Ejemplos de Especificación de Problemas Algorítmicos

Jesús N. Ravelo
Universidad Simón Bolívar
Dpto. de Computación y Tecnología de la Información

Octubre 2009

Resumen

Estas notas presentan varios ejemplos de especificación formal de problemas algorítmicos utilizando el esquema pre/post-condición para programas imperativos. Bajo este esquema de especificación, una precondición describe las propiedades requeridas de los datos de entrada y una postcondición describe las propiedades deseadas de los resultados de salida. No se pretende acá describir las teorías y los métodos de construcción de programas imperativos basadas en este tipo de especificaciones, sino ofrecer una serie de ejemplos representativos de especificaciones que muestren los inconvenientes típicos que pueden encontrarse en el modelado formal de problemas algorítmicos.

0. Introducción

Bajo el esquema pre/post-condición para especificación de problemas algorítmicos, el comportamiento deseado de un programa imperativo[0] a ser construido es descrito en términos de una precondición, la cual describe las propiedades requeridas a los datos de entrada del programa, y una postcondición, la cual describe las propiedades que se desea sean satisfechas por los resultados de salida del programa.

Con base en este estilo de especificación, se han desarrollado muchas teorías y métodos para construcción de programas imperativos que satisfagan una especificación dada, esto es, programas que funcionen correctamente con respecto a alguna especificación pre/post-condición. Los pilares fundamentales de tales metodologías de programación fueron propuestos por C.A.R. Hoare [ACMb, GS94c] a finales de los años 60 del siglo pasado, en la forma de lo que hoy se conocen como *Tripletas de Hoare* [Hoa69], y poco después con mayor elaboración por E.W. Dijkstra [ACMa, GS94b, SL95a] a principios de los años 70 del siglo pasado, en la forma de sus *Transformadores de Predicados wp* [Dij75, Dij76]. Muchos otros investigadores han continuado la labor de estos pioneros [Bac80, Bac88, BW98, DS90, HJ98, Mor88, Mor94, MV94, Mor87] y aún hoy en día se sigue haciendo mucha investigación sobre este tema [APM08, Uus06].

[0]Hay varios estilos o paradigmas de programación, siendo los tres fundamentales el imperativo, el funcional y el lógico. En estas notas sólo nos interesan los programas imperativos, o más concretamente la especificación del comportamiento de programas imperativos, que consisten en dar "órdenes imperativas" a un computador. En cualquier caso, no es de nuestro interés en estas notas otros tipos de programas como los programas funcionales o programas lógicos, por lo que "programa imperativo" puede simplemente ser leído como sinónimo de "programa".

Este estilo de especificación pre/post-condición se ha convertido desde entonces en el más utilizado para describir el comportamiento externo de programas imperativos, entendiendo que tal comportamiento externo nos describe *qué* calcula o debe calcular un programa, sin necesidad de que conozcamos los detalles internos de *cómo* el programa o los programas en cuestión realizan tal cálculo.

No pretendemos en estas notas explicar "desde cero" el estilo de especificación con pre/post-condiciones, por considerar que tal explicación se encuentra lo suficientemente elaborada en la literatura disponible [Gri81, DF88, Kal90, Mez00, Bac03]. En relación con posibles diferencias de formato entre las distintas referencias bibliográficas señaladas, utilizaremos el estilo de A. Kaldewaij [Kal90] y O. Meza [Mez00].

A pesar de que nuestras especificaciones de problemas algorítmicos serán formales, en el sentido de que las pre/post-condiciones serán fórmulas de lógica matemática (específicamente la lógica de predicados de 1er. orden enriquecida con algunos operadores típicos aritméticos), será importante en todo momento que prestemos atención a las descripciones en lenguaje natural[1] de tales problemas. Quienes abogamos por el uso del lenguaje formal de la lógica (o lenguajes formales similares) no pretendemos en absoluto y bajo ninguna circunstancia sugerir que el uso del lenguaje natural debe ser eliminado. Los seres humanos nos comunicamos en lenguaje natural y nuestra primera aproximación a la descripción de cualquier problema algorítmico será siempre mejor entendida inicialmente en lenguaje natural. Sin embargo, es bien conocido que utilizando lenguaje natural nuestras descripciones tienden a contener ambigüedades, ser inconsistentes o ser incompletas. Es por ello que, una vez comprendida mediante lenguaje natural una descripción de problema, el modelar tal descripción en un lenguaje formal matemático nos permitirá redondear la especificación, puliendo detalles que podrían haber quedado opacos en la descripción original. Más aún, es posible que el proceso de formalización es el que nos permita darnos cuenta de que había detalles opacos en la descripción original en lenguaje natural, cuando quizá inicialmente pensábamos que ya todo estaba claro. En algunos de los ejemplos presentados en estas notas tendremos ocasión de enfrentar este tipo de inconvenientes, resaltando en cada caso cuál fue la ayuda prestada por la formalización.

1. Ejemplos Iniciales

En esta sección nos dedicaremos a unos primeros ejemplos que permitan ilustrar los componentes básicos de una especificación, el uso o no de variables de especificación además de las variables de programa, la declaración o no de variables de programa como constantes, etcétera.

1.0. Porcentaje de Aceptación

Se ha realizado una encuesta en la que se le ha preguntado a una cierta cantidad de personas si aprueban o no la actuación del gobierno ante un cierto evento. A disposición se tiene la cantidad de personas encuestadas y la cantidad de personas que respondieron favorablemente, y se desea calcular a partir de estos datos el porcentaje de aceptación que tuvo la actuación del gobierno ante la eventualidad referida.

La descripción dada de nuestro problema indica que tenemos dos datos de entrada: la cantidad de personas encuestadas y la cantidad de personas que respondieron favorablemente. Llamemos

[1] El castellano, en nuestro caso.

a estos datos t, por *t*otal, y f, por *f*avorable, los cuales deben ser de tipo entero por referirse a cantidad de personas. Esto corresponde a la declaración de variables $t, f : int$.

Ahora bien, para que estos datos de entrada tengan sentido, debemos requerir que t sea un número positivo[2] ya que la encuesta ya ha sido llevada a cabo (haber encuestado a cero personas correspondería a no haber hecho la encuesta aún). Esto nos indica que necesitamos el siguiente requerimiento en nuestra precondición: $t > 0$. En cuanto a f, su valor no puede sobrepasar la cantidad de personas encuestadas ni ser negativo[3], lo cual nos da el segundo requerimiento para nuestra precondición: $0 \leqslant f \leqslant t$.

En relación con resultados de salida, necesitamos sólo uno: el porcentaje en cuestión. Llamémosle p, el cual será declarado de tipo real[4]. Necesitamos entonces tener también entre nuestra declaración de variables a la declaración $p : real$.

Finalmente, se requiere que nuestro resultado de salida p en efecto corresponda al porcentaje deseado en relación con los valores de t y de f. Esto corresponde a requerir lo siguiente en nuestra postcondición: $p = (f/t) * 100$.

Uniendo los fragmentos que hemos ido proponiendo, tendríamos la siguiente especificación:

$$
\begin{array}{l}
[\![\ \textbf{const} \ \ t, f : int \,; \\
\quad \textbf{var} \ \ p : real \,; \\
\quad \{ \ t > 0 \ \wedge \ 0 \leqslant f \leqslant t \ \} \\
\quad \textsc{CalcularPorcentajeAceptación} \\
\quad \{ \ p = (f/t) * 100 \ \} \\
]\!] \ \ .
\end{array}
$$

Recuerde que la declaración de constantes ("**const**") corresponde a una declaración de variables de programa ("**var**") a las cuales no se les permite cambiar de valor. El mismo significado puede ser logrado utilizando variables de especificación que representen los valores iniciales de tales variables. La especificación anterior es equivalente a la siguiente, en la cual se utilizan las variables de especificación X y Y:

$$
\begin{array}{l}
[\![\ \textbf{var} \ \ t, f : int \,; \\
\qquad p : real \,; \\
\quad \{ \ t = X \ \wedge \ f = Y \ \wedge \ t > 0 \ \wedge \ 0 \leqslant f \leqslant t \ \} \\
\quad \textsc{CalcularPorcentajeAceptación} \\
\quad \{ \ t = X \ \wedge \ f = Y \ \wedge \ p = (f/t) * 100 \ \} \\
]\!] \ \ .
\end{array}
$$

Note que las variables de especificación X y Y no fueron declaradas ya que éstas no son variables de programa ("**var**").

