$ y se denomina la **postcondición** del algoritmo.

La precondición de un algoritmo, se puede interpretar como una condición sobre los datos de entrada del algoritmo y la postcondición como una condición sobre los datos de salida del algoritmo. Pero ¿cuál es la relación de S con $\{P\}$ y $\{Q\}$?

Un algoritmo S *satisface* una especificación si todas las ejecuciones de S desde un estado que cumple la precondición $\{P\}$ termina en un estado que cumple la postcondición $\{Q\}$. En este caso, se dice que S es correcto (con relación a la especificación).

A continuación se presentan algunos ejemplos de especificación y se determinan sus componentes.

Ejemplo 1. La siguiente especificación es de un algoritmo para calcular el máximo de dos números enteros A y B que serán dados al programa (por consola, por ejemplo):

|[**var** x,y,z : Int;
$\{x = A, y = B\}$
if $x > y$ **then**
  z=x;
else
  z=y;
end
$\{z = \text{máx}\{A, B\}\}$
]|

En este caso el espacio de estados es $\mathbb{Z}^3$, la precondición $\{P\}$ es $\{x = A, y = B\}$, la postcondición $\{z = \text{máx}\{A, B\}\}$ y el algoritmo S es:

if $x > y$ **then**
  z=x;
else
  z=y;
end

Ejemplo 2. La siguiente especificación es de un algoritmo para calcular el residuo de dividir un entero no negativo A por un entero positivo B.

|[**con** A,B : Int;
 var y : Int;
$\{A \geq 0, B > 0\}$
y:=A;
while $y \geq B$ **do**
  y:=y-B;
end
$\{y = A \mod B\}$
]|

En este caso el espacio de estados es $\mathbb{Z}^3$, la precondición $\{P\}$ es $\{A \geq 0, B > 0\}$, la postcondición $\{y = A \mod B\}$ y el algoritmo S es:

while $y \geq B$ **do**
  y:=y-B;
end

De forma básica, se puede definir que una especificación sobre un algoritmo S consiste de una tripla:

$$\{P\}S\{Q\}$$

donde $\{P\}$ y $\{Q\}$ son predicados sobre el estado inicial y final de algunas variables del algoritmo respectivamente y S las instrucciones del algoritmo. Este tipo de triplas se conocen como triplas de **Hoare** y se dirá que la tripla es **válida** si S es correcto.

Ahora el paso siguiente es definir bajo qué condiciones una tripla es válida.

2. Correción de asignación y condicionales

Para presentar el método de verificación de un algoritmo, con relación a una especificación, inicialmente se supondrá que S corresponde a una sola instrucción de asignación o a un solo condicional, por ejemplo:

|[**var** x,y,z : Int;
$\{x = A, y = B\}$
if $x > y$ **then**
  z=x;
else
  z=y;
end
$\{z = \text{máx}\{A, B\}\}$
]|

En estos casos, la validez de la tripla de Hoare correspondiente, podrá ser comprobada a través de una de las siguientes reglas de inferencia:

Regla se asignación: si a partir de una condición P se puede concluir una condición Q donde se ha intercambiado la variable x por una expresión E, entonces la tripla $\{P\}x := E\{Q\}$ es válida. De forma simbólica:

$$\frac{P \Longrightarrow Q(x := E)}{\{P\}x := E\{Q\}}$$

Ejemplo 3. Compruebe la validez de la tripla $\{x > 1\}x := x + 1\{x > 0\}$

Solución: Como esta especificación consiste de la precondición $\{x > 1\}$, la postcondición $\{x > 0\}$ y algoritmo $x := x+1$ entonces aplicando la regla de la asignación para establecer la validez de la tripla es suficiente demostrar que $x > 1 \Longrightarrow (x > 0)(x := x + 1) \equiv x + 1 > 0$. Ahora:

$$\begin{aligned} & x > 1 \\ \equiv\ & x + 1 > 2 \quad \langle \text{ propiedad de la relación } < \rangle \\ \Rightarrow\ & x + 1 > 0 \quad \langle \text{ propiedad transitiva de la relación } < \rangle \\ \equiv\ & (x > 0)(x := x + 1) \quad \langle \text{ regla de sustitución } \rangle \end{aligned}$$

♢

Ejemplo 4. Compruebe la validez de la tripla $\{x = 4\}x := x * x + 1\{x = 17\}$

Solución: Como esta especificación consiste de la precondición $\{x = 4\}$, la postcondición $\{x = 17\}$ y algoritmo $x := x * x + 1$ entonces aplicando la regla de la asignación para establecer la validez de la tripla es suficiente demostrar que $x = 4 \Longrightarrow (x = 17)(x := x * x + 1)$.

