CAPÍTULO 13
La Programación Imperativa

Índice del Capítulo

13.1. Introducción

A partir de ahora cambiaremos el modelo de computación subyacente, y por lo tanto el formalismo para expresar los programas. Esto no significa que lo hecho hasta ahora no nos será útil en lo que sigue, ya que la experiencia adquirida en el cálculo de programas, y la consecuente habilidad para el manejo de fórmulas, seguirá siendo escencial en este caso, mostrando así que pese a sus diferencias, la programación funcional y la imperativa tienen varias cosas en común.

13.2. Especificaciones de programas

Los problemas que se presentan a un programador están, en general, expresados de manera informal, lo cual resulta demasiado impreciso a la hora de programar. Por eso es necesario precisar los problemas por medio de especificaciones formales. En la primera parte del curso se especificaba, usando lógica y un cálculo de funciones, el valor que debía calcularse. Este tipo de especificaciones era adecaudo para el desarrollo de programas funcionales. En el estilo de programación imperativa, la computación no se expresa como cálculo de valores sino como modificación de estados. Para desarrollar programas imperativos es por lo tanto deseable expresar las especificaciones como relaciones entre estados iniciales y finales.

Así es que un programa en el modelo de la programación imperativa estará especificado mediante:

- Una precondición: expresión booleana que describe los estados iniciales de las variables del programa, para los cuales se define el mismo.
- Una postcondición: expresión booleana que describe los estados finales del programa luego de su ejecución.

La notación que utilizaremos:

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

donde P y Q son predicados y S es un programa (o secuencia de comandos), tiene la siguiente interpretación:

> Siempre que se ejecuta S comenzando en un estado que satisface P, el programa termina y el estado final satisface Q.

P es llamada precondición de S, y Q su postcondición. La terna $\{P\}\ S\ \{Q\}$ se denomina tripleta de Hoare.

Derivar un programa imperativo consiste en encontrar S que satisfaga las especificación dada, es decir, consiste en encontrar S tal que, aplicado a un estado que satisfaga P, termine en un estado que satisfaga Q. Mientras que una verificación de que S es correcto (respecto a su especificación) consiste en probar que la tripleta $\{P\}\ S\ \{Q\}$ es verdadera.

Cuando querramos dar solo la especificación de un programa, no utilizaremos la tripleta $\{P\}\ S\ \{Q\}$, dado que esta notación no indica qué variables cambiaron en S. Por ejemplo, $\{true\}\ S\ \{x = y\}$, dice que S termina en un estado que satisface $x = y$, pero no cuáles variables entre x e y cambiaron luego de la ejecución.

Denotaremos formalmente una especificación como:

$$\{P\}\ x :=?\ \{Q\} \tag{13.1}$$

donde P es la precondición del programa, Q la postcondición, y x es la lista de variables que pueden cambiar de valor.

En los lenguajes imperativos, también es necesario explicitar el tipo de las variables y constantes del programa, lo cual se hace a través de una *declaración de variables y constantes*. Esto completa la especificación del programa a la vez que indica implícitamente las operaciones que se pueden realizar con las variables y constantes declaradas.

Como ejemplo consideremos la siguiente especificación de un programa que calcula el mayor múltiplo de 5 menor que n.

$|[$ **var** z : **Integer**;
const n : **Integer**;
$\{true\}$
$z :=?$
$\{z = (max\, i : 0 \leq i < n \wedge mod.i{,}5 = 0 : i)\}$
$]|$

Usamos $|[\ ]|$ para encerrar la declaración de variables y constantes junto con la terna de Hoare. La sintaxis que utilizaremos para esta declaración es la del lenguage Pascal.

En el ejemplo n es declarada como una constante porque es un valor que no cambia nunca en la ejecución del programa.

13.2.1. Representación de variables iniciales y finales

El siguiente programa:

$$x, y := y, x$$

intercambia los valores de las variables x e y. Para poder especificar este programa, necesitamos una forma de poder describir los valores iniciales de las variables x e y. Usaremos para ello dos *constantes de especificación* que llamamos X e Y. La especificación del programa, junto con el programa mismo sería:

$$\{x = X \wedge y = Y\}\ x, y := y, x\ \{x = Y \wedge y = X\}$$

Es importante notar que estas constantes no son constantes del programa , por lo tanto no pueden aparecer en sentencias del mismo. Sólo pueden ocurrir en los predicados que forman la pre y post condición. En estos predicados pueden aparecer tres tipos de nombres: las variables de especificación, que denotaremos con letras mayúsculas; las variables del programa y variables de cuantificación.

A continuación se verán algunos ejemplos de especificación de programas, usando la notación 13.1. En el capítulo siguiente obtendremos los programas que satisfacen las especificaciones dadas, por ahora sólo nos abocaremos a definir las pre y post condiciones.

