Especificación e Implementación de Tipos Abstractos de Datos

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

Resumen

Estas notas presentan un esquema de especificación de tipos abstractos de datos, abreviado TADs, acompañado de técnicas de implementación de estos TADs que garantizan la consistencia entre especificaciones e implementaciones dadas.

El estilo de especificación de TADs utilizado es el llamado *basado en modelos*, y la teoría en la que se fundamenta la correctitud de las implementaciones con respecto a sus respectivas especificaciones es la de *refinamiento de datos*.

0. Introducción

La construcción de un nuevo tipo de datos requiere que las características deseadas de éste sean inicialmente especificadas de manera clara, para luego proceder a implementarlo. A tales nuevos tipos de datos los llamaremos *tipos abstractos de datos*, término que abreviaremos como *TADs*. Este nombre se debe a que el comportamiento deseado de estos tipos debe ser especificado de manera abstracta, independientemente de las múltiples posibles implementaciones concretas que puedan luego construirse.

Los estilos de especificación de TADs más conocidos y utilizados son el estilo *algebraico* y el estilo *basado en modelos*. Especificaciones en el estilo algebraico pueden ser conseguidas en, por ejemplo, [Mit92]. En estas notas utilizaremos el estilo basado en modelos. Un ejemplo de presentación de especificaciones de TADs usando el estilo basado en modelos puede encontrarse en [Lis01]. Sin embargo, en [Lis01] se utilizan modelos de especificación descritos principalmente en lenguaje natural, mientras en estas notas presentaremos tales modelos no sólo en lenguaje natural (siempre imprescindible para nuestra comprensión como seres humanos) sino también por medio de estructuras matemáticas como, por ejemplo, conjuntos y relaciones, entre otras estructuras.

En estas notas, cada nuevo TAD será presentado primero en lenguaje natural, a efectos de lograr una primera aproximación al nuevo tipo de la forma más clara y sencilla posible. Una vez hecho esto, procederemos a la especificación formal del nuevo TAD usando un *modelo abstracto* de representación, basado en estructuras matemáticas como las antes señaladas, esto es, conjuntos, relaciones, etcétera. De esta manera, con una especificación formal, se logra que a la primera aproximación al tipo presentada en lenguaje natural se le "pulan los detalles opacos" con una

buena y precisa formalización. Esto es, se eliminan posibles dudas de interpretación del comportamiento del nuevo tipo, las cuales usualmente surgen debido al caracter inherentemente informal del lenguaje natural, que da pie a ambigüedades e inexactitudes en la descripción.

Una vez especificado formalmente un TAD, se desea, por supuesto, implementarlo. Para esto, lo primero que debe hacerse es escoger una estructura de datos que lo soporte, a la cual llamaremos el *modelo concreto* de representación escogido. El TAD es entonces re-especificado en términos del nuevo modelo concreto, esto es, en términos de la estructura de datos escogida. Esta re-especificación es la que entonces debe ser implementada, programando todas las subrutinas (procedimientos, funciones, o métodos) correspondientes a las operaciones del TAD.

Sin embargo, antes de proceder a la implementación final de la re-especificación del TAD, se debe garantizar la consistencia entre la re-especificación del TAD con modelo concreto y la especificación original del TAD con modelo abstracto. Esto se logra utilizando técnicas de *refinamiento de datos*.

La presentación inicial de todos los aspectos involucrados en el estilo de especificación de TADs basado en modelos, y de todos los pormenores de la técnica de refinamiento de datos, será hecha utilizando permanentemente como ejemplo a un TAD *Diccionario*. Una vez completada toda la presentación del estilo de especificación y de las técnicas de implementación, se mostrarán otros ejemplos interesantes de TADs, algunos de ellos "clásicos", en el sentido de ser TADs de mucho uso en las ciencias e ingeniería de la computación, como *Cola*, *Pila*, *Conjunto*, *Multiconjunto* y *Grafo*. También serán presentados otros TADs que no son clásicos, sino simplemente ejemplos interesantes.

1. Un TAD *Diccionario*

En el Diccionario RAE (de la Real Academia Española), el término "diccionario" aparece con las siguientes acepciones: (i) "Libro en el que se recogen y explican de forma ordenada voces de una o más lenguas, de una ciencia o de una materia determinada.", y (ii) "Catálogo numeroso de noticias importantes de un mismo género, ordenado alfabéticamente. [Por ejemplo:] Diccionario bibliográfico, biográfico, geográfico."

La clase de TAD *Diccionario* de nuestro interés es cercana a la acepción (ii), en el hecho de que nos interesa que nuestros diccionarios correspondan a catálogos, salvo que no necesariamente almacenarán "noticias. . . de un mismo género", sino objetos cualesquiera siempre que, eso sí, sean "de un mismo género". Estos objetos listados o catalogados en un *Diccionario* deberán estar acompañados de una clave que permita identificarlos unívocamente, de manera similar a como los números de cédula de identidad identifican a personas venezolanas y extranjeras residentes en Venezuela.