Note que $(x = 17)(x := x * x + 1)$ corresponde a $x * x + 1 = 17$, por lo tanto lo que se demostrará es:

$$x = 4 \Longrightarrow x * x + 1 = 17.$$

$$
\begin{aligned}
& x = 4 \\
\Rightarrow\ & x * x = 16 \quad \langle \text{ aritmética } \rangle \\
\equiv\ & x * x + 1 = 17 \quad \langle \text{ propiedad de la } = \rangle
\end{aligned}
$$

♢

Regla de selección: si las triplas $\{P \wedge B\}S_0\{Q\}$ y $\{P \wedge \neg B\}S_1\{Q\}$ son válidas entonces la tripla

$$\{P\} \textbf{ if } B \textbf{ then } S_0 \textbf{ else } S_1 \textbf{ end}\{Q\}$$

es válida.

Ejemplo 5. Demostrar la validez de la tripla $\{x = A, y = B\}S\{z = \text{máx}\{A, B\}\}$ donde S es el algoritmo

```
if x > y then
  | z=x;
else
  | z=y;
end
```

Solución: Para verificar la correctitud de este condicional, se debe verificar la validez de las triplas:

- $\{x = A, y = B \wedge x > y\}\ z := x\ \{z = \text{máx}\{A, B\}\}$
- $\{x = A, y = B \wedge \neg(x > y)\}\ z := y\ \{z = \text{máx}\{A, B\}\}$

Ahora, para verificar la tripla $\{x = A, y = B \wedge x > y\}\ z := x\ \{z = \text{máx}\{A, B\}\}$ se hace uso de la regla de asignación, luego es necesario demostrar que

$$x = A, y = B \wedge x > y \Longrightarrow (z = \text{máx}\{A, B\})(z := x)$$

$$
\begin{aligned}
& x = A, y = B \wedge x > y \\
\Rightarrow\ & x = \text{máx}\{x, y\} \quad \langle x > y \rangle \\
\equiv\ & x = \text{máx}\{A, B\} \quad \langle x = A, y = B \rangle \\
\equiv\ & (z = \text{máx}\{A, B\})(z := x) \quad \langle \text{ regla se sustitución } \rangle
\end{aligned}
$$

Para verificar la tripla

$\{x = A, y = B \wedge \neg(x > y)\}\ z := y\ \{z = \text{máx}\{A, B\}\}$ se debe demostrar que:

$$x = A, y = B \wedge \neg(x > y) \Longrightarrow (z = \text{máx}\{A, B\})(z := y)$$

$$
\begin{aligned}
& x = A, y = B \wedge \neg(x > y) \\
\Rightarrow\ & y \geq x \quad \langle \text{ tricotomía de la relación } < \rangle \\
\equiv\ & y = \text{máx}\{x, y\} \quad \langle \text{ definición de máx} \rangle \\
\equiv\ & y = \text{máx}\{A, B\} \quad \langle x = A, y = B \rangle \\
\equiv\ & (z = \text{máx}\{A, B\})(z := y) \quad \langle \text{ regla se sustitución } \rangle
\end{aligned}
$$

♢

¿Qué sucede si en un algoritmo S aparecen varias instrucciones?, ¿cómo se puede realizar la verificación de la validez de la tripla $\{P\}S\{Q\}$?

Regla de la concatenación: si las triplas $\{P\}S_0\{Q_0\}$ y $\{Q_0\}S_1\{Q\}$ son válidas entonces la tripla

$$\{P\}S_0; S_1\{Q\}$$

es válida.