Note también que el resultado deseado puede ser adecuadamente especificado sin necesidad de exigir que las variables que contienen los datos de entrada no puedan ser modificadas. Esto se

[2]Entenderemos siempre en estas notas que positivo significa lo que algunas personas llaman "estrictamente positivo", esto es, estrictamente mayor que cero.

[3]De nuevo, entenderemos siempre en estas notas que negativo significa lo que otras personas llaman "estrictamente negativo".

[4]Dependiendo de las necesidades de quiénes nos solicitan la resolución de este problema, podría bastar que el resultado fuese de tipo entero, considerando algún tipo de redondeo adecuado. En nuestro ejemplo, supondremos que se solicitó precisión mayor que entera.

muestra en la siguiente especificación:

$[\![$ **var** $t, f : int$;
$\quad p : real$;
$\{\ t = X\ \wedge\ f = Y\ \wedge\ t > 0\ \wedge\ 0 \leqslant f \leqslant t\ \}$
CalcularPorcentajeAceptaciónDiferenteAlAnterior
$\{\ p = (Y/X) * 100\ \}$
$]\!]$.

Hemos cambiado el nombre del problema especificado a CalcularPorcentajeAceptación-DiferenteAlAnterior, ya que este problema es formalmente distinto al especificado inicialmente, al cual habíamos llamado CalcularPorcentajeAceptación. En el nuevo problema, con estado inicial $[\,t = 1500\,,\ f = 500\,,\ p = -7{,}21\,]$ puede ser aceptado el estado final $[\,t = -64\,,\ f = 902\,,\ p = 33{,}33\,]$, entre otros similares, mientras que para el problema anterior con tal estado inicial sólo podría aceptarse el estado final $[\,t = 1500\,,\ f = 500\,,\ p = 33{,}33\,]$. Las dos primeras especificaciones sí denotan exactamente al mismo problema algorítmico, razón por la cual en ambas se utilizó el mismo nombre de problema.

Sutileza sobre abusos conjuntivos de notación:

En nuestra expresión $0 \leqslant f \leqslant t$ de la precondición hay un sutil abuso de notación: esta expresión es en realidad sólo una abreviatura de la correcta expresión $0 \leqslant f \wedge f \leqslant t$.

¿Por qué llamamos a esto un abuso de notación? Debido que la expresión $0 \leqslant f \leqslant t$ es formalmente incorrecta si la aplicación de los operadores binarios que aparecen en ella ("$\leqslant$") es interpretada de manera convencional. Expliquemos esto con más detalle valiéndonos de otros ejemplos de expresiones similares.

Siendo a, b y c variables enteras, la expresión $a - b - c$ es usualmente interpretada como $(a - b) - c$, bajo el supuesto de que la asociación implícita se hace hacia la izquierda. Esta interpretación es perfectamente correcta en cuanto al uso del tipo entero en la expresión, ya que: (i) por ser a y b enteros, tenemos que $a - b$ también es un entero; y (ii) por ser $a - b$ un entero y c también, finalmente la expresión $(a - b) - c$ es correctamente de tipo entero. Todos los tipos de las subexpresiones fueron usados adecuadamente de acuerdo con los tipos de los operadores involucrados (en este caso sólo el operador "$-$").

Note que bajo el supuesto de que la asociación implícita del operador de sustracción se haga hacia la derecha, esto es, $a - (b - c)$, la expresión será igualmente correcta en cuanto a los tipos involucrados (aunque el resultado de su evaluación sería diferente al anterior).

Ahora analicemos el caso de nuestra expresión $0 \leqslant f \leqslant t$. Bajo los posibles supuestos de asociación implícita discutidos para el caso de nuestro ejemplo con sustracción, las dos posibles interpretaciones de esta expresión serían $(0 \leqslant f) \leqslant t$ y $0 \leqslant (f \leqslant t)$. Continuemos nuestro análisis con la primera de estas dos interpretaciones, asociando hacia la izquierda. Siendo 0 y f enteros, el tipo de la subexpresión $0 \leqslant f$ es booleano (o lógico) pues el operador de menor-o-igual (también llamado "no-mayor") lleva operandos enteros a un resultado booleano. Siendo $0 \leqslant f$ un booleano y t un entero, sería entonces incorrecta la aplicación del operador de menor-o-igual en la expresión $(0 \leqslant f) \leqslant t$ pues su operando izquierdo sería un booleano y su operando derecho un entero. Y una expresión con errores de tipo como éste es una expresión <u>sin sentido</u> (ojo, no "falsa", ni "mala", sino <u>sin sentido</u>).

Note que la interpretación de asociación hacia la derecha $0 \leqslant (f \leqslant t)$ tiene un error de tipo análogo.

La interpretación adecuada para nuestra expresión, que es el "abuso conjuntivo de notación" que nos llevó a este desvío en el texto, corresponde a interpretar las aplicaciones continuas del operador de menor-o-igual de manera *conjuntiva* en lugar de asociándolas hacia la izquierda o asociándolas hacia la derecha. Esto significa que, en general, con x, y y z números enteros o reales, la expresión $x \leqslant y \leqslant z$ debe ser tomada como abreviación de la expresión $x \leqslant y \wedge y \leqslant z$. La misma interpretación conjuntiva debe ser aplicada a los otros operadores binarios relacionales "$<$", "$\geqslant$" y "$>$", y a combinaciones de éstos como en, por ejemplo, $x > y \geqslant z$, que abrevia a la expresión $x > y \wedge y \geqslant z$.

Fin de sutileza sobre abusos conjuntivos de notación.

Pasemos ahora a otro problema.

1.1. Dame mi vuelto... de cualquier forma, pero dámelo...

Suponga que estamos en el país CUALQUIERALANDIA de cuya unidad monetaria, el *cualco*, existen monedas de 1, 2 y 5 unidades. En una máquina dispensadora de chucherías de CUALQUIERALANDIA, luego de que una persona ha solicitado un producto con un cierto precio y ha introducido suficiente dinero en la máquina, se desea darle vuelto en monedas de 1, 2 y 5 *cualcos*. (La máquina también debe dispensar la chuchería requerida, pero ese componente de la automatización ha sido encargado a otras personas y no a quienes estamos leyendo este texto.) Nuestro requerimiento específico consiste en, dados el precio de la chuchería requerida y la cantidad de dinero que la persona introdujo en la máquina, determinar alguna combinación de cantidades de monedas de las tres unidades correspondiente al vuelto. No tiene por qué ser la mejor combinación de monedas (bajo alguna interpretación de "mejor") sino simplemente alguna combinación. En CUALQUIERALANDIA no existen fragmentos de *cualcos*, por lo que todas las cantidades de dinero pueden ser modeladas sin necesidad de números reales.

Los datos de entrada son el precio de la chuchería en cuestión y la cantidad total de dinero entregada a la máquina; llamemos a éstos p y t respectivamente. Como no existen fragmentos de *cualcos*, éstos pueden ser de tipo entero y, por lo tanto, podemos usar la declaración $p, t : int$. Los resultados de salida serán las tres cantidades de monedas de 1, 2 y 5 *cualcos* correspondientes al vuelto. Éstos también serán de tipo entero, y llamándolos $v1$, $v2$ y $v5$ podemos entonces completar la declaración de objetos involucrados en nuestro problema con $v1, v2, v5 : int$.

En la precondición debemos exigir que los datos de entrada tengan sentido. Es razonable suponer que tanto el precio como el total de dinero deben ser positivos y que el total debe ser suficiente para pagar la chuchería. Todo esto se puede expresar de manera sucinta como sigue: $0 < p \leqslant t$. Note de nuevo el uso conjuntivo de los operadores relacionales, y note además que por transitividad esta expresión también señala que el total es positivo.

En la postcondición debemos expresar qué necesitamos que sea satisfecho por nuestros resultados de salida. Primero, necesitamos que ninguna de las tres cantidades sea negativa, esto es, que cada una sea positiva o cero: $v1 \geqslant 0 \wedge v2 \geqslant 0 \wedge v5 \geqslant 0$. Segundo, necesitamos que la combinación de ellas corresponda al vuelto, esto es: $v1 + 2 * v2 + 5 * v5 = t - p$.