Así, nuestro TAD *Diccionario* permitirá almacenar pares clave/valor, de forma tal que las claves sean únicas e identifiquen unívocamente a su correspondiente valor, esto es, el objeto o cosa asociado con cada clave.

Podríamos conectar a la acepción (i) del término "diccionario" en el Diccionario RAE con nuestro TAD *Diccionario*, teniendo como claves a las distintas palabras que pueden aparecer en un diccionario, y teniendo como valores a una posible acepción de cada palabra. En esto último son diferentes un diccionario (según DRAE) y un *Diccionario* (según nuestro TAD), pues en el primero una palabra puede tener muchas acepciones mientras que en el segundo cada palabra sólo puede tener una acepción (salvo que en un *Diccionario* almacenáramos una lista de acepciones como el único objeto asociado a cada palabra como clave).

Para terminar nuestra presentación informal del TAD *Diccionario*, veamos un par de ejemplos específicos.

En un *Diccionario* podríamos almacenar una pequeña agenda de cumpleaños con las fechas de cumpleaños de nuestras amistades. Las claves serían los nombres de nuestras amistades y los objetos asociados las fechas. Por ejemplo:

Claves	*Valores*
Luisa S.	25 Sept
Carlos A.	3 Jul
Elena C.	12 Dic
Ana R.	17 Ene
Juan A.	12 Dic

.

Note que no hay ningún problema en que haya fechas repetidas. Esto es, en un diccionario varias claves pueden tener asociado el mismo valor. Sin embargo, las claves deben ser únicas, por lo que si contamos con dos personas amigas con el mismo nombre e inicial de apellido, debemos agregar más información en el nombre (clave) para diferenciarlas.

También podríamos usar el TAD *Diccionario* para almacenar notas de estudiantes usando como claves a sus números de carnet:

Claves	*Valores*
2153751	3
2255182	3
2254977	5
2153710	3
2053201	4
2154111	3

.

De nuevo, note la unicidad de claves y posible repetición de valores.

Por último, note que nuestro TAD *Diccionario* es genérico, en el sentido de que acepta diversos tipos de claves (en los dos últimos ejemplos son nombres o números de carnet), siempre y cuando en un mismo *Diccionario* todas las claves sean del mismo tipo, y también diversos tipos de valores (en los dos últimos ejemplos son fechas o notas), de nuevo siempre y cuando en un mismo *Diccionario* todos los valores sean del mismo tipo. Debido a esto, definiremos nuestro TAD *Diccionario* de manera paramétrica, esto es, en términos de dos parámetros *T0* y *T1* correspondientes respectivamente al tipo de las claves y al tipo de los valores: un TAD *Diccionario* $(T0, T1)$.

2. Especificación de un TAD

En la sección anterior vimos una descripción informal en lenguaje natural de nuestro TAD *Diccionario*, para así lograr una primera aproximación sencilla al nuevo TAD. Ahora lo especificaremos formalmente.

2.0. Modelo *Abstracto* de Representación

Lo primero que necesitamos es modelar la descripción dada en lenguaje natural en términos de un *modelo abstracto* formal de representación. Tales modelos abstractos formales corresponden a estructuras matemáticas.

La descripción informal de nuestro TAD *Diccionario* (*T0*, *T1*) da a entender que nuestros diccionarios corresponden formalmente a *funciones* de claves en valores, siendo las claves de tipo *T0* y los valores de tipo *T1*, por lo cual necesitamos funciones de *T0* en *T1*. Efectivamente, utilizaremos en nuestro modelo abstracto de representación a una función de *T0* en *T1* para representar a cada diccionario.

Esta función será una función parcial $T0 \nrightarrow T1$, pues, por supuesto, cada diccionario "conoce" sólo a algunas claves y no a todo el posible dominio *T0* de claves. Esto es, por ejemplo, en mi agenda de cumpleaños el diccionario sólo "conoce" a los nombres de mis amistades como claves y no a cualquier nombre, y en la lista de notas el diccionario sólo "conoce" a los carnets de quienes estén cursando la asignatura correspondiente y no a cualquier número de carnet. Por lo tanto, la función correspondiente al diccionario será en el caso general una función parcial $T0 \nrightarrow T1$ en lugar de una función total $T0 \rightarrow T1$.

Esta función será entonces incluida en nuestra especificación formal del nuevo TAD como un atributo del modelo abstracto de representación, atributo al cual llamaremos, por analogía con las tablas de ejemplo presentadas en la sección anterior, *tabla*. La sintaxis que utilizaremos es la siguiente:

Modelo de Representación
var *tabla* : $T0 \nrightarrow T1$.

El conjunto de claves conocidas por un diccionario es extraíble del atributo *tabla*, pues es el dominio de esta función. Sin embargo, en el resto de la especificación del TAD necesitaremos con bastante frecuencia referirnos a este conjunto. Por lo tanto, para poder hacer referencia fácilmente a este conjunto, lo incluiremos como otro atributo del modelo. (Además, así nuestro ejemplo se hace un poco más interesante...)

