Programación

Asignatura 240205 Programación. Curso 2020–2021

Lectura[3]: **Predicados y cuantificadores**

Dr. José Ramón González de Mendívil
mendivil@unavarra.es
Departamento de Estadística, Informática y Matemáticas
Edificio Las Encinas
Universidad Pública de Navarra

Resumen

En esta Lectura presentamos los predicados, las operaciones entre predicados y la operación de sustitución de variables por expresiones en predicados. Como aplicación de la operación de sustitución se introduce el Axioma de las sentencias de asignación que es un elemento muy importante en la corrección de programas. De hecho, las sentencias de asignación son las únicas sentencias que 'manipulan' los valores de las variables para obtener resultados en los programas que diseñamos. Por ese motivo, es fundamental comprender el Axioma de la asignación. Finalmente, se introducen las nociones básicas sobre cuantificadores y diferentes ejemplos para comprender su uso.

Índice

1. Introducción **2**

2. Predicados **4**

3. Conjunto de estados de un predicado **7**

4. Sustitución de variables por expresiones en predicados **7**

5. Axioma de la sentencia de asignación **8**

6. Axioma de la sentencia continuar **9**

7. Cuantificadores **10**
7.1. Cuantificador universal y existencial 10
7.2. Cuantificadores para sumas y productos 12
7.3. Otros cuantificadores 13

Bibliografía **14**

A. Resumen: Equivalencias entre predicados **15**

B. Resumen: Cuantificador universal y existencial **16**

C. Resumen: Otras cuantificaciones **17**

1. Introducción

Una forma muy extendida de ver los programas (y la labor del programador) es considerarlos como meras recetas que indican los pasos que hay que seguir para realizar un determinado propósito. Las explicaciones que se ofrecen en muchos libros se presentan de manera operacional, es decir, indicando cómo funcionan los programas. Esto sucede amenudo con los libros de texto que enseñan distintos lenguajes de programación. En la asignatura de Programación obligamos a que los programas que realicemos vengan acompañados de la demostración de que, efectivamente, son correctos: cumplen el propósito para el que han sido diseñados. Pero es justamente comprender cómo se demuestra la corrección de las sentencias de un programa lo que nos permite obtener un método constructivo para el diseño de los mismos. Esto no resta capacidad a los programadores a la hora de plasmar sus ideas e intuiciones para obtener un programa, al contrario, las ideas de diseño de un programa y el trabajo de corrección se ayudan mutuamente para obtener 'buenas' soluciones. Considera el siguiente programa,

```
1:  algoritmo convertir
2:      var
3:          s, m, h : entero
4:      fvar
5:      Requisito: La cantidad de segundos en s tiene que ser positiva
6:      leer(s);                                  ▷ lee una cantidad de segundos
7:      h := s div 3600;
8:      s := s mod 3600;
9:      m := s div 60;
10:     s := s mod 60
11:     escribir(s, m, h)                         ▷ s segundos, m minutos, h horas
12: falgoritmo
```

Cuando sunministras un valor A (entero positivo) como entrada al programa, la ejecución de la línea 6, **leer**(s), define un estado inicial $\langle A, -, - \rangle$, o como hemos indicado en la Lectura[2], $\{s = A \wedge s > 0\}$. La ejecución de las asignaciones en las líneas 7 y 8, hacen que antes de ejecutar la asignación en la línea 9, obtengas el estado $\{h = A \text{ div } 3600 \wedge s = A \text{ mod } 3600\}$. A partir de este estado como $s = A \text{ mod } 3600$, la ejecución de las asignaciones 9 y 10, te conducen al estado final $\{h = A \text{ div } 3600 \wedge m = (A \text{ mod } 3600) \text{ div } 60 \wedge s = (A \text{ mod } 3600) \text{ mod } 60\}$.

No es difícil demostrar, usando el teorema de la división entera, que el estado final es equivalente a $\{A = h \cdot 3600 + m \cdot 60 + s \wedge 0 \leq m < 60 \wedge 0 \leq s < 60\}$. Así pues, cuando finalmente se ejecuta la línea 11, **escribir**$(s,\ m,\ h)$, los valores que se muestran son la conversión de la cantidad suministrada inicialmente A, en segundos, minutos y horas respectivamente. Valores que se han obtenido en las variables s, m y h respectivamente.

Lo esencial del programa anterior no reside de dónde tomamos los datos o dónde los escribimos; lo esencial es que, si comenzamos en un estado inicial en el conjunto $\{s = A \wedge s > 0\}$, al ejecutar las sentencias del programa llegamos al estado final $\{A = h \cdot 3600 + m \cdot 60 + s \wedge 0 \leq m < 60 \wedge 0 \leq s < 60\}$. Esta relación entre los estados iniciales y los estados finales de la ejecución de las sentencias lo escribimos como,

```
1:  algoritmo convertir
2:      var
3:          s, m, h : entero
4:      fvar
5:      {s = A ∧ s > 0}
6:      h := s div 3600;
```

7: $s := s \bmod 3600$;
8: $m := s \text{ div } 60$;
9: $s := s \bmod 60$
10: $\{A = h \cdot 3600 + m \cdot 60 + s \wedge 0 \leq m < 60 \wedge 0 \leq s < 60\}$
11: **falgoritmo**

La interpretación operacional del programa anterior es,

Para cualquier valor de A, si la ejecución de las sentencias 7, 8, 9 y 10 del programa comienza en un estado que cumple $s = A \wedge s > 0$*, el programa termina, y termina en un estado que cumple* $A = h \cdot 3600 + m \cdot 60 + s \wedge 0 \leq m < 60 \wedge 0 \leq s < 60$.

Al primer predicado le denominamos **Precondición** y al predicado al final le denominamos **Postcondición**. Como hemos indicado, no nos importa cómo se establece el estado incial que cumple la precondición. No consideramos cómo es la entrada/salida de los datos. Nos debemos centrar en diseñar programas que expresan la computación realizada para obtener el estado final (de salida) en términos de el estado inicial (de entrada). Tampoco nos preocupamos por no usar un lenguaje de programación convencional (Pascal, C, Java, Phyton, etc...) para escribir las sentencias de los programas. Usamos un pesudocódigo para que los algoritmos se expresen de la forma más natural posible a nuestro entendimiento. El lenguaje concreto no es el objeto de las clases de teoría pero sí de las clases prácticas donde codificamos los algoritmos.