Componiendo la especificación completa tendríamos entonces:

$$
\begin{array}{l}
[\![\ \ \textbf{const}\ \ p, t : int\,; \\
\quad \textbf{var}\ \ v1, v2, v5 : int\,; \\
\quad \{\ \ 0 < p \leqslant t\ \ \} \\
\quad \textsc{CalcularUnVuelto} \\
\quad \{\ \ v1 \geqslant 0\ \ \wedge\ \ v2 \geqslant 0\ \ \wedge\ \ v5 \geqslant 0\ \ \wedge\ \ v1 + 2 * v2 + 5 * v5\ =\ t - p\ \ \} \\
]\!]\ \ .
\end{array}
$$

Note que este problema tiene una característica interesante, la cual se ve reflejada en la especificación: los resultados no son únicos. Esto es, en general se tendrán varias posibles combinaciones válidas de 1, 2 y 5 *cualcos* correspondientes al vuelto. Por ejemplo, con el estado inicial

$$[\, p = 3\,,\, t = 10\,,\, v1 = -28\,,\, v2 = 407\,,\, v5 = 0\,]$$

puede ser aceptados los estados finales

$$
\begin{array}{l}
[\, p = 3\,,\, t = 10\,,\, v1 = 7\,,\, v2 = 0\,,\, v5 = 0\,]\,, \\
[\, p = 3\,,\, t = 10\,,\, v1 = 3\,,\, v2 = 2\,,\, v5 = 0\,] \text{ y} \\
[\, p = 3\,,\, t = 10\,,\, v1 = 0\,,\, v2 = 2\,,\, v5 = 1\,]\,,
\end{array}
$$

entre otros.

A estas especificaciones de problemas en las que los resultados finales aceptables no son únicos se les llama *relacionales* o *no-funcionales*, en contraposición con las especificaciones *funcionales* en las que tales resultados sí son únicos[5]. También se les puede llamar especificaciones *no-determinísticas*, en contraposición con las *determinísticas* en las que se dice que los resultados, por ser únicos, están determinados por los datos de entrada.

2. Análisis por Casos

En esta nueva sección trabajaremos con problemas en los que deben ser analizadas varias posibles situaciones o casos. Veremos que hay más de una manera de formalizar en el lenguaje de la lógica tal tipo de multiplicidad de situaciones.

2.0. Valor Absoluto

Quizá el problema más sencillo en el que se presenta la necesidad de hacer análisis por casos es en el problema de cálculo de valor absoluto: teniendo como entrada un número real se desea calcular el valor absoluto de éste.

Utilizando el operador aritmético de valor absoluto, denotado convencionalmente como "$|\_|$",

[5] Estos nombres se deben a la posible modelación de especificaciones como relaciones entre entradas y salidas, y a la caracterización de relaciones y funciones en teoría de conjuntos.

la especificación es bastante concisa:

$[\![$ **const** a : *real* ;
var b : *real* ;
{ *true* }
CalcularValorAbsoluto
{ $b = |a|$ }
$]\!]$.

Note que la precondición *true* señala que no hay requerimientos especiales sobre el dato de entrada[6].

Ahora bien, ¿qué habría pasado si quisiésemos especificar este problema y no hubiésemos contado con la suerte de que existe una notación convencional y ampliamente conocida para la operación de valor absoluto? Habríamos tenido que recurrir a fórmulas lógicas más complejas para expresar el resultado que se desea calcular. Veremos a continuación dos maneras de modelar lógicamente la postcondición anterior sin hacer uso del operador de valor absoluto "$|\_|$".

Sabemos que el valor absoluto de un número cualquiera es, cuando éste es positivo, él mismo y, cuando éste es negativo, el opuesto de sí mismo[7]. En el caso de cero, como el opuesto de cero es él mismo, su valor absoluto corresponde tanto a él mismo como al opuesto de sí mismo.

Esta diferenciación de un caso u otro caso es modelada en lógica mediante el operador de disyunción "$\vee$". En un caso, tendríamos que el dato de entrada es positivo y que el resultado es la misma entrada, correspondiente al operador de conjunción "$\wedge$". En el otro caso, tendríamos que el dato de entrada es negativo y que el resultado es el opuesto de la entrada, de nuevo modelable con el operador de conjunción "$\wedge$". El caso del cero puede ser indistintamente anexado al primer caso o al segundo caso, o inclusive a ambos, debido a la consistencia del resultado para cero en cualquiera de las dos situaciones. Todo esto nos lleva a la siguiente especificación:

$[\![$ **const** a : *real* ;
var b : *real* ;
{ *true* }
CalcularValorAbsoluto
{ $(a \geqslant 0 \ \wedge \ b = a) \ \vee \ (a \leqslant 0 \ \wedge \ b = -a)$ }
$]\!]$.

Hay otra manera de modelar esta situación con fórmulas lógicas, expresando lo que ocurre en cada caso mediante una implicación. En un caso tenemos que *si* el dato de entrada es positivo *entonces* el resultado debe ser igual a la entrada, lo cual es una implicación "$\Rightarrow$". Igualmente, en el otro caso, tenemos que *si* el dato de entrada es negativo *entonces* se debe dar como resultado

[6]Con cualquier fórmula lógica como precondición, lo que estamos expresando es que sólo son aceptables aquellos estados iniciales en los que la evaluación de tal fórmula lógica resulte verdadera. Por ejemplo, con $n > 30$ como precondición, siendo n una variable entera, estaríamos diciendo que son aceptables los estados iniciales en los que n vale 31, ó 32, ó 33, ó 34, etcétera; esto es, estados en los que la evaluación de la fórmula lógica $n > 30$ da verdadero como resultado. Contrariamente, los estados en los que n vale 30, ó 29, ó 28, etcétera, en los cuales la evaluación de la fórmula lógica $n > 30$ da falso, son considerados inaceptables. Como la evaluación de la fórmula *true* da verdadero en cualquier estado, en este caso para cualquier valor entero de n, esta precondición expresa que todo posible estado inicial es aceptable.

[7]El "opuesto" corresponde a la operación llamada con frecuencia simplemente "$-$ unario".

al opuesto de la entrada, lo cual corresponde a otra implicación "⇒". Ahora bien, bajo esta modelación la unión de los casos no debe ser hecha mediante disyunción sino mediante conjunción "∧". Se necesita que, *si* la entrada es positiva, ocurra bla-bla-bla *y también* que, *si* la entrada es negativa, ocurra otro-bla-bla-bla. Con esta nueva modelación la especificación queda como sigue:

$$
\begin{array}{l}
[\![\ \textbf{const} \ \ a : \mathit{real} \,; \\
\quad \textbf{var} \ \ b : \mathit{real} \,; \\
\quad \{ \ \mathit{true} \ \} \\
\quad \textsc{CalcularValorAbsoluto} \\
\quad \{ \ (a \geqslant 0 \ \Rightarrow \ b = a) \ \wedge \ (a \leqslant 0 \ \Rightarrow \ b = -a) \ \} \\
]\!] \quad .
\end{array}
$$

Para finalizar este ejemplo introductorio al análisis por casos, note que la primera versión de nuestra especificación construida con el operador "|_|" también contiene implícitamente un análisis de casos, puesto que la definición del operador de valor absoluto "|_| se construye por casos:

$$
|x| \ = \ \begin{cases} \ \ x & \text{si } \ x \geqslant 0 \,, \text{ y} \\ -x & \text{si } \ x \leqslant 0 \,. \end{cases}
$$

De hecho, la formalización de esta definición del operador de valor absoluto "|_| en el lenguaje de la lógica corresponde a las modelaciones presentadas en la postcondición de nuestras segunda y tercera versiones de la especificación.

Posibles dudas:

Si Ud. está dudando del porqué la versión lógica del análisis por casos con implicaciones tiene como conector principal a una conjunción en lugar de una disyunción, esto es, si Ud. cree que la postcondición adecuada en la tercera versión de nuestra especificación ha debido ser

$$
(a \geqslant 0 \ \Rightarrow \ b = a) \ \vee \ (a \leqslant 0 \ \Rightarrow \ b = -a) \quad ,
$$

le sugerimos busque resolver su propia duda de la siguiente manera: evalúe el valor de nuestra postcondición propuesta (con conjunción) y de su fórmula (con disyunción) en los estados $[\, a = 17 \,,\ b = 17 \,]$, $[\, a = -51 \,,\ b = 51 \,]$, $[\, a = 17 \,,\ b = 8 \,]$, y $[\, a = -51 \,,\ b = 2 \,]$. Ya que los dos primeros corresponden a estados finales deseables mientras que los dos últimos son indeseables, la postcondición adecuada debería ser verdadera en los dos primeros casos y falsa en los otros dos. Nuestra fórmula (con conjunción) cumple con esto, mientras la otra (con disyunción) no lo cumple pues da verdadero en todos los casos.