Tendríamos ahora entonces:

Modelo de Representación
var *conoc* : set *T0*
tabla : $T0 \nrightarrow T1$.

Por último, especificaremos nuestro TAD *Diccionario* acotando la capacidad de las tablas a ser almacenadas. Así, agregaremos un atributo más al modelo abstracto de representación que indique la capacidad máxima de cada diccionario. Tal capacidad no podrá ser cambiada una vez que un diccionario sea creado, por lo cual declararemos a este atributo en el modelo de representación como atributo constante en lugar de como atributo variable.

Nuestra versión final del modelo abstracto de representación de nuestro TAD es entonces:

Modelo de Representación
const *MAX* : int
var *conoc* : set *T0*
tabla : $T0 \nrightarrow T1$.

Así como en este modelo abstracto de representación de nuestro ejemplo utilizamos un conjunto y una función, en todos nuestros modelos abstractos de representación utilizaremos estructuras

matemáticas tomadas de la teoría de conjuntos. Estas estructuras serán siempre una de las siguientes: *conjuntos*, *multiconjuntos*, *secuencias*, *relaciones* y *funciones*.

Para estandarizar la notación utilizada sobre estas estructuras, nos apoyaremos en la presentación de [Mor94, capítulo 9], donde se le dedica una sección a cada una de las cinco estructuras arriba mencionadas. En algunos pocos casos, nos desviaremos ligeramente de la notación utilizada en [Mor94] y haremos la aclaratoria correspondiente.

Continuamos ahora con el resto de la especificación...

2.1. Invariante de Representación

Una vez que han sido declarados todos los atributos del modelo de representación, debemos analizar qué características adicionales deben ser satisfechas por estos atributos (adicionales al hecho de pertenecer a los tipos indicados en la declaración del modelo).

Por ejemplo, teniendo un atributo entero, puede que nos interese que tal atributo sea un número impar, o que sea no-negativo, o que sea múltiplo de 10, etcétera. Debemos analizar qué características son necesarias para que el atributo tenga sentido.

En el caso de nuestro atributo entero MAX en nuestro modelo de representación, si éste representa la capacidad máxima de un diccionario, no tiene sentido que sea negativo. Y quizá conviene descartar también la posibilidad de que sea cero. Exigiremos entonces que sea estrictamente positivo. La sintaxis que utilizaremos es la siguiente:

Invariante de Representación
$MAX > 0$.

En el caso de un atributo de tipo conjunto, nos podría interesar exigir que no sea vacío, o que su cardinalidad sea un número par, o cualquier otra característica. En nuestro ejemplo, nos interesa limitar la cardinalidad de *conoc*, la cual denotamos como $\#\, conoc$, siguiendo la notación de [Mor94, capítulo 9]. Si MAX corresponde a la capacidad máxima de un diccionario, la cantidad de claves conocidas, esto es, $\#\, conoc$, no puede superar tal capacidad. Esto enriquece nuestro invariante de representación de la siguiente forma:

Invariante de Representación
$MAX > 0 \ \wedge \ \#\, conoc \leqslant MAX$.

Por último, en el caso de un atributo de tipo función, nos podría interesar que ésta no fuese vacía, o que los elementos de su dominio o de su rango cumplan con algunas otras restricciones. En nuestro ejemplo, el atributo función *tabla* debe ser consistente con el conjunto *conoc* de claves conocidas, en el sentido de que toda clave que esté en el dominio de la función debe ser conocida y toda clave conocida debe estar en el dominio de la función: el dominio de *tabla* tiene que ser exactamente igual a *conoc*. Esto da forma a la versión final de nuestro invariante de representación:

Invariante de Representación
$MAX > 0 \ \wedge \ \#\, conoc \leqslant MAX \ \wedge \ conoc = \mathrm{dom}\, tabla$.

En resumen, el invariante de representación impone restricciones adicionales que deben ser satisfechas por los atributos del modelo de representación, ya sean éstas restricciones individuales sobre algún atributo particular (como nuestra primera restricción sobre MAX) o exigencias de

que dos o más atributos sean consistentes entre ellos (como nuestras otras dos restricciones, que demandan consistencia entre *conoc* y *MAX*, y entre *conoc* y *tabla*, respectivamente).

En caso de que no haga falta exigir características adicionales al modelo de representación, el invariante de representación puede ser omitido, lo cual corresponde formalmente (y es equivalente) a utilizar:

Invariante de Representación
true .

2.2. Operaciones

Como última componente, la especificación de un nuevo TAD debe indicar qué operaciones son aplicables a los objetos del nuevo tipo. Note que esto es una de las principales características de todo tipo de datos. Por ejemplo, en el caso del tipo entero, las operaciones que cualquier típico lenguaje de programación ofrece sobre este tipo incluyen suma, resta, multiplicación, cociente y resto de división entera, etcétera. Para el tipo booleano, las operaciones típicas incluyen conjunción, disyunción, negación y quizá algunas otras. Los TADs, como tipos de datos que son, no son una excepción a esta regla y, por lo tanto, es componente primordial de su especificación enumerar y describir el comportamiento esperado de todas sus operaciones. En el caso de los TADs, sus operaciones serán ofrecidas en forma de subrutinas, esto es, procedimientos y funciones.