Si el programa anterior era la solución a un problema, ¿cuál era el problema?. Los **problemas algorítmicos**, aquellos que resolvemos mediante programas, se determinan mendiante **especificaciones**. Básicamente una especificación tiene la misma 'forma' que un programa pero sin sentencias. Sólo indica la relación de los estados iniciales a la entrada y los estados finales a la salida de cualquier posible programa que pretenda ser una solución al problema. La especificación del programa anterior sería,

1: **var**
2: s, m, h : entero ▷ declaración de variables
3: **fvar**
4: $\{s = A \wedge s > 0\}$ ▷ Precondición
5: 'convertir' ▷ nombre del programa que queremos diseñar
6: $\{A = h \cdot 3600 + m \cdot 60 + s \wedge 0 \leq m < 60 \wedge 0 \leq s < 60\}$ ▷ Postcondición

La especificación contiene la declaración de variables que define el conjunto de estados posibles, la precondición y la postcondición, y el nombre del programa que queremos diseñar.

En general, cuando encontramos unas sentencias S que cumplen con la una determinada especificación, con precondición P y postcondición Q, para una declaración de variable dada, escribimos,

$\{P\}\, S\, \{Q\}$ se cumple (S satisface la especificación $\{P\} - \{Q\}$).

El papel que juegan los predicados en todo este proceso de especificar problemas algorítmicos y de diseñar 'programas correctos' es muy importante y debemos dedicar un tiempo a su estudio.

Nota: las expresiones booleanas que vimos en la Lectura[2], son un tipo de predicados pero, para obtener toda la capacidad que ofrencen los predicados, es necesario definir nuevas expresiones. Las expresiones booleanas de la Lectura[2] se pueden programar en sentencias de los programas. Las nuevas expresiones

que vamos a presentar mediante cuantificadores no se pueden programar directamente, y serán objeto de programas que realizaremos a lo largo del curso.

2. Predicados

El estudiante sabe reconocer que una expresión como $x^2+2\cdot x+1$ es una función polinómica de grado dos. Se suele indicar, sea $p(x)$ la función $x^2+2\cdot x+1$. Es decir, $p(x)$ es una función, $p:\mathbb{R}\to\mathbb{R}$. A cada valor v de $\mathbb{R}$, la función $p(x)$ le hace corresponder un único valor de $\mathbb{R}$, calculado como $v^2+2\cdot v+1$. Las funciones no están limitadas a tener una única variable. Por ejemplo, la función $g(x,y)\stackrel{\text{def}}{=}x^2+y^2$ es una función multivariable, $g:\mathbb{R}\times\mathbb{R}\to\mathbb{R}$. Así, $g(-2,3)$ toma el valor 13, según la definición de la función. En otras ocasiones, el estudiante habrá escrito $x^2+2\cdot x+1=0$ para calcular las raíces del polinomio $p(x)$. Ciertos valores de $\mathbb{R}$ harán que la igualdad indicada, $x^2+2\cdot x+1=0$, se cumpla (la igualdad es verdadera), y esos valores son las raíces del polinomio. Otros valores distintos a las raices harán que la igualdad anterior no se cumpla (la igualdad es falsa). Así, que esta expresión $x^2+2\cdot x+1=0$ se puede entender como una nueva función que va de $\mathbb{R}\to\{V,F\}$. Los mismo sucede con una expresión como $x\geq y$ siendo x, y variables enteras. Sea $P(x,y)$ la función $x\geq y$. Ésta es una función $P:\mathbb{Z}\times\mathbb{Z}\to\{V,F\}$. Si calculas $P(2,3)$, $2\geq 3$, el resultado es F (falso).

Para una declaración de variables dada (Lectura[2]) sea Γ su 'espacio de estados'[1]. Un **predicado** P es una función $P:\Gamma\to\{V,F\}$. Sea la declaración, **var** x, y: entero **fvar**. Ejemplos de predicados:

- $x\geq y$
- $x=3\wedge y=4$
- $x^2+y^2=13$

Al evaluar estos predicados en el estado $\langle\sigma_x,\sigma_y\rangle=\langle 3,2\rangle$, el primero resulta cierto, el segundo falso y el tercero cierto.

En la Lectura[2], al estudiar las expresiones booleanas, se introdujeron las operaciones $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\equiv$ sobre el conjunto $\{V,F\}$ mediante las tablas de las operaciones (similar a las tablas de multiplicación que estudiabamos cuando niños). Ahora, extendemos estas operaciones sobre los predicados. Para cualesquiera predicados P y Q, y para cualquier estado $\sigma\in\Gamma$,

1. $\wedge$ (conjunción). $P\wedge Q$ es el predicado tal que es cierto en todos los estados en los que P y Q son ciertos, y falso en los otros estados. Es decir, $(P\wedge Q)(\sigma)\stackrel{\text{def}}{=}P(\sigma)\wedge Q(\sigma)$.
2. $\vee$ (disyunción). $P\vee Q$ es el predicado tal que es falso en todos los estados en los que P y Q son falsos, y cierto en los otros estados. Es decir, $(P\vee Q)(\sigma)\stackrel{\text{def}}{=}P(\sigma)\vee Q(\sigma)$.
3. $\equiv$ (igualdad lógica, equivalencia). $P\equiv Q$ es el predicado tal que es cierto en todos los estados en los que P y Q toman el mismo valor, y falso en los otros estados. Es decir, $(P\equiv Q)(\sigma)\stackrel{\text{def}}{=}P(\sigma)\equiv Q(\sigma)$.
4. $\Rightarrow$ (implicación). $P\Rightarrow Q$ es el predicado tal que es falso en todos los estados en los que P es cierto y Q es falso, y cierto en los otros estados. Es decir, $(P\Rightarrow Q)(\sigma)\stackrel{\text{def}}{=}P(\sigma)\Rightarrow Q(\sigma)$.
5. $\neg$ (negación). $\neg P$ es el predicado tal que es cierto en todos los estados donde P es falso, y falso en todos los estados donde P es cierto. Es decir, $(\neg P)(\sigma)\stackrel{\text{def}}{=}\neg(P(\sigma))$.