Igualmente, si Ud. tiene dudas sobre el hecho de que en nuestras postcondiciones el caso del cero es considerado en ambas ramas, evalúe todas nuestras postcondiciones en los estados $[\, a = 0 \,,\ b = 0 \,]$, $[\, a = 0 \,,\ b = 18 \,]$, y $[\, a = 0 \,,\ b = -7 \,]$. De nuevo, verá que la evaluación siempre da verdadero para el primer estado y falso para los otros dos, lo cual es perfectamente consistente con el hecho de que sólo el primero es deseable como estado final.

Y si aún se está preguntando por qué no consideramos al -0 como posible resultado para b, dése cuenta de que sí lo hicimos, ya que -0 es 0 (de la misma manera que $0 + 0$ es 0).

Fin de posibles dudas.

2.1. Precio según Peso

Volvamos ahora al país CUALQUIERALANDIA (sección 1.1), en el que funciona la empresa CUALCOTRANSPORTES, la cual transporta paquetes entre dos puntos cualesquiera de la capital del país. CUALCOTRANSPORTES calcula sus tarifas básicas de acuerdo con el peso de los paquetes a transportar: los primeros 5 kilogramos tienen un costo único de 300 *cualcos*; sobre los 5 kilogramos, cada kilo adicional o fracción cuesta 40 *cualcos* más. Esto es, transportar un paquete de 2,3 kilos o de 4,8 kilos cuesta 300 *cualcos*, mientras que transportar un paquete de 7 kilos y uno de 10,2 kilos cuesta respectivamente 380 y 540 *cualcos*.

Aparte de la tarifa básica, CUALCOTRANSPORTES ofrece a su clientela frecuente la posibilidad de pagar una cuota única anual, cuyo monto es irrelevante para nuestro problema algorítmico, gracias a la cual de allí en adelante se paga una tarifa especial por cada envío. Esta tarifa especial es de 100 *cualcos* por cada 5 kilos o fracción. Esto es, transportar un paquete de 2,3 kilos o de 4,8 kilos cuesta a tales clientes especiales 100 *cualcos*, y transportar un paquete de 7 kilos y uno de 10,2 kilos les cuesta respectivamente 200 y 300 *cualcos*.

Nuestro problema algorítmico consiste en calcular el costo de un envío sabiendo el peso del paquete y sabiendo también si la persona que solicita el servicio es cliente especial o no.

Como entrada primero tenemos el peso, el cual es un número real. Además, tenemos como otro dato de entrada la información sobre el carácter especial o no de la tarifa a utilizar, la cual podemos modelar con un booleano, con verdadero como la señal para usar la tarifa especial y falso como la señal contraria[8]. Esto corresponde a las declaraciones $p : real$ y $e : boolean$. En la precondición basta con requerir que el peso sea positivo.

El único resultado de salida será el costo calculado para el envío, el cual será de tipo entero y por tanto puede ser declarado con $c : int$. La postcondición que determina cuál debe ser el costo a calcular corresponderá a una fórmula de análisis por casos en la cual se considerarán tres posibilidades: (i) que se deba usar la tarifa convencional para un paquete que no pesa más de 5 kilos, (ii) que se deba usar la tarifa convencional para un paquete que pesa más de 5 kilos, y (iii) que se deba usar la tarifa especial. Al igual que hicimos con el problema de cálculo de valor absoluto (sección 2.0), podemos modelar el análisis de casos como disyunción de conjunciones o como conjunción de implicaciones.

Todo esto correspondería a las siguientes dos especificaciones:

$$
\begin{array}{l}
[\![\ \textbf{const}\ \ p : real\,; \\
\qquad\qquad e : boolean\,; \\
\quad \textbf{var}\ \ c : int\,; \\
\quad \{\ p > 0\ \} \\
\quad \textsc{CalcularCostoEnvío} \\
\quad \left\{ \begin{array}{l}
(\neg e\ \wedge\ p \leqslant 5\ \wedge\ c = 300) \\
\quad \vee \\
(\neg e\ \wedge\ p > 5\ \wedge\ c = 300 + 40 * \lceil p - 5 \rceil) \\
\quad \vee \\
(e\ \wedge\ c = 100 * \lceil p/5 \rceil)
\end{array} \right\} \\
]\!]\ \ .
\end{array}
$$

[8] La asignación de verdadero a especial y falso a no-especial es arbitraria. Sería igualmente válido utilizar la interpretación contraria.

y

$$
\begin{array}{l}
[\![\ \ \textbf{const}\ \ p : \mathit{real}\,; \\
\qquad\qquad\ e : \mathit{boolean}\,; \\
\quad \textbf{var}\ \ c : \mathit{int}\,; \\
\quad \{\ \ p > 0\ \ \} \\
\quad \textsc{CalcularCostoEnvío} \\
\quad \left\{ \begin{array}{l}
(\neg e\ \wedge\ p \leqslant 5\ \ \Rightarrow\ \ c = 300) \\
\quad \wedge \\
(\neg e\ \wedge\ p > 5\ \ \Rightarrow\ \ c = 300 + 40 * \lceil p - 5 \rceil) \\
\quad \wedge \\
(e\ \ \Rightarrow\ \ c = 100 * \lceil p/5 \rceil)
\end{array} \right\} \\
]\!]\ \ .
\end{array}
$$

El operador "$\lceil \_ \rceil$" utilizado en la postcondición es el conocido operador de "techo" que redondea números reales hacia arriba, esto es, hacia el entero no-inferior (superior o igual) más cercano.

Casos y subcasos, es equivalente:

Los tres casos analizados también podrían haber sido presentados mediante casos y subcasos. Esto es, hemos podido haber hecho primero la división entre los casos tarifa convencional y especial, y luego haber subdividido el caso convencional en los subcasos peso no-mayor y mayor a 5 kilos. Contemplando las dos posibles modelaciones lógicas, esto habría correspondido a las dos siguientes postcondiciones:

$$
\begin{array}{l}
(\neg e\ \ \wedge\ \ ((p \leqslant 5\ \wedge\ c = 300)\ \vee\ (p > 5\ \wedge\ c = 300 + 40 * \lceil p - 5 \rceil))) \\
\quad \vee \\
(e\ \ \wedge\ \ c = 100 * \lceil p/5 \rceil)
\end{array}
$$

y

$$
\begin{array}{l}
(\neg e\ \ \Rightarrow\ \ ((p \leqslant 5\ \Rightarrow\ c = 300)\ \wedge\ (p > 5\ \Rightarrow\ c = 300 + 40 * \lceil p - 5 \rceil))) \\
\quad \wedge \\
(e\ \ \Rightarrow\ \ c = 100 * \lceil p/5 \rceil)\ \ .
\end{array}
$$

Es un ejercicio interesante de lógica el demostrar que estas dos fórmulas y las dos postcondiciones antes presentadas son todas equivalentes entre sí.

También hemos podido haber hecho la jerarquización de casos en sentido contrario, aun cuando para este problema no habría sido muy conveniente: primero analizando si el peso es no-mayor o mayor a 5 kilos, y luego subdividiendo estos casos en tarifa convencional o especial. Esto habría generado fórmulas lógicas con la siguiente estructura:

$$(p \leqslant 5\ \wedge\ \boxed{?})\ \vee\ (p > 5\ \wedge\ \boxed{?})$$

y

$$(p \leqslant 5\ \Rightarrow\ \boxed{?})\ \wedge\ (p > 5\ \Rightarrow\ \boxed{?})\ \ .$$

Le invitamos a que complete Ud. estas dós fórmulas y le invitamos también a que demuestre que sus propuestas son equivalentes a todas las fórmulas anteriores.

Fin de casos y subcasos, es equivalente.