Para nuestro TAD *Diccionario*, podríamos contar con las siguientes operaciones básicas: (i) agregar un par clave/valor a un diccionario; (ii) eliminar un par clave/valor de un diccionario; (iii) dada una clave, determinar el valor asociado a tal clave en un diccionario; y (iv) dada una clave, determinar si ésta es conocida o no por un diccionario. Y, además de todas estas operaciones, necesitamos una operación de inicialización o creación de un nuevo diccionario. (Note que todas las operaciones anteriores son aplicables a un diccionario preexistente, por lo cual hace falta al menos una primera operación de creación o inicialización.)

Cada una de estas operaciones, en vista de que serán ofrecidas como procedimientos o funciones, debe ser especificada de la manera conocida: a través de un par pre/post-condición.

Crear Empecemos con la operación de creación o inicialización. Daremos a esta operación como único parámetro de entrada a la capacidad máxima deseada para el diccionario a ser creado, y la operación devolverá como parámetro de salida un diccionario con la capacidad máxima indicada. Esto determina la interfaz del procedimiento correspondiente:

proc *crear* (**in** m : int ; **out** d : *Diccionario*)
{ **Pre** : ... }
{ **Post** : ... } .

Ahora nos falta describir el comportamiento de la operación completando el par pre/post-condición. La única restricción que hace falta exigir al único parámetro de entrada es que sea un valor adecuado como capacidad máxima y, ya que en el invariante de representación lo que se especificó como valores aceptables para el atributo *MAX* fueron números positivos, pues se exigirá que m sea positivo en la precondición. En cuanto al diccionario de salida, éste deberá tener como capacidad máxima a m y en cuanto a sus pares clave/valor iniciales, lo natural es que no cuente con ninguno al principio; esto será por tanto lo indicado en la postcondición. Veamos entonces la

especificación completa de nuestra primera operación:

$$\begin{array}{l}\textbf{proc}\ \mathit{crear}\,(\,\textbf{in}\ m : \mathrm{int}\,;\ \textbf{out}\ d : \mathit{Diccionario}\,) \\ \quad \{\ \textbf{Pre}: \quad m > 0\ \} \\ \quad \{\ \textbf{Post}:\ d.\mathit{MAX} = m \ \wedge\ d.\mathit{conoc} = \emptyset \ \wedge\ d.\mathit{tabla} = \emptyset\ \} \end{array} \quad .$$

Es conveniente destacar que la notación utilizada para referirnos a cada atributo de cada diccionario, según los atributos declarados en el modelo de representación para <u>todo</u> diccionario, es un "." infijo. Por ello, el atributo *conoc* del diccionario d es denotado por $d.\mathit{conoc}$; y este atributo *conoc* es un atributo de <u>todo</u> diccionario por haber sido declarado como atributo del modelo de representación utilizado para <u>todos</u> los diccionarios.

Nota:

> En este punto es importante mencionar que utilizamos la usual convención de considerar a los parámetros de entrada como constantes. De allí que en la postcondición arriba indicada para la operación *crear*, el valor (final) de m se refiera al mismo valor (inicial) de entrada de m.
>
> Continuaremos utilizando esta convención sobre parámetros de entrada a lo largo de todas estas notas.

(Fin de nota.)

Por último, hay un punto muy interesante e importante a discutir en relación con la postcondición especificada para la operación *crear* y el invariante de representación presentado en la subsección anterior. Recuerde que el invariante de representación es una exigencia que se le impone al modelo de representación, y es por tanto una exigencia impuesta a <u>todo</u> diccionario. De acuerdo con esto, los atributos del diccionario d siempre deben cumplir con el invariante de representación. Esto es, d siempre debe cumplir con lo siguiente:

$$d.\mathit{MAX} > 0 \ \wedge\ \#\, d.\mathit{conoc} \leqslant d.\mathit{MAX} \ \wedge\ d.\mathit{conoc} = \mathrm{dom}\, d.\mathit{tabla} \quad .$$

Por lo tanto, esta última condición es parte <u>*implícita*</u> de la postcondición de la operación *crear*.

Como consecuencia de esta conjunción *implícita* de la postcondición especificada explícitamente con el invariante de representación aplicado a d, la postcondición dada contiene redundancia: la fórmula central implica a la fórmula derecha en presencia del invariante y, viceversa, la derecha implica a la central. Tenemos entonces que, en presencia del invariante aplicado a d, la postcondición dada arriba es equivalente a

$$\{\ \textbf{Post}:\ d.\mathit{MAX} = m \ \wedge\ d.\mathit{conoc} = \emptyset\ \} \quad ,$$

debido a que $\mathrm{dom}\, f = \emptyset$ implica $f = \emptyset$ para cualquier función f, y también es equivalente a

$$\{\ \textbf{Post}:\ d.\mathit{MAX} = m \ \wedge\ d.\mathit{tabla} = \emptyset\ \} \quad ,$$

debido a que $\mathrm{dom}\, \emptyset = \emptyset$.