[1] Recuerda que el espacio de estados Γ es el producto cartesiano de los dominios de las variables declaradas y que las coordenadas de una tupla $\sigma=\langle....\rangle\in\Gamma$ se identifican con los nombres de las variables.

Para evitar algunos paréntesis, la negación tiene la mayor prioridad, seguido de la conjunción y la disyunción, luego la implicación, y la operación menos prioritaria es la igualdad lógica. Por comodidad, el predicado constante que es cierto en cada estado, se denomina también *cierto*; y el predicado costante que es falso en cada estado, se denomina *falso*. Algunos predicados al evaluarlos en cualquier estado resultan ser siempre ciertos. Ejemplos: $(x+1)^2 = x^2+2x+1$, $P \equiv P$, *cierto*. El primer ejemplo, nos permite 'sustituir' la expresión $(x+1)^2$ por su expresión equivalente x^2+2x+1 debido a que 'siempre' se cumple la igualdad. La proposición[2], '**el predicado P es cierto en todos los estados**', la denotamos como $[[P]]$.

Ejemplos:

- $[[(x+1)^2 = x^2+2x+1]]$ se cumple, ya que operando, $(x+1)^2 = (x+1)(x+1) = x^2+2x+1$ para cualquier x.
- $[[x \geq 1 \Rightarrow x \geq 0]]$ se cumple. Para los valores de x, donde $x \geq 1$ es cierto, como $1 > 0$, es cierto que $x \geq 0$, luego $x \geq 1 \Rightarrow x \geq 0$ se cumple para dichos valores. Para los valores de x donde $x \geq 1$ es falso, la implicación es cierta.
- $[[x \geq 0 \Rightarrow x \geq 1]]$ no se cumple. Si x se sustituye por el valor 4, la implicación es cierta, pero si x se sustituye por 0, la implicación es falsa.

Al demostrar para dos predicados dados que $[[P \equiv Q]]$ se cumple, $P(\sigma) \equiv Q(\sigma)$ es cierto para todo estado σ, entonces, representan la misma 'función' de verdad. Así, en cualquier otro predicado donde aparece P lo podemos sustituir por Q y *viceversa*, sin cambiar su significado. Esta sustitución se conoce como la **Regla de Leibniz**:

> Si $[[P \equiv Q]]$ entonces cualquier ocurrencia de P en un predicado R puede ser sustituida por Q sin modificar el resultado de la evaluación original de R en cualquier estado. Es decir, si $[[P \equiv Q]]$ entonces $[[R \equiv R(P \leftarrow Q)]]$.

En lo que sigue, vamos a mostrar una lista de propiedades que son muy comunes[3]:

Idempotencia:

- $[[P \wedge P \equiv P]]$
- $[[P \vee P \equiv P]]$

Commutatividad:

- $[[P \wedge Q \equiv Q \wedge P]]$
- $[[P \vee Q \equiv Q \vee P]]$
- $[[(P \equiv Q) \equiv (Q \equiv P)]]$

Asociatividad:

- $[[(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)]]$
- $[[(P \vee Q) \vee R \equiv P \vee (Q \vee R)]]$
- $[[((P \equiv Q) \equiv R) \equiv (P \equiv (Q \equiv R))]]$

Distributividad:

[2] Al ser una proposición podrá ser cierta o no, hay que demostrar su certeza o su falsedad.
[3] Se asume que ya están demostradas. Las demostraciones son fáciles empleando tablas de verdad.

- $[[P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)]]$
- $[[P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)]]$
- $[[P \vee (Q \equiv R) \equiv P \vee Q \equiv P \vee R]]$

Absorción:

- $[[P \wedge (P \vee R) \equiv P]]$
- $[[P \vee (P \wedge R) \equiv P]]$

Reglas cierto-falso:

- $[[P \wedge cierto \equiv P]] \quad [[P \wedge falso \equiv falso]]$
- $[[P \vee falso \equiv P]] \quad [[P \vee cierto \equiv cierto]]$

De Morgan:

- $[[\neg(P \wedge Q) \equiv \neg P \vee \neg Q]] \quad [[\neg(P \vee Q) \equiv \neg P \wedge \neg Q]]$

Negación:

- $[[\neg\neg P \equiv P]]$
- $[[P \vee \neg P \equiv cierto]]$
- $[[P \wedge \neg P \equiv falso]]$

Implicación:

- $[[P \Rightarrow Q \equiv \neg P \vee Q]]$
- $[[(P \Rightarrow Q) \wedge (Q \Rightarrow P) \equiv P \equiv Q]]$
- $[[P \Rightarrow P \vee Q]] \quad [[P \wedge Q \Rightarrow P]]$
- $[[falso \Rightarrow P]] \quad [[P \Rightarrow cierto]]$
- $[[cierto \Rightarrow P \equiv P]] \quad [[P \Rightarrow falso \equiv \neg P]]$

Equivalencia:

- $[[P \equiv P]]$
- $[[P \equiv P \equiv cierto]]$
- $[[\neg P \equiv P \equiv falso]]$

Ejemplo: Como ejemplo del uso de algunas de las equivalencias anteriores vamos a demostrar que
$[[P \wedge (\neg P \vee Q) \equiv P \wedge Q]]$ se cumple:

$$
\begin{array}{l}
P \wedge (\neg P \vee Q) \\
\equiv \text{ distributibidad de } \wedge \text{ sobre } \vee \\
(P \wedge \neg P) \vee (P \wedge Q) \\
\equiv \text{ regla de negación} \\
falso \vee (P \wedge Q) \\
\equiv \text{ reglas cierto-falso} \\
P \wedge Q
\end{array}
$$