3. Secuencias y Cuantificaciones

Los problemas algorítmicos más interesantes son usualmente aquellos en los que hay que manipular una cantidad no predeterminada de datos de entrada o generar una cantidad no predeterminada de resultados de salida, a diferencia de los problemas que hemos manejado en las secciones anteriores, en lo que siempre se tenía una cantidad fija y predeterminada de datos de entrada y de resultados de salida (uno o dos o tres o alguna otra cantidad). En esta sección presentaremos problemas de este nuevo tipo.

En problemas con cantidades no predeterminadas de datos o resultados, éstos son manejados mediante variables capaces de almacenar varios valores. En estas notas declararemos a tales variables como de tipo "secuencia", utilizando la sintaxis "*sequence of T*" siendo T algún otro tipo. Nuestras secuencias siempre serán homogéneas, en el sentido de que todos los valores que compongan una secuencia deben tener el mismo tipo. A tal tipo se le llama el tipo base de la secuencia, el cual puede ser cualquier otro tipo y corresponde al genérico T utilizado arriba. Por ejemplo, para que una variable pueda almacenar una secuencia de enteros como $\langle -7, 3, 3, 2, -7, 0, 1, 3, -1 \rangle$ o $\langle 9, 5, 4, 2 \rangle$ debe ser declarada con tipo "*sequence of int*", para que una variable pueda almacenar una secuencia de valores lógicos booleanos como $\langle \mathit{true}, \mathit{false}, \mathit{true}, \mathit{true} \rangle$ debe ser declarada con tipo "*sequence of boolean*", para que una variable pueda almacenar una secuencia de números reales como $\langle 23{,}2\,;\, -45{,}3\,;\, 7{,}5\,;\, 7{,}5\,;\, -1{,}0 \rangle$ o $\langle 3{,}14\,;\, 2{,}73 \rangle$ debe ser declarada con tipo "*sequence of real*" (note que en las secuencias de reales hemos utilizado el ";" como separador de los elementos para evitar ambigüedad con la "," que en cada número real separa la parte entera del resto del número), etcétera. Como el tipo base de una secuencia puede ser cualquier tipo, una secuencia también podría contener secuencias. Por ejemplo, una variable de tipo "*sequence of sequence of int*" puede almacenar la secuencia $\langle \langle 10 \rangle, \langle 4, 4 \rangle, \langle 10, 20, 30 \rangle \rangle$.

Los únicos dos operadores que permitiremos sobre secuencias serán el operador de longitud, denotado de manera similar al operador de valor absoluto "$|\_|$", y el operador de indexación, denotado por medio de corchetes "$\_\,[\,\_\,]$". En relación con el operador de longitud, si las secuencias de enteros *s0*, *s1* y *s2* tienen valores $\langle -7, 3, 3, 2, -7, 0, 1, 3, -1 \rangle$, $\langle 9, 5, 4, 2 \rangle$ y $\langle\,\rangle$, respectivamente, sus longitudes serán 9, 4 y 0, respectivamente. Esto es, $|s0| = 9$, $|s1| = 4$ y $|s2| = 0$. En relación con el operador de indexación, éste permite obtener los elementos que se encuentran dentro de una secuencia, entendiendo que el orden en que se encuentran tales elementos es relevante y la numeración empieza en cero y continúa de manera consecutiva. Así, para *s0* se tendrá que $s0[0] = -7$, $s0[1] = 3$, $s0[2] = 3$, $s0[2] = 2$ y así sucesivamente hasta $s0[8] = -1$. Nótese entonces que para cualquier secuencia s, la expresión $s[i]$ sólo puede ser utilizada para valores enteros i tales que se cumpla $0 \leqslant i < |s|$.

Ahora bien, la mayoría de los problemas algorítmicos que involucran secuencias, ya sea de datos de entrada o de resultados de salida, requieren que se usen cuantificaciones para especificar sus pre/post-condiciones. Esto se debe a que las cuantificaciones corresponden a la generalización natural de operadores binarios para secuencias: un operador binario (asociativo y conmutativo) aplicable a dos valores es generalizado mediante cuantificaciones para que sea aplicado a una secuencia de valores. Así, es mediante cuantificaciones que podremos expresar condiciones interesantes aplicables a secuencias.

Dicho todo esto, es importante señalar que la especificación de problemas que involucran secuencias se realiza con frecuencia utilizando cuantificaciones. Sin embargo, también hay problemas que involucran secuencias que pueden ser especificados sin cuantificadores y, viceversa, problemas sin secuencias que son especificados haciendo uso de cuantificadores. Procedamos entonces a presentar ejemplos de problemas algorítmicos que involucren secuencias o cuantificaciones.

3.0. Porcentajes Estatales, Regionales y Nacional de Aceptación

Volvamos al problema de la sección 1.0 referente al porcentaje de aceptación de alguna cierta actuación del gobierno ante un cierto evento, pero supongamos ahora que la encuesta ha sido realizada en varios estados del país y que los datos se tienen separadamente por estado. Esto quiere decir que la información sobre cantidad de personas encuestadas y cantidad de respuestas favorables se tiene individualizada por cada estado. Con esta información se desea calcular porcentajes de aceptación a nivel estatal, regional y nacional.

A diferencia de nuestro problema original 1.0, en el que contábamos con los datos individuales t y f ($t$otal y $f$avorables), en este nuevo problema tendremos ese par de datos de entrada para cada uno de los estados del país. Así, siendo n la cantidad de estados, los datos de entrada estarán almacenados en dos secuencias st y sf, llamadas así por $s$ecuencia de $t$otales y $s$ecuencia de $f$avorables, ambas de longitud n, que contendrán los datos en cuestión. Esto quiere decir que estamos considerando a los n estados del país como numerados de 0 a $n-1$, y que la cantidad de personas encuestadas y la cantidad de respuestas favorables del estado i están almacenadas en $st[i]$ y $sf[i]$, respectivamente.

Por ejemplo, si nuestros datos de entrada correspondiesen a 7 estados con $n = 7$ y las secuencias $st = \langle\, 800, 750, 380, 750, 750, 120, 1000 \,\rangle$ y $sf = \langle\, 250, 216, 221, 89, 654, 32, 600 \,\rangle$, esto querría decir que en el estado 0 fueron encuestadas 800 personas de las cuales 250 respondieron favorablemente, en el estado 1 fueron encuestadas 750 personas de las cuales 216 respondieron favorablemente, y así sucesivamente hasta el estado 6, en el que fueron encuestadas 1000 personas de las cuales 600 respondieron favorablemente.

Especifiquemos entonces el problema de cálculo del porcentaje nacional de aceptación. En este caso nuestro problema es similar al original de la sección 1.0, salvo que el total de personas encuestadas corresponde a la sumatoria de todos los elementos de la secuencia st y la cantidad de respuestas favorables corresponde a la sumatoria de todos los elementos de la secuencia sf. La especificación formal sería entonces la que se indica a continuación:

$$
\begin{array}{l}
[\![\ \ \textbf{const}\ \ n : \mathit{int}\,; \\
\qquad\quad\ \ st, sf : \mathit{sequence\ of\ int}\,; \\
\quad \textbf{var}\ \ pn : \mathit{real}\,; \\
\quad \left\{ \begin{array}{l} n > 0 \ \wedge\ |st| = n \ \wedge\ |sf| = n \\ \quad \wedge\ \ (\forall\, i\ :\ 0 \leqslant i < n\ :\ st[i] > 0 \ \wedge\ 0 \leqslant sf[i] \leqslant st[i]\,) \end{array} \right\} \\
\quad \textsc{CalcularPorcentajeNacionalAceptación} \\
\quad \{\ \ pn\ =\ ((\sum i : 0 \leqslant i < n : sf[i]\,)\,/\,(\sum i : 0 \leqslant i < n : st[i]\,)) * 100\ \ \} \\
]\!]\ \ .
\end{array}
$$

Note que la cuantificación universal de la precondición señala el mismo requerimiento que nuestro problema original 1.0, salvo que éste se exige estado por estado. Note además que es imprescindible exigir que haya al menos un estado, esto es, exigir que n sea positivo, para evitar la posibilidad de que no haya ninguna persona encuestada, en cuyo caso el denominador de la división especificada en la postcondición podría ser cero.