Estas dos últimas postcondiciones pueden ser consideradas mejores que la primera en vista de que son más breves, pero la primera tiene la ventaja de ser más clara y explícita. En estas notas preferiremos la claridad por encima de la longitud de una especificación, por lo cual nos quedamos con la primera postcondición explícita dada originalmente.

Sin embargo, más adelante veremos la utilidad de contar con varias postcondiciones equivalentes, haciendo uso de la más explícita para especificar mientras se hace uso de la más breves para otras labores.

Agregar Según se indicó en la introducción de esta subsección, esta operación es para agregar a un diccionario dado un par clave/valor. El parámetro de tipo diccionario deberá ser de entrada/salida, ya que debe ser modificado, mientras los otros dos parámetros, del par clave/valor a agregar, serán sólo de entrada:

proc *agregar* (**in-out** d : *Diccionario* ; **in** c : *T0* ; **in** v : *T1*)
{ **Pre** : ... }
{ **Post** : ... } .

De acuerdo con la descripción en lenguaje natural de esta operación, "agregar el par (c, v) al diccionario d", el comportamiento de ésta puede parecer obvio. Pero un momento de reflexión plantea la siguiente duda: ¿debe la clave c ser totalmente nueva, o puede ésta ya existir en el diccionario, en cuyo caso se reemplaza su valor asociado anterior por el nuevo valor v?

Éste es el valor agregado que resulta de combinar una descripción inicial en lenguaje natural con una posterior especificación formal detallada con pre/post-condición. Luego de que la primera descripción informal nos permite obtener una inicial comprensión básica de la operación, la segunda especificación formal "pule los detalles opacos" que puedan haber quedado de la primera descripción informal debido a la ambigüedad y tendencia a la incompletitud propias del uso de lenguaje natural.

Optaremos por la primera interpretación en la pregunta planteada arriba, exigiendo que la clave c sea totalmente nueva, y dejaremos la formalización de la segunda interpretación como ejercicio (ver ejercicio **2.4-b**). La especificación completa de nuestra operación es entonces:

proc *agregar* (**in-out** d : *Diccionario* ; **in** c : *T0* ; **in** v : *T1*)
{ **Pre** : $c \notin d.conoc \ \wedge \ \#d.conoc < d.MAX$ }
{ **Post** : $d.conoc = d_0.conoc \cup \{ c \} \ \wedge \ d.tabla = d_0.tabla \cup \{ (c, v) \}$ } .

Cuando tengamos un parámetro de entrada/salida, como es el caso de d en esta operación, en la postcondición podremos referirnos a su valor inicial indexando con 0 a su nombre. Por tanto, en la postcondición recién planteada, d_0 se refiere al valor del diccionario d en el estado inicial, mientras d se refiere al valor de d en el estado final.

La precondición exige que la clave c sea nueva en el diccionario y que aún quede capacidad en el diccionario para almacenar un nuevo par. La postcondición señala que el diccionario resultante conoce la nueva clave c además de continuar conociendo todas las claves anteriores, y que la nueva clave c está asociada al valor v mientras las claves conocidas previamente siguen asociadas a los mismos valores que al principio de la operación.

Vale ahora el mismo comentario sobre el invariante de representación que hicimos antes en relación con la especificación de la operación *crear*, pero ahora con un par de variaciones. En la especificación de *crear*, dijimos que el invariante de representación sobre el diccionario d estaba presente implícitamente en la postcondición. Ahora bien, en el caso de la operación *agregar*, el invariante de representación sobre el parámetro d está presente *tanto en la precondición como en la postcondición*. La diferencia estriba en el hecho de que en la operación *crear* el diccionario d era un parámetro sólo de salida, mientras ahora en la operación *agregar* es un parámetro de entrada/salida. Por último, el hecho de que se exija el invariante de representación a d en la precondición también garantiza que en la postcondición d_0 cumpla con el invariante de representación.

De acuerdo con esto, la precondición especificada para *agregar* es equivalente a

{ **Pre** : $c \notin \mathrm{dom}\, d.tabla \ \wedge \ \#d.tabla < d.MAX$ } ,

mientras la postcondición especificada para *agregar* puede ser abreviada sin pérdida de información, esto es, siendo equivalente, como

$$\{\ \textbf{Post}:\ d.tabla = d_0.tabla \cup \{\,(c, v)\,\}\ \} \qquad .$$

Tal como con la operación *crear*, nos quedaremos con el par pre/post-condición inicialmente planteado, por considerarlo la especificación más clara y completa.