3. Conjunto de estados de un predicado

Dada una declaración de variables y su conjunto de estados Γ, y dado un predicado P sobre dichas variables, denotamos por $\{P\}$ el subconjunto de estados definido como

$$\{P\} \stackrel{\text{def}}{=} \{\sigma \in \Gamma \,|\, P(\sigma) \equiv V\} \tag{1}$$

Esta visión entre predicados y los subconjuntos de estados que determinan, es útil en Programación, ya permite dotar de 'semantica' a las sentencias de los programas. La ejecución de determinada sentencia en un programa se realiza sobre un estado para generar un nuevo estado, pero el estado antes de ejecutar la sentencia, normalmente, no es arbitrario, sino que cumple unas propiedades (determinadas por un predicado) y por tanto, el estado pertenece a un conjunto. Lo mismo pasa con el estado que obtienes despúes de ejecutar la sentencia. Se puede decir que una sentencia 'transforma' un subconjunto de estados en otro subconjunto de estados.

$\sigma_i \in \{P\}$
S ejecutar la sentencia desde el estado σ_i
$\sigma_f \in \{Q\}$ estado obtenido después de ejecutar la sentencia

En lo que sigue vamos a definir cuándo un predicado es **más débil** (o **más fuerte**) que otro predicado. Si dos predicados P y Q cumplen $[[P \equiv Q]]$ entonces $\{P\} = \{Q\}$, es decir, sus subconjuntos de estados son los mismos y tienen las mismas propiedades. ¿Qué sucede si se cumple $[[P \Rightarrow Q]]$?. Todo estado $\sigma \in \{P\}$ es tal que $P(\sigma)$ es cierto, por tanto como $[[P \Rightarrow Q]]$, entonces, necesariamente, $Q(\sigma)$ es cierto, lo que indica que $\sigma \in \{Q\}$. Concluimos que todo estado de P es un estado de Q, es decir que $\{P\} \subseteq \{Q\}$.

Ejemplos, sea x de tipo entero:

- $[[x \geq 2 \Rightarrow x \geq 1]]$, $\{x \geq 2\} = \{2, 3, ...\}$, $\{x \geq 1\} = \{1, 2, 3, ...\}$, $\{x \geq 2\} \subseteq \{x \geq 1\}$.
- $[[x \geq 0 \Rightarrow x^2 \geq 0]]$, $\{x \geq 0\} = \{0, 1, 2, 3, ...\}$, $\{x^2 \geq 0\} = \{..., -2, -1, 0, 1, 2, ...\} = \mathbb{Z}$, $\{x \geq 0\} \subseteq \{x^2 \geq 0\}$.

Cuando se cumple $[[P \Rightarrow Q]]$, o equivalente $\{P\} \subseteq \{Q\}$, decimos que P es **más fuerte** que Q y que Q es **más débil** que P.

Ejercicio 1 *Relaciona los siguientes predicados entre sí, según la relación 'más fuerte que':*

$P1$*:* $x > 0$
$P2$*:* $x > 0 \wedge y > 0$
$P3$*:* $x > 0 \vee y > 0$
$P4$*:* $y \geq 0$
$P5$*:* $x \geq 0 \wedge y \geq 0$

4. Sustitución de variables por expresiones en predicados

Cuando declaramos las variables, los predicados que formamos hacen referencia a esas variables y a expresiones que podemos formar con ellas. Ejemplo, para la declaración **var** x, y: entero **fvar**, un predicado puede ser $x^2 + y^2 \leq 13$. Una operación importante que podemos realizar es la **sustitución** de variables por expresiones del mismo tipo que la variable. Dada x de un determinado tipo y dada una expresión Exp del mismo tipo, la sustitución de x por la expresión Exp en un predicado Q se denota como,

$$Q(x := Exp)$$

Otras formas que usamos para denotar la sustitución es $(Q)^x_{Exp}$ o también $Q(x \leftarrow Exp)$. No importa mucho como lo denotemos, lo importante es que $Q(x := Exp)$ es un nuevo predicado '*Q en el que x se sustituye por Exp*'[4]. La operación de sustitución es más prioritaria que cualquier otra operación sobre predicados y distribuye sobre todas las operaciones excepto otra sustitución. Ejemplos,

- $[[(x^2 + 2x)(x := x + 1) \equiv (x + 1)^2 + 2(x + 1)]]$
- $[[(x \geq y)(x := x + 1) \equiv x + 1 \geq y]]$
- $[[(P(x := y))(x := y) \equiv P(x := y)]]$
- $[[(P(x := y))(y := x) \equiv P(y := x)]]$
- $[[(P \wedge Q)(x := Exp) \equiv P(x := Exp) \wedge Q(x := Exp)]]$

5. Axioma de la sentencia de asignación

Los programas que hemos visto en la lecturas anteriores y en la sección 1 de está lectura continenen varias sentencias, y entre ellas, sentencias de asignación. Tan legitimo es un programa que contenga varias sentencias como el formado por una única sentencia de asignación. Los programas tienen que ser correctos y en esta sección analizamos cuáles son los programas que son 'siempre' correctos usando una única sentencia de asignación.

Cualquier cambio de estado que ocurre durante la ejecución de un programa se debe a la ejecución de una sentencia de asignación debido a que es la única sentencia que tiene la capacidad de modificar el valor de una variable (la variable asignada). En general, la sentencia de asignación tiene la forma $x := Exp$. Para un tipo de variable x y una expresión Exp del mismo tipo que la variable, la ejecución de la sentencia calcula primero la expresión Exp evaluada en el estado anterior a la ejecución σ_{ant} y reemplaza el valor de x por el valor de la expresión Exp para formar el nuevo estado σ_{post}. Si después de la ejecución de $x := Exp$, en el nuevo estado σ_{post} obtenido, el valor de la variable x en dicho estado cumple las propiedades dadas por un predicado Q necesariamente, el valor de la expresión Exp calculada en σ_{ant} tenía exáctamente las mismas propiedades.