La expresión $S_0; S_1$ hace referencia a un algoritmo que contiene las instrucciones de S_0 y las de S_1, que ejecuta primero S_0 y luego S_1. La idea detrás de esta regla es que si el algoritmo S_1 transforma correctamente desde el estado inicial descrito por $\{P\}$ al estado descrito por $\{Q_0\}$ y S_1 transforma correctamente desde el estado $\{Q_0\}$ a el estado $\{Q\}$ entonces la concatenaci´on de los dos programas transforma correctamente desde un estado descrito por $\{P\}$ a un estado descrito por $\{Q\}$

Ejemplo 6. Demostrar que la tripla $\{x = A, y = B\}x := x + y;\ y := x - y;\ x := x - y\{x = B, y = A\}$ es válida.

Solución: Dado que el algoritmo S corresponde a varias instrucciones consecutivas, se hará uso de la regla de la concatenación y se demostrará la validez de las triplas:

- $\{x = A, y = B\}x := x + y\{x = A + B, y = B\}$
- $\{x = A + B, y = B\}y := x - y\{x = A + B, y = A\}$
- $\{x = A + B, y = A\}x := x - y\{x = B, y = A\}$

Note que la postcondición de una regla es la precondición de la siguiente. Como cada algoritmo en cada tripla corresponde a una asignación, se dejará al lector la validación de los tres programas.

Luego de verificar la correctitud de cada uno de los tres algoritmos, se puede concluir que la tripla

$$\{x = A, y = B\}x := x + y;\ y := x - y;\ x := x - y\{x = B, y = A\}$$

es válida. ♢

Una pregunta que surge después de analizar este ejemplo es cómo se pudo establecer que las condiciones intermedias $\{x = A + B, y = B\}, \{x = A + B, y = A\}$ eran las adecuadas para aplicar la regla de la concatenación. La forma de construirlas se sale de los objetivos de este módulo, pero serán abordadas en un módulo posterior. Aunque una manera básica es registrar el estado de las variables después de ejecutar cada instrucción.

3. Corrección de ciclos

Suponga que se tiene que validar una tripla de Hoare, de la forma

$$\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\{Q\}$$

es decir, verificar la correctitud de un ciclo. ¿Cómo asegurar que se alcanzará el estado $\{Q\}$, cuando finalice el ciclo? la solución es determinar una condición que describa el objetivo del ciclo, permanezca invariante en la ejecución de S y que al terminar el ciclo se pueda concluir $\{Q\}$.

Regla invariante de ciclo: si para un ciclo de la forma **while** B **do** S es posible hallar una condición I tal que la tripla $\{I \wedge B\}S\{I\}$ es válida, entonces la tripla $\{I\}$ **while** B **do** S $\{I \wedge \neg B\}$ es válida. De forma simbólica:

$$\frac{\{I \wedge B\}S\{I\}}{\{I\}\ \textbf{while}\ B\ \textbf{do}\ S\ \{I \wedge \neg B\}}$$

La condición I se denomina **invariante del ciclo**. Se debe aclarar de esta regla que la validez de la expresión $\{I\}$ **while** B **do** S $\{I \wedge \neg B\}$ solo indica que si el ciclo termina, entonces el programa alcanzará el estado $\{I \wedge \neg B\}$, nada asegura que el ciclo debe terminar, por lo tanto será necesaria una demostración "independiente" de este hecho.

Ejemplo 7. Demostrar que $\{x = 3i \wedge N \geq 0\}$ es un invariante del ciclo

```
while i ≤ N do
    x:=3+x;
    i:=i+1;
end
```

Solución: Para demostrar que $\{x = 3i \wedge N \geq 0\}$ es un invariante del ciclo, es necesario demostrar que la tripla:

$$\{x = 3i \wedge N \geq 0 \wedge i \leq N\}x := 3 + x;\ i := i + 1;\ \{x = 3i \wedge N \geq 0\}$$

Para ello se utiliza la regla de la concatenación y el hecho que

- $\{x = 3i \wedge N \geq 0 \wedge i \leq N\}x := 3 + x\{x = 3(i + 1) \wedge N \geq 0\}$
- $\{x = 3(i + 1) \wedge N \geq 0\}i := i + 1;\ \{x = 3i \wedge N \geq 0\}$

son triplas de Hoare válidas. ♢

Para demostrar que un ciclo termina, se puede hacer uso de la siguiente regla:

Regla de terminación de ciclo: si para un ciclo de la forma **while** B **do** S existe un *invariante* I y una función t entera definida sobre el espacio de estados tal que:

1. $I \wedge B \Rightarrow t \geq 0$
2. $\{I \wedge B \wedge t = C\}\ S\ \{t < C\}$ es una tripla válida.