Por otra parte, recuerde que el atributo MAX del modelo de representación fue declarado como atributo constante. Esto quiere decir que, una vez creado un diccionario e inicializado su atributo MAX, este valor ya no puede cambiar. Por lo tanto, la postcondición de *agregar* (y de toda operación que tenga un parámetro de entrada/salida d de tipo diccionario) también incluye implícitamente a la condición

$$d.MAX = d_0.MAX \qquad .$$

Eliminar Esta operación es para eliminar de un diccionario dado a un par clave/valor. Sin embargo, debido a la naturaleza de los diccionarios, bastará indicar la clave que se desea eliminar, pues el valor será implícitamente el asociado a tal clave en el diccionario. Al igual que en la operación *agregar*, el parámetro de tipo diccionario deberá ser de entrada/salida, y el segundo parámetro, la clave a eliminar, será sólo de entrada:

proc *eliminar* (**in-out** d : *Diccionario* ; **in** c : *T0*)
$\{\ \textbf{Pre}:\ \ldots\ \}$
$\{\ \textbf{Post}:\ \ldots\ \}$.

De nuevo, nos planteamos una duda de interpretación: ¿debe la clave c ser conocida por el diccionario, o puede ser ésta tanto conocida como desconocida por el diccionario?

Lo consistente con la interpretación escogida en el caso de la operación *agregar*, esto es, para agregar una clave, ésta debe ser nueva, será de nuevo la primera interpretación de la interrogante planteada: para eliminar una clave, ésta debe ser conocida. La formalización de la segunda interpretación queda de nuevo como ejercicio (ver ejercicio **2.4-b**). Nuestra especificación de *eliminar* es entonces:

proc *eliminar* (**in-out** d : *Diccionario* ; **in** c : *T0*)
$\{\ \textbf{Pre}:\ \ c \in d.conoc\ \}$
$\{\ \textbf{Post}:\ d.conoc = d_0.conoc - \{\,c\,\} \ \wedge\ d.tabla = d_0.tabla - \{\,(c, d_0.tabla\ c)\,\}\ \}$.

La precondición exige que la clave c sea conocida por el diccionario, mientras la postcondición señala que el diccionario resultante deja de conocer la clave c pero continúa conociendo el resto de las claves que conocía previamente, y el par que asociaba a la clave c con su valor, obtenido por la aplicación de la función inicial $d_0.tabla$ a la clave c, esto es, $d_0.tabla\ c$, es eliminado de la tabla de pares registrados, manteniéndose todo el resto presente.

Nuevamente, la presencia implícita del invariante de representación sobre d tanto en la precondición como en la postcondición (d es parámetro de entrada/salida) hace que la precondición especificada sea equivalente a

$$\{\ \textbf{Pre}:\ \ c \in \operatorname{dom} d.tabla\ \}$$

y que la postcondición dada pueda ser abreviada equivalentemente como

$$\{\ \textbf{Post}:\ d.tabla = d_0.tabla - \{\,(c, d_0.tabla\ c)\,\}\ \} \qquad .$$

Como hemos comentado anteriormente, las condiciones abreviadas serán convenientes de utilizar más adelante, pero a efectos de la especificación preferimos las más completas y claras.

Por otra parte, gracias a propiedades de operadores presentes en la teoría de conjuntos, esta última postcondición también puede ser reescrita como

$$\{\ \textbf{Post}:\ d.tabla = d_0.tabla - (\{\, c \,\} \times T1)\ \} \qquad ,$$

o como

$$\{\ \textbf{Post}:\ d.tabla = \{\, c \,\} \lhd d_0.tabla\ \} \qquad ,$$

siendo $\lhd$, según la notación de [Mor94, capítulo 9], el operador de co-restricción de dominio, esto es, restricción de dominio de una función (o relación) al complemento de un conjunto dado (ver [Mor94, página 88]).

Buscar y determinar existencia Las últimas dos operaciones, según la lista dada en la introducción a esta subsección, corresponden a, dada una clave, buscar su valor asociado en un diccionario dado y determinar si ésta es conocida o no por un diccionario dado. Las llamaremos *buscar* y *existe*.

Para mantener consistencia con la interpretación dada a la clave de entrada en *agregar* y *eliminar*, en la operación *buscar* exigiremos en la precondición que la clave de entrada sea conocida por el diccionario. En la operación *existe*, por la naturaleza misma de la operación, la clave podrá ser cualquiera.

En estas dos operaciones, el parámetro d de tipo diccionario será de entrada, ya que no hace falta modificarlo. Por lo tanto, el invariante de representación sobre d será supuesto implícitamente sólo en la precondición. Sin embargo, recuerde que acostumbramos exigir que un parámetro de entrada permanezca constante en una subrutina y, por lo tanto, d tendrá el mismo valor inicial al final de la operación, por lo que las referencias a d en la postcondición se refieren al mismo valor inicial de d.

La especificación completa de las operaciones se encuentra en la figura **2.3.(a)**.

2.3. Armando el aparato completo...

Ya hemos construido por completo una especificación para nuestro TAD *Diccionario*. Los componentes de la especificación corresponden a lo descrito en cada una de las subsecciones anteriores: modelo (abstracto) de representación, invariante de representación, y operaciones.