Así que, para cualquier predicado Q, si Q se cumple después de ejecutar $x := Exp$ entonces $Q(x := Exp)$ se cumple antes de su ejecución. Por otra parte, como se ha ejecutado la evaluación de la expresión sin error, eso significa que cumple las condiciones de que la expresión está bien definida antes de la ejecución, es decir, también se cumple def(Exp). En conclusión para cualquier Q, el siguiente programa (o fragmento de programa) donde se identifica la sentencia de asignación, es 'siempre' correcto:

1: **var**
2: x: tipo base
3: .....
4: **fvar**
5: .....
6: $\{Q(x := Exp) \wedge \text{def}(Exp)\}$
7: $x := Exp$
8: $\{Q\}$
9: .....

Por tanto,

[4] **Toda aparición** de x en Q se sustituye por Exp!!.

Axioma de la asignación:
Para cualquier sentencia de asignación $x{:=}Exp$ sintácticamente correcta y cualquier predicado Q:

$$\{Q(x := Exp) \wedge \mathrm{def}(Exp)\}\ x := Exp\ \{Q\} \text{ se cumple.} \tag{2}$$

Ejercicio 2 *Calcula mediante el axioma de la asignación la precondición para que los siguientes programas basados en una sentencia de asignación sean correctos. Comprueba mediante algunos ejemplos que las ejecuciones son correctas. Se supone, en cada ejercicio, que las variables han sido correctamente declaradas.*

1. $\{P\}\, x := x + 1\, \{x > 0\}$
2. $\{P\}\, x := x + 1\, \{x^3 - 5x^2 + 2x > 0\}$
3. $\{P\}\, x := x + 1\, \{x = x + 1\}$
4. $\{P\}\, x := x\ mod\ 2\, \{x = x\ mod\ 2\}$
5. $\{P\}\, x := x/y\, \{x = A/B \wedge y = B\}$
6. $\{P\}\, a := \neg a \vee b\, \{a \vee b\}$

Nota: En algún ejemplo hemos empleado también la sentencia de asignación múltiple $\langle x, y\rangle := \langle Exp_1, Exp_2\rangle$. Su Axioma es similar al dado anteriormente:

Axioma de la asignación múltiple:
Para cualquier sentencia de asignación múltiple $\langle x, y\rangle := \langle Exp_1, Exp_2\rangle$ sintácticamente correcta y cualquier predicado Q:

$$\{Q(x \leftarrow Exp_1, y \leftarrow Exp_2) \wedge \mathrm{def}(Exp_1) \wedge \mathrm{def}(Exp_2)\}\ \langle x, y\rangle := \langle Exp_1, Exp_2\rangle\ \{Q\} \text{ se cumple.} \tag{3}$$

En la sustitución múltiple $Q(x \leftarrow Exp_1, y \leftarrow Exp_2)$, se sustituyen las expresiones simultáneamente (a la vez!, no una seguida de la otra).

6. Axioma de la sentencia continuar

La sentencia escrita como **continuar** es una sentencia que sólo sirve para 'proseguir' con la ejecución y se emplea para completar los casos de las sentencias alternativas. Esta sentencia no tiene ningún efecto sobre el estado de ejecución salvo el efecto de proseguir la ejecución en la siguiente sentencia. Por tanto,

Axioma de 'continuar':
Para cualquier predicado Q:

$$\{Q\}\ \textit{continuar}\ \{Q\} \text{ se cumple.} \tag{4}$$

Nota: En la siguiente lectura completaremos los Axiomas de la asignación y continuar dando sus reglas de inferencia.

7. Cuantificadores

7.1. Cuantificador universal y existencial

Para introducir los cuantificadores comenzamos con un ejemplo sencillo. Consideremos una variable a: **tipo** $[1..N]$ **de** entero, una variable x: entero y una variable n: entero. En la declaración de la tabla, N es una constante mayor que cero. Dado un estado de asignación, donde las variables toman ya unos valores concretos, nos preguntamos por ejemplo: ¿son distintos a x los elementos de la tabla entre 1 y n?. La pregunta tendrá una respuesta cierta o falsa y el predicado que responde a esa pregunta lo podemos formular como sigue:

$$a[1] \neq x \wedge a[2] \neq x \wedge a[3] \neq x \wedge a[4] \neq x \wedge ... \wedge a[n] \neq x \tag{5}$$

Observa que cada expresión booleana $a[.] \neq x$ es un patrón que se repite para los valores desde 1 hasta n. El predicado será cierto (y por tanto, la pregunta afirmativa) si todas las conjunciones se cumplen en el estado donde lo evaluamos y falso si alguna no se cumple. El predicado anterior lo escribimos de la forma:

$$\forall \gamma : 1 \leq \gamma \leq n : a[\gamma] \neq x \tag{6}$$

El símbolo $\forall$ se lee 'para todo', y la expresión anterior se denomina **cuantificador universal**.

Observa que hemos resumido de manera muy compacta una expresión inicial en la que no hemos podido indicar todos los términos (hemos escrito los puntos suspensivos porque no sabemos a priori el valor de la variable n cuando escribimos el predicado). El cuantificador universal generaliza la operación de conjunción. El predicado $1 \leq \gamma \leq n$ determina el **rango** (o **dominio**) del cuantificador, los valores que puede tomar γ, y esta variable, que se encuentra dentro del cuantificador, la denominamos **variable ligada** (al cuantificador).