Entonces el ciclo termina.

Ejemplo 8. Demostrar que termina el siguiente ciclo:

```
while i ≤ N do
    x:=3+x;
    i:=i+1;
end
```

Solución: La idea es construir una función t sobre las variables del programa, que su valor sea un número entero y que en cada iteración del ciclo el valor de la función t, suponiendo que al inicio del un ciclo tiene un valor de C. Para este ejercicio una función t que se podría tomar es

$$t : N - i$$

note que esta función decrece en cada iteración y de alguna forma está calculando las iteraciones faltantes para terminar la repetición del ciclo. Ahora, se demuestra los dos hechos mencionados en el criterio de terminación de ciclo.

- $I \wedge B \Rightarrow t \geq 0$:

  Para este ciclo el invariante calculado anteriormente fue $\{x = 3i \wedge N \geq 0\}$ por lo tanto lo que se desea demostrar es:

  $$x = 3i \wedge N \geq 0 \wedge i \leq N \Rightarrow N - i \geq 0$$

  $$\begin{aligned} & x = 3i \wedge N \geq 0 \wedge i \leq N \\ \Rightarrow\ & i \leq N \quad \langle \text{ propiedad de simplificación de } \wedge \rangle \\ \equiv\ & N - i \geq 0 \quad \langle \text{ propiedad de la relación } < \rangle \\ \equiv\ & t \geq 0 \quad \langle \text{ definición de } t \text{ y sustitución } \rangle \end{aligned}$$

- $\{I \wedge B \wedge t = C\}\ S\ \{t < C\}$:

  Para este ejercicio, la tripla corresponde a:

  $$\{x = 3i \wedge N \geq 0 \wedge i \leq N \wedge N - i = C\} x := x + 3;\ i := i + 1;\ \{N - i < C\}$$

  que para demostrar su validez, se debe emplear la regla de la concatenación y comprobar:

  - $\{x = 3i \wedge N \geq 0 \wedge i \leq N \wedge N - i = C\} x := x + 3 \{x = 3(i + 1) \wedge N \geq 0 \wedge i \leq N \wedge N - i = C\}$

- $\{x = 3(i+1) \wedge N \geq 0 \wedge i \leq N \wedge N - i = C\} i := i + 1 \{N - i < C\}$

El lector puede comprobar con facilidad cualquiera de las dos, aquí se expondrá la demostración de la validez de la segunda tripla:

$$
\begin{aligned}
& x = 3(i+1) \wedge N \geq 0 \wedge i \leq N \wedge N - i = C \\
\Rightarrow\ & N - i = C \quad \langle \text{ propiedad de simplificación de } \wedge \rangle \\
\equiv\ & N - i - 1 = C - 1 \quad \langle \text{ propiedad de la } = \rangle \\
\Rightarrow\ & N - i - 1 < C \quad \langle \text{ transitividad de } < \text{ y el hecho que } C - 1 < C \rangle \\
\equiv\ & (N - i < C)(i := i + 1) \quad \langle \text{ propiedad de sustitución } \rangle
\end{aligned}
$$

Por lo tanto el ciclo termina. ♢

Con los criterios de invarianza y terminación de ciclo. Se puede definir en este momento un procedimiento para demostrar cuando un algoritmo cuya especificación es de la forma:

$\{P\}$
H
{invariante: I, cota: t}
while B **do**
| S
end
$\{Q\}$

donde H y S es un conjunto de instrucciones en un lenguaje de programación, $\{P\}$ y $\{Q\}$ son la precondición y postcondición respectivamente.

Para demostrar la correctitud de un programa, cuya estructura corresponde a la presentada anteriormente, se debe:

1. Determinar un invariante $\{I\}$ para el ciclo.
2. Determinar una función t para comprobar que el ciclo termina.
3. Demostrar que $\{P\}H\{I\}$ es una tripla válida.
4. Demostrar que $I \wedge \neg B \Rightarrow Q$

Ejemplo 9. Demostrar la correctitud del siguiente algoritmo:

```
|[ var x,i: Int;
 con N: Int;
{N ≥ 0}
x:= 0;
i:= 0;
while i ≤ N do
  |  x:=3+x;
  |  i:=i+1;
end
{x ≥ 3(N + 1)}
]|
```

Solución: Los ítems 1. y 2. del proceso de verificación de un ciclo descritos anteriormente, fueron realizados en los ejemplos 7 y 8. Por lo tanto solo falta demostrar los items 3. y 4.