Con la información de la encuesta realizada a nivel nacional podemos también especificar el problema algorítmico correspondiente a calcular porcentajes estatales de aceptación. Esto es, con los mismos datos de entrada anteriores y además un número de estado e, se puede requerir el porcentaje estatal de aceptación correspondiente al estado e. El problema podría especificarse

como sigue:

$$
\begin{array}{l}
[\![\ \ \textbf{const}\ \ n, e : \mathit{int}\,; \\
\qquad\qquad\ \ \mathit{st}, \mathit{sf} : \mathit{sequence\ of\ int}\,; \\
\quad \textbf{var}\ \ \mathit{pe} : \mathit{real}\,; \\
\quad \left\{ \begin{array}{l} 0 \leqslant e < n \ \ \wedge\ \ |\mathit{st}| = n \ \ \wedge\ \ |\mathit{sf}| = n \\ \qquad \wedge\ \ (\,\forall\, i\ :\ 0 \leqslant i < n\ :\ \mathit{st}[i] > 0\ \ \wedge\ \ 0 \leqslant \mathit{sf}[i] \leqslant \mathit{st}[i]\,) \end{array} \right\} \\
\quad \textsc{CalcularPorcentajeEstatalAceptación} \\
\quad \{\ \ \mathit{pe}\ =\ (\mathit{sf}[e]\ /\ \mathit{st}[e]) * 100\ \ \} \\
[\!]\ \ .
\end{array}
$$

Note que el nuevo requerimiento $0 \leqslant e < n$ de la precondición, el cual exige que el número de estado e sea un número válido, implica al requerimiento anterior $n > 0$ por transitividad, razón por la cual el requerimiento anterior puede omitirse para evitar redundancia.

Más interesante que el problema de cálculo de porcentajes estatales es el problema de cálculo de porcentajes regionales, entendiendo que una región no es más que un conjunto no-vacío de estados. Por ejemplo, si tuviésemos de nuevo que nuestros datos de entrada correspondiesen a 7 estados, con $n = 7$, $\mathit{st} = \langle\, 800, 750, 380, 750, 750, 120, 1000 \,\rangle$ y $\mathit{sf} = \langle\, 250, 216, 221, 89, 654, 32, 600 \,\rangle$, y si tuviésemos que la región noroeste del país comprende a los estados numerados 1, 4 y 5, tendríamos entonces que el porcentaje de aceptación en la región noroeste del país sería $((216 + 654 + 32)\ /\ (750 + 750 + 120)) * 100$, esto es, $(902/1620) * 100$, lo cual da un resultado aproximado de 55,68 %. Este nuevo problema necesita entonces un nuevo dato de entrada que permita determinar cuáles son los estados que conforman la región cuyo porcentaje de aceptación se desea calcular. Esta información puede venir dada en una secuencia de booleanos cuyos valores *true* señalen los estados deseados, usando por tanto *false* para los estados a ser ignorados. En nuestro ejemplo, la región noroeste estaría dada por la secuencia de booleanos $\langle\, \mathit{false}, \mathit{true}, \mathit{false}, \mathit{false}, \mathit{true}, \mathit{true}, \mathit{false} \,\rangle$. La especificación formal sería la siguiente:

$$
\begin{array}{l}
[\![\ \ \textbf{const}\ \ n : \mathit{int}\,; \\
\qquad\qquad\ \ \mathit{st}, \mathit{sf} : \mathit{sequence\ of\ int}\,; \\
\qquad\qquad\ \ r : \mathit{sequence\ of\ boolean}\,; \\
\quad \textbf{var}\ \ \mathit{pr} : \mathit{real}\,; \\
\quad \left\{ \begin{array}{l} n > 0 \ \ \wedge\ \ |\mathit{st}| = n \ \ \wedge\ \ |\mathit{sf}| = n \\ \qquad \wedge\ \ (\,\forall\, i\ :\ 0 \leqslant i < n\ :\ \mathit{st}[i] > 0\ \ \wedge\ \ 0 \leqslant \mathit{sf}[i] \leqslant \mathit{st}[i]\,) \\ \qquad \wedge\ \ (\,\exists\, i\ :\ 0 \leqslant i < n\ :\ r[i]\,) \end{array} \right\} \\
\quad \textsc{CalcularPorcentajeRegionalAceptación} \\
\quad \{\ \ \mathit{pr}\ =\ ((\,\textstyle\sum i\ :\ 0 \leqslant i < n\ \wedge\ r[i]\ :\ \mathit{sf}[i]\,)\ /\ (\,\textstyle\sum i\ :\ 0 \leqslant i < n\ \wedge\ r[i]\ :\ \mathit{st}[i]\,)) * 100\ \ \} \\
[\!]\ \ .
\end{array}
$$

La nueva postcondición es como la postcondición correspondiente al problema del porcentaje nacional de aceptación, salvo que las sumatorias no deben considerar incondicionalmente a todos los estados i sino sólo a los estados para los que la secuencia indicadora de la región r señale *true*, esto es, los estados i para los que la fórmula booleana $r[i]$ se satisface. Note que el nuevo requerimiento de la precondición conformado por un cuantificador existencial exige que la región tenga al menos un estado. Esto es imprescindible para evitar que el denominador de la división incluida en la postcondición sea cero.

Por último, especifiquemos el problema de calcular los porcentajes estatales máximo y mínimo de aceptación. Nos concentraremos sólo en dar como salida estos dos porcentajes, sin importar en

cuáles estados se dieron tales resultados. La formalización de la especificación sería la siguiente:

$$
\begin{array}{l}
[\![\ \ \textbf{const}\ \ n : \mathit{int}\,; \\
\qquad\quad\ \ \mathit{st}, \mathit{sf} : \mathit{sequence\ of\ int}\,; \\
\quad \textbf{var}\ \ \mathit{pmax}, \mathit{pmin} : \mathit{real}\,; \\
\quad \left\{ \begin{array}{l} n > 0 \ \wedge\ |\mathit{st}| = n \ \wedge\ |\mathit{sf}| = n \\ \quad \wedge\ (\forall\, i \,:\, 0 \leqslant i < n \,:\, \mathit{st}[i] > 0 \ \wedge\ 0 \leqslant \mathit{sf}[i] \leqslant \mathit{st}[i]\,) \end{array} \right\} \\
\quad \textsc{CalcularPorcentajesEstatalesExtremosAceptación} \\
\quad \left\{ \begin{array}{l} \mathit{pmax} \;=\; (\,\textbf{max}\, i \,:\, 0 \leqslant i < n \,:\, (\mathit{sf}[i] \,/\, (\mathit{st}[i]) * 100\,) \\ \quad \wedge \\ \mathit{pmin} \;=\; (\,\textbf{min}\ i \,:\, 0 \leqslant i < n \,:\, (\mathit{sf}[i] \,/\, (\mathit{st}[i]) * 100\,) \end{array} \right\} \\
]\!]\ \ .
\end{array}
$$

3.1. Factibilidad de Vuelto

Volvamos ahora a nuestro problema 1.1, relacionado con el cálculo de vueltos de una máquina dispensadora de chucherías del país Cualquieralandia utilizando monedas de 1, 2 y 5 *cualcos*. Consideremos ahora una versión un poco más real del problema, en el que no sólo recibimos como entrada la cantidad de dinero introducida por alguna persona en la máquina y el precio de la chuchería requerida por tal persona (datos que nos permiten calcular el monto total correspondiente al vuelto), sino que también recibimos como entrada la cantidad de monedas de cada una de las tres unidades que la máquina contiene, pues de estas disponibilidades depende que la máquina pueda realmente dar vuelto.

Nuestro nuevo problema corresponde entonces a, dados el precio de la chuchería requerida, la cantidad de dinero que la persona introdujo en la máquina, y la cantidad de monedas disponibles de 1, 2 y 5 *cualcos* que contiene la máquina, determinar si es posible dar vuelto con la disponibilidad actual. Por ahora nos olvidaremos de dar vuelto en sí mismo y nos concentraremos sólo en determinar si es factible dar vuelto.

Los datos de entrada incluyen los mismos p y t del problema original 1.1, correspondientes al precio de la chuchería solicitada y a la cantidad total de dinero entregada a la máquina, y a estos datos de entrada les anexamos las tres cantidades *d1*, *d2* y *d5* de monedas de 1, 2 y 5 *cualcos*, respectivamente, disponibles en la máquina. Nuestro único resultado de salida será un booleano f que señale si es factible dar vuelto o no. Entendiendo el nombre f como abreviatura de "$f$actible", un valor resultante de *true* significará que sí es posible dar vuelto y un valor resultante de *false* significará que no lo es.