Dado el cuantificador universal anterior, podemos 'separar' cualquier término del cuantificador, pero para simplificar, separamos el último para que observes la relación con la operación de conjunción:

$(\forall \gamma : 1 \leq \gamma \leq (n-1) : a[\gamma] \neq x) \wedge (a[n] \neq x)$

Ahora, en el rango del cuantificador ya no tenemos el valor n, puesto que lo hemos separado y nos hemos quedado con los valores de $1 \leq \gamma \leq (n-1)$. La separación anterior la podemos repetir para $n-1$,

$(\forall \gamma : 1 \leq \gamma \leq (n-2) : a[\gamma] \neq x) \wedge (a[n-1] \neq x) \wedge (a[n] \neq x)$

y podríamos continuar separando hata llegar a

$$(\forall \gamma : 1 \leq \gamma \leq 0 : a[\gamma] \neq x) \wedge (a[1] \neq x) \wedge (a[2] \neq x) \wedge ... \wedge (a[n-1] \neq x) \wedge (a[n] \neq x) \tag{7}$$

Llegados aquí, ¿cuánto vale $(\forall \gamma : 1 \leq \gamma \leq 0 : a[\gamma] \neq x)$?. El predicado $1 \leq \gamma \leq 0$ es 'siempre' falso, $[[1 \leq \gamma \leq 0 \equiv falso]]$, e indica que el conjunto de valores que puede tomar γ está vacío!!. Por otro lado, (7) es equivalente a (6) y esta equivalente a (5) luego necesariamente,

$[[(\forall\gamma : 1 \leq \gamma \leq 0 : a[\gamma] \neq x) \equiv cierto]]$,

ya que cierto es el elemento neutro de la conjunción.

En todo lo anterior hemos supuesto que la variable ligada γ es entera y que el rango $1 \leq \gamma \leq n$ tiene asociado el conjuno $[1..n]$ (ver tipo intervalo de enteros). Por ese motivo, también podemos escribir el cuantificador universal anterior como,

$$\forall\gamma : 1..n : a[\gamma] \neq x \tag{8}$$

Nota: Lo importante no es la notación. Lo importante es que el rango en el cuantificador representa un conjunto de valores para la variable ligada. No importa cómo denotemos este conjunto, pero cualquier notación que usemos debe entenderse bien.

Llegados a este punto, consideramos generalizar las ideas anteriores dando algunas propiedades del cuantificador universal:

$[[(\forall\gamma : falso : P) \equiv cierto]]$ regla del rango vacío

$[[(\forall\gamma : a \leq \gamma \leq b : P) \equiv (\forall\gamma : a \leq \gamma \leq (b-1) : P) \wedge P(\gamma \leftarrow b)]]$ separación por el último si $a \leq b$

$[[(\forall\gamma : a \leq \gamma \leq b : P) \equiv P(\gamma \leftarrow a) \wedge (\forall\gamma : (a+1) \leq \gamma \leq b : P)]]$ separación por el primero si $a \leq b$

$[[(\forall\gamma : \gamma = a : P) \equiv P(\gamma \leftarrow a)]]$ regla de un sólo valor de γ

$[[(\forall\gamma : R : P) \wedge (\forall\gamma : R : Q) \equiv (\forall\gamma : R : P \wedge Q)]]$

$[[(\forall\gamma : R : P) \wedge (\forall\gamma : S : P) \equiv (\forall\gamma : R \vee S : P)]]$

Consideremos de nuevo el ejemplo inicial y hagamos la pregunta ¿hay algún elemento en la tabla entre 1 y n igual a x?. De nuevo la pregunta será afirmativa o no, si se cumple el siguiente predicado,

$a[1] = x \vee a[2] = x \vee a[3] = x \vee a[4] = x \vee ... \vee a[n] = x$

Basta con que alguno de los predicados sea cierto para que la pregunta sea afirmativa. Observa que ahora hemos empleado la operación de disyunción entre predicados. La pregunta será falsa si todos los predicados anteriores son falsos. De nuevo podemos generalizar la disyunción utilizando el **cuantificador existencial** denotado por $\exists$ y que se lee 'existe'. Lo anterior lo escribimos como,

$$\exists\gamma : 1 \leq \gamma \leq n : a[\gamma] = x \tag{9}$$

Si hacemos todo el proceso anterior de separación, entonces en el caso de que sea cierto que

$n = 0$, $\exists\gamma : 1 \leq \gamma \leq 0 : a[\gamma] = x$ obtenemos que su valor es falso (el elemento neutro de la disyunción). Algunas de las propiedades del cuantificador existencial son:

$[[(\exists\gamma : falso : P) \equiv falso]]$ regla del rango vacío

$[[(\exists\gamma : a \leq \gamma \leq b : P) \equiv (\exists\gamma : a \leq \gamma \leq (b-1) : P) \vee P(\gamma \leftarrow b)]]$ separación por el último si $a \leq b$

$[[(\exists\gamma : a \leq \gamma \leq b : P) \equiv P(\gamma \leftarrow a) \vee (\exists\gamma : (a+1) \leq \gamma \leq b : P)]]$ separación por el primero si $a \leq b$

$[[(\exists\gamma : \gamma = a : P) \equiv P(\gamma \leftarrow a)]]$ regla de un sólo valor de γ

$[[(\exists\gamma : R : P) \vee (\exists\gamma : R : Q) \equiv (\exists\gamma : R : P \vee Q)]]$

$[[(\exists\gamma : R : P) \vee (\exists\gamma : S : P) \equiv (\exists\gamma : R \vee S : P)]]$

Habrás observado que las variables ligadas en los cuantificadores las hemos indicado con una letra griega, por ejemplo γ. Esto es así para diferenciarlas claramente de las variables declaradas en los programas. **Las variables ligadas sólo existen en el cuantificador y no se pueden emplear en las sentencias de los programas**. Procuraremos hacerlo así en las clases de teoría, pero el estudiante debe ser consciente de que, cuando se usan cuantificadores, las variables ligadas no pueden tomar el mismo nombre que las variables declaradas en los programas, ni siquiera en predicados distintos. Otra consideración es que, cambiar el nombre a una variable ligada (por uno distinto a los indicados en las expresiones) no modifica el significado del predicado. Otra cuestión, los rangos de los cuantificadores, el conjunto de valores de la variable ligada, pueden no estar acotados. Por ejemplo, la proposición 'x es un número entero positivo par' la podemos escribir con el predicado,

$$\exists\gamma : \gamma > 0 : x = 2\gamma$$

Una relación entre los cuantificadores universal y existencial que debemos tener en cuenta es que están relacionados por las leyes de De Morgan de la siguiente manera:

$$\begin{aligned} &[[\,(\forall\gamma : R : P) \equiv \neg(\exists\gamma : R : \neg P)\,]] \\ &[[\,(\exists\gamma : R : P) \equiv \neg(\forall\gamma : R : \neg P)\,]] \end{aligned} \tag{10}$$