- $\{P\}H\{I\}$:

  Para este caso corresponde a demostrar que $\{N \geq 0\}x := 0; i := 0\{x = 3i \wedge N \geq 0\}$, pero por la regla de la concatenación se puede comprobar de forma inmediata.

- $I \wedge \neg B \Rightarrow Q$:

  En este caso, se traduce en demostrar que

$$x = 3i \wedge N \geq 0 \wedge \neg(i \geq N) \Rightarrow x \geq 3(N + 1)$$

$$\begin{aligned}
& x = 3i \wedge N \geq 0 \wedge \neg(i \geq N) \\
\Rightarrow\ & x = 3i \wedge i \geq N + 1 \quad \langle \text{ tricotomía de la relación } < \text{ y simpliciación de } \wedge \rangle \\
\Rightarrow\ & x = 3i \wedge 3i \geq 3(N + 1) \quad \langle \text{ propiedad de la } < \rangle \\
\Rightarrow\ & x \geq 3(N + 1) \quad \langle \text{ sustitución } \rangle
\end{aligned}$$

♢

4. Ejercicios

Los siguientes ejercicios tienen como objetivo que el estudiante afiance los conceptos presentados en la lectura, no se deben entregar al tutor del módulo.

1. Determinar si los siguientes programas son correctos:

a)

|[**var** x,y,N:Int;
{$N > 1$}
x:=N-1;
y:=x+2;
{$xy > 1$}
]|

b)

|[**var** x: Int;
con A: Int;
{$true$}
if $A \geq 0$ **then**
| x:=A;
else
| x:=-A;
end
{$x = |A|$}
]|

c)

⟦**var** $x, y, N : Int$;
{$N > 0$}
$x := 0$;
$y := N$;
{I :?, t :?}//proponga un invariante y cota para el ciclo
While $x < y$ **do**
$x := x + 1$;
$y := y - 1$;
end
{$y = ⟦N/2⟧$}
⟧

d)

|[**var** x,i: Int;
con N: Int;
{$N \geq 1$}
x:=1;
i:=0;
while $x < N$ **do**
| x:=2*x;
| i:=i+1;
end
{$i = ⟦\log_2(N)⟧$}
]|

2. Para cada una de las siguientes situaciones, diseñe un algoritmo (especificando su precondición y postconsidición) y luego demuestre la correctitud de los mismos:

   a) Dado un número entero positivo, determinar si el número es par.

   b) Dado tres números enteros positivos, determinar si uno de ellos es un número par.

   c) Dado dos números enteros positivos A, B, determinar A^B [su programa solo puede utilizar sumas, restas o multiplicaciones].

   d) Dados tres números reales, determinar cuál es el número mayor.

e) Dado un número entero positivo A, determinar $[\![log_3(A)]\!]$. [$[\![x]\!]$ es la función parte entera.]

f) Dados dos enteros positivos A y B, determinar el cociente de dividir A entre B [su programa solo puede utilizar sumas, restas o multiplicaciones].

Bibliografía

[1] Bohórquez, J. (2012) *Lógica y matemáticas discretas en la informática. El estilo calculatorio*. Escuela Colombiana de Ingeniería.

[2] Grimaldi, R. (1998) *Matemáticas discreta y combinatoria*. México: Addison-Wesley Iberoamericana.

[3] Hammack, R. (2013) *Book of proof*, second edition, Editor: Richard Hammack.

[4] Rosen, K.H. y Pérez, J.M. (2004) *Matemática discreta y sus aplicaciones*, Madrid: McGraw-Hill.

INFORMACIÓN TÉCNICA

Módulo: Elementos en Teoría de la Computación
Unidad 3: Introducción al análisis de algoritmos
Escenario 6: Introducción a la verificación de algoritmos

Autor: Diego Arévalo Ovalle

Asesor Pedagógico: Óscar Mauricio Salazar
Diseñador Gráfico: Diego Arévalo Ovalle
Asistente: Alejandra Morales