Proponemos entonces la siguiente especificación formal:

$$
\begin{array}{l}
[\![\ \ \textbf{const}\ \ p, t, \mathit{d1}, \mathit{d2}, \mathit{d5} : \mathit{int}\,; \\
\quad \textbf{var}\ \ f : \mathit{boolean}\,; \\
\quad \{\ 0 < p \leqslant t \ \wedge\ \mathit{d1} \geqslant 0 \ \wedge\ \mathit{d2} \geqslant 0 \ \wedge\ \mathit{d5} \geqslant 0\ \} \\
\quad \textsc{DeterminarFactibilidadDeVuelto} \\
\quad \left\{ \begin{array}{l} f \;\equiv\; (\exists\, x, y, z \,:\, x, y, z \in \mathbb{Z} \ \wedge\ 0 \leqslant x \leqslant \mathit{d1} \ \wedge\ 0 \leqslant y \leqslant \mathit{d2} \ \wedge\ 0 \leqslant z \leqslant \mathit{d5} \,: \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad x + 2 * y + 5 * z \;=\; t - p\,) \end{array} \right\} \\
]\!]\ \ .
\end{array}
$$

Nuestra precondición propuesta sólo señala requerimientos básicos sobre la entrada, mientras que

la postcondición expresa que el booleano resultado f debe ser igual a la existencia de un vuelto dentro de la disponibilidad indicada por la entrada.

Combinemos ahora nuestro problema de factibilidad con el problema original de determinar un posible vuelto. Esto es, enriquezcamos nuestros resultados de salida teniendo, además del booleano f de factibilidad, tres enteros $v1$, $v2$ y $v5$ que correspondan a un vuelto. Sin embargo, es muy importante notar que estos tres enteros indicarán un vuelto con sentido sólo en el que caso de que f indique que es posible determinar un vuelto. Una posible especificación formal es la siguiente:

$$
\begin{array}{l}
[\![\ \ \textbf{const}\ \ p, t, d1, d2, d5 : int\,; \\
\quad \textbf{var}\ \ f : boolean\,; \\
\qquad\quad v1, v2, v5 : int\,; \\
\quad \{\ \ 0 < p \leqslant t\ \ \wedge\ \ d1 \geqslant 0\ \ \wedge\ \ d2 \geqslant 0\ \ \wedge\ \ d5 \geqslant 0\ \ \} \\
\quad \textsc{CalcularUnVueltoSiSeDeterminaFactible} \\
\quad \left\{ \begin{array}{l}
(f\ \equiv\ (\exists\, x, y, z\ :\ x, y, z \in \mathbb{Z}\ \ \wedge\ \ 0 \leqslant x \leqslant d1\ \ \wedge\ \ 0 \leqslant y \leqslant d2\ \ \wedge\ \ 0 \leqslant z \leqslant d5\ : \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad x + 2 * y + 5 * z\ =\ t - p\,)) \\
\quad \wedge \\
(f\ \Rightarrow\ \ 0 \leqslant v1 \leqslant d1\ \ \wedge\ \ 0 \leqslant v2 \leqslant d2\ \ \wedge\ \ 0 \leqslant v5 \leqslant d5 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ \ v1 + 2 * v2 + 5 * v5\ =\ t - p\,)
\end{array} \right\} \\
]\!]\ \ .
\end{array}
$$

La postcondición que hemos propuesto es la conjunción de dos requerimientos, una equivalencia y una implicación. El primer requerimiento de nuestra postcondición, la equivalencia, señala cuál es el valor deseado para f al igual que en la versión anterior del problema. El segundo requerimiento de la postcondición, la implicación, señala lo que se desea como valores resultantes para $v1$, $v2$ y $v5$, pero sólo cuando f es verdadera. En términos lógicos, si la equivalencia del primer requerimiento determina que f debe ser falsa, en ese caso la implicación del segundo requerimiento queda inmediatamente satisfecha independientemente de los valores de $v1$, $v2$ y $v5$ (ya que una implicación es verdadera cuando su antecendente es falso independientemente del valor de su consecuente). Por otra parte, si la equivalencia del primer requerimiento determina que f debe ser verdadera, en ese caso la implicación del segundo requerimiento exige valores adecuados para $v1$, $v2$ y $v5$ (ya que una implicación con antecendente verdadero será verdadera sólo si su consecuente también es verdadero).

Por ejemplo, con el estado inicial

$$[\,p = 3\,,\ t = 10\,,\ d1 = 1\,,\ d2 = 0\,,\ d5 = 2\,,\ f = \ldots,\ v1 = \ldots,\ v2 = \ldots,\ v5 = \ldots\,]\ \ ,$$

del cual podemos ignorar los valores de las variables f, $v1$, $v2$ y $v5$ (razón por la cual hemos dejado sus valores sin especificar con puntos suspensivos), no es posible dar vuelto. Debido a esto, será aceptable tanto el estado final

$$[\,p = 3\,,\ t = 10\,,\ d1 = 1\,,\ d2 = 0\,,\ d5 = 2\,,\ f = false\,,\ v1 = -8\,,\ v2 = 64\,,\ v5 = 0\,]$$

como el estado final

$$[\,p = 3\,,\ t = 10\,,\ d1 = 1\,,\ d2 = 0\,,\ d5 = 2\,,\ f = false\,,\ v1 = 491\,,\ v2 = -98\,,\ v5 = -71\,]\ \ ,$$

y cualquier otro estado final en el que f sea falsa y las variables declaradas como constantes mantengan sus valores iniciales, teniendo las variables $v1$, $v2$ y $v5$ cualquier valor. Todo esto es

consecuencia de que, siendo f falsa, la implicación del segundo requerimiento de la postcondición es verdadera en todos estos estados finales sin importar los valores de *v1*, *v2* y *v5*.

En contraposición con el ejemplo anterior, para el estado inicial

$$[\, p = 3\,,\, t = 10\,,\, d1 = 23\,,\, d2 = 0\,,\, d5 = 2\,,\, f = \ldots\,,\, v1 = \ldots\,,\, v2 = \ldots\,,\, v5 = \ldots\,] \quad ,$$

sí será posible dar vuelto. Serían aceptables entonces tanto el estado final

$$[\, p = 3\,,\, t = 10\,,\, d1 = 23\,,\, d2 = 0\,,\, d5 = 2\,,\, f = \mathit{true}\,,\, v1 = 7\,,\, v2 = 0\,,\, v5 = 0\,]$$

como el estado final

$$[\, p = 3\,,\, t = 10\,,\, d1 = 23\,,\, d2 = 0\,,\, d5 = 2\,,\, f = \mathit{true}\,,\, v1 = 2\,,\, v2 = 0\,,\, v5 = 1\,] \quad .$$

En este caso, siendo f verdadera, la implicación del segundo requerimiento de la postcondición sí exige valores apropiados para *v1*, *v2* y *v5*.

Especifiquemos ahora una versión aún más completa de este problema considerando que, cuando se señale un vuelto factible, la disponibilidad de monedas de la máquina deberá ser actualizada. Esto significa que las variables con datos de entrada *d1*, *d2* y *d5* ya no serán declaradas como constantes y pasarán a cumplir el doble papel de variables con datos de entrada y con resultados de salida. Este nuevo problema puede ser especificado formalmente de la siguiente manera:

$$
\begin{array}{l}
[\![\ \ \textbf{const}\ \ p, t : \mathit{int}\,; \\
\quad \textbf{var}\ \ d1, d2, d5, v1, v2, v5 : \mathit{int}\,; \\
\qquad\quad f : \mathit{boolean}\,; \\
\quad \{\ 0 < p \leqslant t\ \wedge\ d1 \geqslant 0\ \wedge\ d2 \geqslant 0\ \wedge\ d5 \geqslant 0\ \wedge\ d1 = X\ \wedge\ d2 = Y\ \wedge\ d5 = Z\ \} \\
\quad \textsc{CalcularUnVueltoActualizandoDisponibilidadSiSeDeterminaFactible} \\
\quad \left\{ \begin{array}{l}
(f \equiv (\exists\, x, y, z : x, y, z \in \mathbb{Z}\ \wedge\ 0 \leqslant x \leqslant X\ \wedge\ 0 \leqslant y \leqslant Y\ \wedge\ 0 \leqslant z \leqslant Z : \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad x + 2 * y + 5 * z\ =\ t - p\,)) \\
\quad \wedge \\
(f\ \Rightarrow\ 0 \leqslant v1 \leqslant X\ \wedge\ 0 \leqslant v2 \leqslant Y\ \wedge\ 0 \leqslant v5 \leqslant Z \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ \ v1 + 2 * v2 + 5 * v5\ =\ t - p \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ \ d1 = X - v1\ \wedge\ d2 = Y - v2\ \wedge\ d5 = Z - v5\,) \\
\quad \wedge \\
(\neg f\ \Rightarrow\ d1 = X\ \wedge\ d2 = Y\ \wedge\ d5 = Z\,)
\end{array} \right\} \\
[\!]\ \ .
\end{array}
$$

En esta nueva formalización es necesario utilizar variables de especificación para poder diferenciar en la postcondición los valores iniciales de las variables *d1*, *d2* y *d5* de los valores finales de las mismas. Las variables de especificación utilizadas para esto son X, Y y Z.