7.2. Cuantificadores para sumas y productos

Muchos problemas prácticos de programación tratan sobre el cálculo de la suma de una secuencia de números, o funciones, o el cálculo de productos de números o funciones. Considera la suceción de Fibonacci, definida como, 0, 1, 1, 2, 3, 5, 8, 13, 21,..., más precisamente, $f_0 = 0$, $f_1 = 1$, y para cualquier término n con $n \geq 2$, $f_n = f_{n-1} + f_{n-2}$ (la suma de los dos anteriores). Si queremos calcular la suma de los $N+1$ primeros términos de dicha sucesión podemos escribir dicha suma como,

$$f_0 + f_1 + f_2 + ... + f_N$$

o bien, emplear el cuantificador para la suma, denotado $\sum$,

$$\sum \gamma : 0 \leq \gamma \leq N : f_\gamma$$

Otra forma, por otra parte muy habitual, de escribir la suma anterior sería,

$$\sum_{\gamma=0}^{N} f_\gamma$$

En general para una suma de expresiones aritméticas usamos,

$$\sum \gamma : R : Exp(\gamma)$$

donde R es el predicado que delimita el rango de la suma. Si $[[R \equiv false]]$, es decir el rango es vacío, la suma vale 0, el elemento neutro de la suma. La suma se puede 'separar', en el ejemplo anterior,

$$(\sum \gamma : 0 \leq \gamma \leq N : f_\gamma) = (\sum \gamma : 0 \leq \gamma \leq k : f_\gamma) + (\sum \gamma : (k+1) \leq \gamma \leq N : f_\gamma)$$

dada una variable k: $0..N$. Otros ejemplo donde encontramos la suma son:

- Producto de dos matrices cuadradas $N \times N$, $A = (a_{ij})$ y $B = (b_{ij})$, $A \cdot B$. El elemento p_{ij} de la matrix producto, se calcula como, $p_{ij} = \sum_{\gamma=1}^{N} a_{i\gamma} \cdot b_{\gamma j}$.
- La expansión en serie de Taylor de funciones:

  Función exponencial: $e^x = \sum_{\gamma=0}^{\infty} \frac{x^\gamma}{\gamma!}$ con $|x| < \infty$

  Función coseno: $cos(x) = \sum_{\gamma=0}^{\infty} (-1)^\gamma \frac{x^{2\gamma}}{(2\gamma)!}$ con $|x| < \infty$

Para el producto empleamos la expresión,

$$\prod \gamma : R : Exp(\gamma)$$

En el caso en el que el rango del producto sea vacío, $[[R \equiv false]]$, el producto vale el elemento neutro del producto, es decir, 1. Como ejemplo, el producto de Wallis permite calcular el número π:

$$\prod_{\gamma=1}^{\infty} \left(\frac{2\gamma}{2\gamma-1} \cdot \frac{2\gamma}{2\gamma+1}\right) = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \ldots = \frac{\pi}{2}$$

7.3. Otros cuantificadores

Otras expresiones que se pueden cuantificar son el cálculo del máximo o del mínimo de un conjunto de números (o de funciones)[5]. Otro cuantificador habitual es el operador de contero. Básicamente indica el número de veces que un predicado se hace cierto. Ejemplo: se dispone de una tabla t: **tabla** $[1..N]$ **de** entero, y se quiere indicar 'el número de elementos en los que hay un número par'. Para ese propósito, lo indicamos como,

$\#\gamma : 1 \leq \gamma \leq N : (t[\gamma] \bmod 2 = 0)$

[5]Se darán las explicaciones en los ejercicios donde se utilicen.

Si la tabla es [1, -3, 2, 5, 8, 6] (para $N = 6$), la expresión anterior dedulve el resultado 3, ya que el predicado $(t[\gamma] \bmod 2 = 0)$ sólo es cierto para $\gamma \in \{3, 5, 6\}$.

Cuando separas, por ejemplo, el último término, entonces, el resultado de la separación queda condicionado por el valor del predicado en ese término,

$$(\#\gamma : 1 \leq \gamma \leq N : P(\gamma)) = (\#\gamma : 1 \leq \gamma \leq N - 1 : P(\gamma)) + 1 \text{ si se cumple } P(\gamma \leftarrow N)$$

$$(\#\gamma : 1 \leq \gamma \leq N : P(\gamma)) = (\#\gamma : 1 \leq \gamma \leq N - 1 : P(\gamma)) \text{ si se cumple } \neg P(\gamma \leftarrow N)$$

Ejercicio 3 *Dada una tabla t:* **tabla** $[1..N]$ **de** *entero, con N una constante positiva, expresa, con la ayuda de los cuantificadores dados en esta lectura, las siguientes proposiciones:*

1. *r es la suma de los elementos de t.*
2. *m es el máximo de los elementos de t.*
3. *Los valores de los elementos de t están en orden creciente.*
4. *Si t contiene el valor* 1*, entonces t contiene el valor* 0*.*
5. *Hay un elemento en t que es igual a la suma de todos los elementos que le preceden.*
6. *El número de elementos pares de t es igual al número de elementos impares de t.*

Referencias

[1] Anne Kaldewaij. Programming: The derivation of algorithms. Prentice Hall International, series in computer science. 1990. [6]

[6] La notación que hemos seguido en esta Lectura[3], se encuentra en el libro de Kaldewaij. Las propiedades más habituales sobre los predicados y cuantificadores se resumen en los capítulos 1 y 3 del citado libro. El tratamiento más formal del cuantificador de 'conteo' se presenta al final del capítulo 3.