La especificación completa del TAD se encuentra ensamblada en la figura **2.3.(a)**. Allí Ud. verá que a la especificación se le dió el nombre $\mathbb{A}$. Más adelante se explicará la razón de este nombre (cuando debamos construir una re-especificación del mismo TAD, re-especificación a la que llamaremos $\mathbb{B}$).

2.4. Ejercicios

2.4-a. *...demostrar equivalencia entre las postcondiciones y precondiciones de las que se dijo tal cosa (y en esos puntos poner referencia a este ejercicio)...*

2.4-b. *...re-especificación de las operaciones relajando las precondiciones...*

Especificación $\mathbb{A}$ de TAD *Diccionario* (*T0*, *T1*)

Modelo de Representación

const MAX : int
var *conoc* : set *T0*
tabla : $T0 \nrightarrow T1$

Invariante de Representación

$MAX > 0 \quad \wedge \quad \#\, conoc \leqslant MAX \quad \wedge \quad conoc = \mathrm{dom}\, tabla$

Operaciones

proc *crear* (**in** m : int ; **out** d : *Diccionario*)
{ **Pre** : $m > 0$ }
{ **Post** : $d.MAX = m \quad \wedge \quad d.conoc = \emptyset \quad \wedge \quad d.tabla = \emptyset$ }

proc *agregar* (**in-out** d : *Diccionario* ; **in** c : *T0* ; **in** v : *T1*)
{ **Pre** : $c \notin d.conoc \quad \wedge \quad \#\, d.conoc < d.MAX$ }
{ **Post** : $d.conoc = d_0.conoc \cup \{\, c \,\} \quad \wedge \quad d.tabla = d_0.tabla \cup \{\, (c, v) \,\}$ }

proc *eliminar* (**in-out** d : *Diccionario* ; **in** c : *T0*)
{ **Pre** : $c \in d.conoc$ }
{ **Post** : $d.conoc = d_0.conoc - \{\, c \,\} \quad \wedge \quad d.tabla = d_0.tabla - \{\, (c, d_0.tabla\ c) \,\}$ }

proc *buscar* (**in** d : *Diccionario* ; **in** c : *T0* ; **out** v : *T1*)
{ **Pre** : $c \in d.conoc$ }
{ **Post** : $v = d.tabla\ c$ }

proc *existe* (**in** d : *Diccionario* ; **in** c : *T0* ; **out** e : boolean)
{ **Pre** : true }
{ **Post** : $e \equiv (c \in d.conoc)$ }

Fin TAD

Figura **2.3.(a)**: Especificación con modelo abstracto de un TAD *Diccionario*

2.4-c. *. . . re-especificar el TAD Diccionario sin considerar capacidad máxima. . .*

2.4-d. *. . . buscar especificaciones de TADs basadas en modelos en el texto de Liskov [Lis01], y compararlas con las nuestras. . .*

2.4-e. *. . . especificar un TAD MultiDiccionario, en el que cada clave pueda tener asociados múltiples valores. . .*

3. Otros ejemplos de especificación de TADs (clásicos)

En esta sección veremos algunos otros ejemplos de especificación de TADs. Todos los TADs que consideramos en esta sección son "clásicos", en el sentido reseñado en la introducción de estas notas de ser TADs muy usados en las ciencias e ingeniería de la computación. En la futura sección **8** presentaremos otros TADs que no son clásicos, mas son ejemplos interesantes.

3.0. Un TAD *Cola*

Entre los TADs clásicos se encuentran, además de nuestro ya conocido TAD *Diccionario*, los TADs *Cola* y *Pila*. Estos dos TADs se parecen un poco el uno al otro, mas no son exactamente iguales. En esta sección presentaremos un TAD *Cola* y dejaremos como ejercicio lo relacionado con *Pila*.

Una *cola* es, tal como en la vida real dentro de un banco o en alguna institución pública de servicio, una estructura a la que se anexan nuevos elementos por un extremo, "el final", y de la que se extraen elementos por el otro extremo, "el principio". En inglés, esta estructura es conocida como una secuencia FIFO, por *first-in-first-out*, ya que es una secuencia en la que el primer elemento que entra será el primero en salir.

Un TAD *Cola* usualmente cuenta con las siguientes operaciones: (i) agregar un elemento a la cola; (ii) extraer un elemento de la cola; (iii) ver el primer elemento de la cola; y (iv) determinar si una cola tiene o no elementos. Y, al igual que en el caso del TAD *Diccionario*, se requiere una operación de inicialización o creación de un nueva cola, usualmente vacía.

En la especificación que presentaremos del TAD *Cola*, utilizaremos en el modelo abstracto a una secuencia para representar a la cola (ver secuencias como estructuras matemátícas en [Mor94, sección 9.3]). Además, al igual que en el caso de *Diccionario*, limitaremos cada cola a una cierta capacidad máxima, que también será un atributo del modelo abstracto de representación.