El segundo requerimiento de la postcondición corresponde a la implicación de la versión anterior, pero ahora no sólo se especifican condiciones para *v1*, *v2* y *v5* sino también para los valores finales de *d1*, *d2* y *d5*. Además, a diferencia de la versión anterior, utilizamos X, Y y Z para referirnos a los valores iniciales de *d1*, *d2* y *d5*, lo cual era antes innecesario ya que estas variables no podían cambiar de valor.

Por último, la postcondición tiene un nuevo tercer requerimiento: una implicación que exige a las variables *d1*, *d2* y *d5* mantener sus valores iniciales en caso de que se determine que dar vuelto es infactible. Nótese que, en caso de infactibilidad de vuelto, los valores de *v1*, *v2* y *v5* siguen siendo irrelevantes.

Redondeemos con un par de ejemplos: con estado inicial

$$[\, p = 3\,,\, t = 10\,,\, \mathit{d1} = 1\,,\, \mathit{d2} = 0\,,\, \mathit{d5} = 2\,,\, f = \ldots,\, \mathit{v1} = \ldots,\, \mathit{v2} = \ldots,\, \mathit{v5} = \ldots\,] \quad ,$$

será aceptable tanto el estado final

$$[\, p = 3\,,\, t = 10\,,\, \mathit{d1} = 1\,,\, \mathit{d2} = 0\,,\, \mathit{d5} = 2\,,\, f = \mathit{false}\,,\, \mathit{v1} = -8\,,\, \mathit{v2} = 64\,,\, \mathit{v5} = 0\,]$$

como el estado final

$$[\, p = 3\,,\, t = 10\,,\, \mathit{d1} = 1\,,\, \mathit{d2} = 0\,,\, \mathit{d5} = 2\,,\, f = \mathit{false}\,,\, \mathit{v1} = 491\,,\, \mathit{v2} = -98\,,\, \mathit{v5} = -71\,] \quad ,$$

entre otros posibles estados finales; por otra parte, con estado inicial

$$[\, p = 3\,,\, t = 10\,,\, \mathit{d1} = 23\,,\, \mathit{d2} = 0\,,\, \mathit{d5} = 2\,,\, f = \ldots,\, \mathit{v1} = \ldots,\, \mathit{v2} = \ldots,\, \mathit{v5} = \ldots\,] \quad ,$$

serán aceptables el estado final

$$[\, p = 3\,,\, t = 10\,,\, \mathit{d1} = 16\,,\, \mathit{d2} = 0\,,\, \mathit{d5} = 2\,,\, f = \mathit{true}\,,\, \mathit{v1} = 7\,,\, \mathit{v2} = 0\,,\, \mathit{v5} = 0\,]$$

y el estado final

$$[\, p = 3\,,\, t = 10\,,\, \mathit{d1} = 21\,,\, \mathit{d2} = 0\,,\, \mathit{d5} = 1\,,\, f = \mathit{true}\,,\, \mathit{v1} = 2\,,\, \mathit{v2} = 0\,,\, \mathit{v5} = 1\,] \quad .$$

Note de nuevo que en los estados finales en los que f es falsa, los valores de *v1*, *v2* y *v5* son irrelevantes para determinar el valor de verdad de la postcondición.

3.2. *Continuará...*

Una versión futura de estas notas tendrá en esta subsección más ejemplos de problemas relacionados con secuencias y cuantificaciones...

4. *Continuará...*

Una versión futura de estas notas tendrá otra sección con más ejemplos de especificación de problemas algorítmicos interesantes...

Referencias

[ACMa] ACM (Association for Computing Machinery). Turing Award recipient in 1972: E.W. Dijkstra. `www.acm.org/awards/taward.html`.

[ACMb] ACM (Association for Computing Machinery). Turing Award recipient in 1980: C.A.R. Hoare. `www.acm.org/awards/taward.html`.

[APM08] P. Audebaud and C. Paulin-Mohring, editors. *Mathematics of Program Construction – 9th International Conference MPC 2008*. Number 5133 in Lecture Notes in Computer Science. Springer, Marseille, France, 2008.

[Bac80] R.-J.R. Back. Correctness preserving program refinements: Proof theory and applications. Tract 131, Mathematisch Centrum, Amsterdam, 1980.

[Bac88] R.-J.R. Back. A calculus of refinements for program derivations. *Acta Informatica*, 25:593–624, 1988.

[Bac03] R.C. Backhouse. *Program Construction: Calculating Implementations from Specifications*. John Wiley & Sons, 2003.

[BW98] R.-J.R. Back and J. von Wright. *Refinement Calculus*. Graduate Texts in Computer Science. Springer-Verlag, 1998.

[DF88] E.W. Dijkstra and W.H.J. Feijen. *A Method of Programming*. Addison-Wesley, 1988.

[Dij75] E.W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[Dij76] E.W. Dijkstra. *A Discipline of Programming*. Series in Automatic Computation. Prentice Hall, 1976.

[DS90] E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics*. Texts and Monographs in Computer Science. Springer-Verlag, 1990.

[Gri81] D. Gries. *The Science of Programming*. Texts and Monographs in Computer Science. Springer-Verlag, 1981.

[GS94a] D. Gries and F.B. Schneider. *A Logical Approach to Discrete Math*. Texts and Monographs in Computer Science. Springer-Verlag, 1994.

[GS94b] D. Gries and F.B. Schneider. HISTORICAL NOTE 10.1. Edsger W. Dijkstra (1930–). Página 183 de [GS94a], 1994.

[GS94c] D. Gries and F.B. Schneider. HISTORICAL NOTE 1.4. C.A.R. Hoare (1934–). Página 19 de [GS94a], 1994.

[HJ98] C.A.R. Hoare and He Jifeng. *Unifying Theories of Programming*. International Series in Computer Science. Prentice Hall, 1998.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 583, 1969.

[Kal90] A. Kaldewaij. *Programming: The Derivation of Algorithms*. International Series in Computer Science. Prentice Hall, 1990.

[Mez00] O. Meza. Introducción a la programación. Departamento de Computación y Tecnología de la Información, Universidad Simón Bolívar. (Disponible vía la página web del Prof. Meza, `www.ldc.usb.ve/~meza`; buscar enlace "Algoritmos y Estructuras I (CI-2611/91)" y luego "Material del Curso"; consultado por última vez en Octubre 2009.), 2000.

[Mor87] J.M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming*, 9(3):287–306, 1987.

[Mor88] C.C. Morgan. The specification statement. *ACM Transactions on Programming Languages and Systems*, 10(3):403–419, 1988. Reprinted in [MV94].

[Mor94] C.C. Morgan. *Programming from Specifications*. International Series in Computer Science. Prentice Hall, 2nd edition, 1994.

[MV94] C.C. Morgan and T.N. Vickers, editors. *On the Refinement Calculus*. FACIT Series in Computer Science. Springer-Verlag, 1994.

[SL95a] D. Shasha and C. Lazere. Edsger W. Dijkstra – Appalling Prose and the Shortest Path. En [SL95b], páginas 55–67, 1995.

[SL95b] D. Shasha and C. Lazere. *Out of Their Minds – The Lives and Discoveries of 15 Great Computer Scientists*. Copernicus, 1995.

[Uus06] T. Uustalu, editor. *Mathematics of Program Construction – 8th International Conference MPC 2006*. Number 4014 in Lecture Notes in Computer Science. Springer, Kuressaare, Estonia, 2006.

J.N. Ravelo / Octubre 2009