(13.1) **Ejemplo.** Especificaremos un programa que dadas dos variables enteras no negativas a y b, calcula el producto de ambas, mediante los siguientes predicados:

$$|[\ \textbf{var}\ a, b, z : \textbf{Integer};\ \ \{a \geq 0 \wedge b \geq 0\}\ z :=?\ \{z = a * b\}\ ]|$$

El producto de a y b tiene que ser almacenado en alguna variable, elegimos z.

(13.2) **Ejemplo.** Especificaremos ahora un programa que dada una variable entera n, calcula la suma de los n primeros números naturales y almacena el resultado en n.

$$
\begin{array}{ll}
|[& \textbf{var } n : \textbf{Integer}; \\
& \{n \geq 0 \wedge n = N\} \\
& n :=? \\
& \{n = (\sum\, i : 0 < i < N : i)\} \\
]|
\end{array}
$$

(13.3) **Ejemplo.** Los arreglos (arrays en inglés) son estructuras de datos que proveen la mayoría de los lenguajes de programación imperativos. Un arreglo puede pensarse como una lista de tamaño fijo. Utilizaremos la siguiente notación para describir un arreglo de $n+1$ elementos: $b[0, \ldots, n]$. El acceso a los elementos es similar al de listas, se realiza mediante índices. Por ejemplo, $b[0]$ corresponde al primer elemento del arreglo b.

Especificaremos un programa que dado un entero no negativo n y un arreglo $b[0, \ldots, n]$, calcula el producto de los primeros $n+1$ elementos del arreglo.

$$
\begin{array}{ll}
|[& \textbf{const } n : \textbf{Integer}; \\
& \textbf{var } b : \textbf{array } [0 \ldots n] \textbf{ of Integer}; \\
& \{n \geq 0\} \\
& z :=? \\
& \{z = (\prod\, i : 0 \leq i \leq n : b[i])\} \\
]|
\end{array}
$$

(13.4) **Ejemplo.** Por último daremos la especificación de un programa que dados una variable entera n y un arreglo $b[0, \ldots, n]$ de enteros, ordena el arreglo de forma creciente.

$$
\begin{array}{ll}
|[& \textbf{const } n : \textbf{Integer}; \\
& \textbf{var } b : \textbf{array } [0 \ldots n] \textbf{ of Integer}; \\
& \{n \geq 0 \wedge b = B\} \\
& b :=? \\
& \{perm(b, B) \ \wedge \ (\forall\, i : 0 \leq i < n : b[i] \leq b[i+1])\} \\
]|
\end{array}
$$

donde $perm$ es un predicado que toma dos arreglos y determina si uno es una permutación del otro. Dejamos como ejercicio la definición de este predicado.

Antes de continuar con el desarrollo de programas, veremos algunas reglas para la correcta interpretación y utilización de tripletas de Hoare.

13.3. Leyes sobre tripletas

El primer teorema, conocido como *ley de exclusión de milagros*, expresa:

(13.5) Teorema.

$$\{P\}\ S\ \{false\} \equiv P \equiv false$$

Esta ley establece que para cualquier programa S, si se requiere que termine y que los estados finales satisfagan $false$, entonces ese programa no puede ejecutarse exitosamente para ningún estado inicial. O bien, leído de otra manera, no existe ningún estado tal que si se ejecuta un programa S comenzando en él se puede terminar en un estado que satisfaga $false$.

La expresión

$$\{P\}\ S\ \{true\}$$

indica que, cada vez que comienza en un estado que satisface P, la ejecución de S concluye en un estado que satisface el predicado $true$. Esto se interpreta diciendo que la ejecución de S *termina* cada vez que se comienza en un estado que satisface P.

Las leyes que siguen a continuación expresan el hecho de que una precondición puede ser fortalecida y una postcondición puede ser debilitada, lo cual formulamos de la siguiente manera:

(13.6) Teorema. Fortalecimiento de precondición.

$$\{P\}\ S\ \{Q\} \wedge (P_o \Rightarrow P) \Rightarrow \{P_o\}\ S\ \{Q\}$$

(13.7) Teorema. Debilitamiento de postcondición.

$$\{P\}\ S\ \{Q\} \wedge (Q \Rightarrow Q_o) \Rightarrow \{P\}\ S\ \{Q_o\}$$

Se dice que una expresión P_0 es más fuerte que P si y solo si $P_0 \Rightarrow P$. Recíprocamente, P_0 es más débil que P si y solo si $P \Rightarrow P_0$. Con lo cual, la primera regla dice que si P_0 es más fuerte que P y vale $\{P\}\ S\ \{Q\}$, entonces $\{P_0\}\ S\ \{Q\}$ también es válida.

La segunda regla dice que si Q_0 es más débil que Q y vale $\{P\}\ S\ \{Q\}$, entonces también vale $\{P\}\ S\ \{Q_0\}$.

Utilizaremos los dos teoremas anteriores para demostrar que ciertas tripletas son verdaderas.

(13.8) Ejemplo. Probaremos que la tripleta $\{x = 1\}\ x := x + 1\ \{x > 0\}$ es cierta. Para ello supondremos válida la tripleta $\{x \geq 0\}\ x := x + 1\ \{x > 0\}$ (en el capítulo siguiente veremos cómo demostrar su validez).