A. Resumen: Equivalencias entre predicados

Idempotencia:

- $[[P \wedge P \equiv P]]$
- $[[P \vee P \equiv P]]$

Commutatividad:

- $[[P \wedge Q \equiv Q \wedge P]]$
- $[[P \vee Q \equiv Q \vee P]]$
- $[[(P \equiv Q) \equiv (Q \equiv P)]]$

Asociatividad:

- $[[(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)]]$
- $[[(P \vee Q) \vee R \equiv P \vee (Q \vee R)]]$
- $[[((P \equiv Q) \equiv R) \equiv (P \equiv (Q \equiv R))]]$

Distributividad:

- $[[P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)]]$
- $[[P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)]]$
- $[[P \vee (Q \equiv R) \equiv P \vee Q \equiv P \vee R]]$

Absorción:

- $[[P \wedge (P \vee R) \equiv P]]$
- $[[P \vee (P \wedge R) \equiv P]]$

Reglas cierto-falso:

- $[[P \wedge cierto \equiv P]]$ $[[P \wedge falso \equiv falso]]$
- $[[P \vee falso \equiv P]]$ $[[P \vee cierto \equiv cierto]]$

De Morgan:

- $[[\neg(P \wedge Q) \equiv \neg P \vee \neg Q]]$ $[[\neg(P \vee Q) \equiv \neg P \wedge \neg Q]]$

Negación:

- $[[\neg\neg P \equiv P]]$
- $[[P \vee \neg P \equiv cierto]]$
- $[[P \wedge \neg P \equiv falso]]$

Implicación:

- $[[P \Rightarrow Q \equiv \neg P \vee Q]]$
- $[[(P \Rightarrow Q) \wedge (Q \Rightarrow P) \equiv P \equiv Q]]$
- $[[P \Rightarrow P \vee Q]]$ $[[P \wedge Q \Rightarrow P]]$
- $[[falso \Rightarrow P]]$ $[[P \Rightarrow cierto]]$
- $[[cierto \Rightarrow P \equiv P]]$ $[[P \Rightarrow falso \equiv \neg P]]$

Equivalencia:

- $[[P \equiv P]]$
- $[[P \equiv P \equiv cierto]]$
- $[[\neg P \equiv P \equiv falso]]$

B. Resumen: Cuantificador universal y existencial

$[[(\forall\gamma : falso : P) \equiv cierto]]$ (regla del rango vacío)

$[[(\forall\gamma : a \leq \gamma \leq b : P) \equiv (\forall\gamma : a \leq \gamma \leq (b-1) : P) \wedge P(\gamma \leftarrow b)]]$ (separación por el último si $a \leq b$)

$[[(\forall\gamma : a \leq \gamma \leq b : P) \equiv P(\gamma \leftarrow a) \wedge (\forall\gamma : (a+1) \leq \gamma \leq b : P)]]$ (separación por el primero si $a \leq b$)

$[[(\forall\gamma : \gamma = a : P) \equiv P(\gamma \leftarrow a)]]$ (regla de un sólo valor de γ)

$[[(\forall\gamma : R \wedge S : P) \equiv (\forall\gamma : R : S \Rightarrow P)]]$ (regla del rango conjuntivo)

$[[Q \vee (\forall\gamma : R.\gamma : P.\gamma) \equiv (\forall\gamma : R.\gamma : Q \vee P.\gamma)]]$

$[[Q \wedge (\forall\gamma : R.\gamma : P.\gamma) \equiv (\forall\gamma : R.\gamma : Q \wedge P.\gamma)]]$ (para R no vacío)

$[[(\forall\gamma : R : P) \wedge (\forall\gamma : R : Q) \equiv (\forall\gamma : R : P \wedge Q)]]$

$[[(\forall\gamma : R : P) \wedge (\forall\gamma : S : P) \equiv (\forall\gamma : R \vee S : P)]]$

$[[(\forall\gamma : R.\gamma : P.\gamma) \vee (\forall\gamma : S.\gamma : Q.\gamma) \equiv (\forall\gamma, \beta : R.\gamma \wedge S.\beta : P.\gamma \vee Q.\beta)]]$

$[[(\exists\gamma : falso : P) \equiv falso]]$ (regla del rango vacío)

$[[(\exists\gamma : a \leq \gamma \leq b : P) \equiv (\exists\gamma : a \leq \gamma \leq (b-1) : P) \vee P(\gamma \leftarrow b)]]$ (separación por el último si $a \leq b$)

$[[(\exists\gamma : a \leq \gamma \leq b : P) \equiv P(\gamma \leftarrow a) \vee (\forall\gamma : (a+1) \leq \gamma \leq b : P)]]$ (separación por el primero si $a \leq b$)

$[[(\exists\gamma : \gamma = a : P) \equiv P(\gamma \leftarrow a)]]$ (regla de un sólo valor de γ)

$[[(\exists\gamma : R \wedge S : P) \equiv (\exists\gamma : R : S \wedge P)]]$ (regla del rango conjuntivo)

$[[Q \wedge (\exists\gamma : R.\gamma : P.\gamma) \equiv (\exists\gamma : R.\gamma : Q \wedge P.\gamma)]]$

$[[Q \vee (\exists\gamma : R.\gamma : P.\gamma) \equiv (\exists\gamma : R.\gamma : Q \vee P.\gamma)]]$ (para R no vacío)

$[[(\exists\gamma : R : P) \vee (\exists\gamma : R : Q) \equiv (\exists\gamma : R : P \vee Q)]]$

$[[(\exists\gamma : R : P) \vee (\exists\gamma : S : P) \equiv (\forall\gamma : R \vee S : P)]]$

$[[(\exists\gamma : R.\gamma : P.\gamma) \wedge (\exists\gamma : S.\gamma : Q.\gamma) \equiv (\exists\gamma, \beta : R.\gamma \wedge S.\beta : P.\gamma \wedge Q.\beta)]]$

Por las Leyes de De Morgan,

$[[\,(\forall\gamma : R : P) \equiv \neg(\exists\gamma : R : \neg P)\,]]$

$[[\,(\exists\gamma : R : P) \equiv \neg(\forall\gamma : R : \neg P)\,]]$

C. Resumen: Otras cuantificaciones