Dado que la primer tripleta tiene una precondición más fuerte que la segunda y siendo esta la única diferencia, podemos probar que la tripleta es cierta de la siguiente manera:

$$\begin{array}{ll} & \{x = 1\}\ x := x + 1\ \{x > 0\} \\ \Leftarrow & \quad \langle \text{Fortalecimiento de precondición} \rangle \\ & \{x \geq 0\}\ x := x + 1\ \{x > 0\} \wedge (x = 1 \Rightarrow x \geq 0) \\ = & \quad \langle \text{supuesto; aritmética; neutro de } \wedge \rangle \\ & true \end{array}$$

(13.9) Ejemplo. Ahora veremos un ejemplo donde probamos que una tripleta es cierta debilitando su postcondición. Supondremos válida la tripleta del ejemplo anterior y probaremos $\{x \geq 0\}\ x := x + 1\ \{x \geq 0\}$.

$$
\begin{array}{ll}
 & \{x \geq 0\}\ x := x + 1\ \{x \geq 0\} \\
\Leftarrow & \quad \langle \text{Debilitamiento de postcondición} \rangle \\
 & \{x \geq 0\}\ x := x + 1\ \{x > 0\} \wedge (x > 0 \Rightarrow x \geq 0) \\
= & \quad \langle \text{supuesto; aritmética; neutro de } \wedge \rangle \\
 & true
\end{array}
$$

Supongamos que las tripletas $\{P\}\ S\ \{Q\}$ y $\{P\}\ S\ \{R\}$ son válidas. Entonces la ejecución de S en un estado que satisface P termina en uno que satisface Q y R, por lo tanto termina en un estado que satisface $Q \wedge R$. Esta ley se expresa de la siguiente manera:

(13.10) Teorema.

$$\{P\}\ S\ \{Q\} \wedge \{P\}\ S\ \{R\} \equiv \{P\}\ S\ \{Q \wedge R\}$$

Por último, la ley que sigue

(13.11) Teorema.

$$\{P\}\ S\ \{Q\} \wedge \{R\}\ S\ \{Q\} \equiv \{P \vee R\}\ S\ \{Q\}$$

13.4. El transformador de predicados wp

Para cada comando S se puede definir una función $wp.S : Predicados \rightarrow Predicados$ tal que si Q es un predicado, $wp.S.Q$ representa el predicado P más débil para el cual vale $\{P\}\ S\ \{Q\}$. Para cada Q, $wp.S.Q$ se denomina precondición más débil de S con respecto a Q. En otras palabra, para todo predicado Q, $wp.S.Q = P$ si y solo si $\{P\}\ S\ \{Q\}$ y para cualquier predicado P_0 tal que $\{P_0\}\ S\ \{Q\}$ vale $P_0 \Rightarrow P$ (P es más débil que P_0).

La definición de $wp.S$ permite relacionar las expresiones $\{P\}\ S\ \{Q\}$ y $wp.S.Q$:

(13.12) Teorema. Relación entre Terna de Hoare y wp.

$$\{P\}\ S\,\{Q\}\ \equiv\ P \Rightarrow wp.S.Q$$

13.5. Ejercicios

13.1 Escribir especificaciones para los programas que realizan las siguientes actividades:

a) Dadas dos variables enteras, copia el valor de la primera en la segunda.

b) Almacena en la variable x el máximo valor del arreglo de enteros $b[0, \ldots, n]$.

c) Dada una variable entera, almacena en ella su valor absoluto.

d) Dadas tres variables enteras, devuelve el máximo de las tres.

e) Dada una variable entera x, cuyo valor es mayor a 1, determina si x almacena un número primo.

f) Dada una variable entera $n \geq 0$ y un arreglo $b[0, \ldots, n]$, almancena en cada posición del arreglo la suma de todos los elementos del mismo.

13.2 Dar el significado operativo de las tripletas: $\{true\}\ S\ \{true\}$ y $\{false\}\ S\ \{true\}$

13.3 Deducir a partir de las leyes dadas en la sección 13.3 que

$$\{P_0\}\ S\ \{Q_0\} \quad \text{y} \quad \{P_1\}\ S\ \{Q_1\}$$

implican

$$\{P_0 \wedge P_1\}\ S\ \{Q_0 \wedge Q_1\} \quad \text{y} \quad \{P_0 \vee P_1\}\ S\ \{Q_0 \vee Q_1\}$$

13.4 Demostrar que $\{false\}\ S\ \{P\}$ es válida para cualquier S y cualquier P.

13.5 Demostrar que las leyes vistas en la sección 13.3, siguen de las siguientes leyes para $wp.S$:

a) $wp.S.false \equiv false$

b) $wp.S.Q \wedge wp.S.R \equiv wp.S.(Q \wedge R)$

c) $wp.S.Q \vee wp.S.R \Rightarrow wp.S.(Q \vee R)$