La figura **3.0.(b)** muestra una especificación completa de un TAD *Cola*. Al igual que en el caso de *Diccionario*, nuestro TAD *Cola* es genérico, esto es, parametrizado por el tipo T de los elementos a ser almacenados en la cola.

Como maneras interesantes de re-frasear algunas de las pre/post-condiciones de nuestra especificación del TAD *Cola*, de acuerdo con las definiciones de los operadores sobre secuencias presentados en [Mor94, sección 9.3], la precondición de *desencolar* y *primero* es equivalente a

$$\#\, c.contenido > 0 \quad ,$$

mientras la postcondición de *desencolar* es equivalente a

$$(\, \exists\, e \,::\, c_0.contenido = \langle e \rangle \mathbin{+\!\!+} c.contenido \,)$$

y la postcondición de *primero* es equivalente a

$$(\, \exists\, s \,::\, c.contenido = \langle x \rangle \mathbin{+\!\!+} s \,) \quad .$$

Especificación $\mathbb{A}$ de TAD *Cola* (T)

Modelo de Representación

const MAX : int
var *contenido* : seq T

Invariante de Representación

$MAX > 0 \ \wedge \ \#\, contenido \leqslant MAX$

Operaciones

proc *crear* (**in** m : int ; **out** c : *Cola*)
{ **Pre** : $m > 0$ }
{ **Post** : $c.MAX = m \ \wedge \ c.contenido = \langle\rangle$ }

proc *encolar* (**in-out** c : *Cola* ; **in** $x : T$)
{ **Pre** : $\#\, c.contenido < c.MAX$ }
{ **Post** : $c.contenido = c_0.contenido +\!\!+ \langle x \rangle$ }

proc *desencolar* (**in-out** c : *Cola*)
{ **Pre** : $c.contenido \neq \langle\rangle$ }
{ **Post** : $c.contenido = \mathrm{tl}\ c_0.contenido$ }

proc *primero* (**in** c : *Cola* ; **out** $x : T$)
{ **Pre** : $c.contenido \neq \langle\rangle$ }
{ **Post** : $x = \mathrm{hd}\ c.contenido$ }

proc *vacia* (**in** c : *Cola* ; **out** v : boolean)
{ **Pre** : true }
{ **Post** : $v \equiv (c.contenido = \langle\rangle)$ }

Fin TAD

Figura **3.0.(b)**: Especificación con modelo abstracto de un TAD *Cola*

3.1. Un TAD *Conjunto*

Otro par de TADs clásicos son *Conjunto* y *Multiconjunto*. Al igual que en la sección anterior, presentaremos sólo un TAD *Conjunto* y dejaremos como ejercicio el trabajar con un TAD *Multiconjunto*.

Definiremos un TAD *Conjunto* en el que las primeras operaciones trabajan los elementos uno a uno, que serían las siguientes: (i) agregar un elemento a un conjunto, permitiendo que la operación sea siempre aplicable, indepedientemente de si el elemento estaba previamente en el conjunto o no; (ii) eliminar un elemento de un conjunto, de nuevo permitiendo que la operación sea siempre aplicable, esté o no el elemento previamente en el conjunto; (iii) extraer un elemento cualquiera de un conjunto no-vacío; (iv) determinar si un elemento está o no en un conjunto; y (v) determinar si un conjunto está vacío o no. Además de estas primeras operaciones, que trabajan sobre un solo conjunto y manejan elementos uno por uno, tendremos un segundo grupo de operaciones correspondientes a típicas operaciones binarias de la teoría de conjuntos: (vi) unir dos conjuntos devolviendo el resultado en un tercero; (vii) ídem para intersectar; y (viii) ídem para calcular diferencia de conjuntos. Por último, como de costumbre, tendremos una operación de inicialización de un conjunto en vacío.

En la especificación de nuestro TAD *Conjunto*, utilizaremos en el modelo abstracto a un conjunto, lo cual requiere una breve aclaratoria de sutilezas que será dada en el siguiente párrafo. Y, como en nuestras especificaciones anteriores, el modelo abstracto de representación también indicará una cierta capacidad máxima de almacenamiento.

Ahora bien, en cuanto a la dualidad entre "TAD *Conjunto*" vs. "conjunto como un atributo del modelo de representación", debemos aclarar la diferencia (y similitudes). Un conjunto como modelo abstracto de representación se refiere a la estructura matemática conjunto de la teoría de conjuntos (tal como en todos nuestros otros modelos abstractos de representación), mientras un conjunto como TAD es un tipo de datos que aspiramos a implementar en un computador utilizando algún lenguaje de programación. He allí la diferencia entre "conjunto" como TAD y como modelo abstracto. En consecuencia, los conjuntos como modelo abstracto pueden ser utilizados con todas las posibles operaciones que provee la teoría de conjuntos para ello, mientras los conjuntos de nuestro TAD sólo pueden ser utilizados con las operaciones que se definan para el TAD. El operador de producto cartesiano sobre conjuntos, el operador de conjunto potencia, y el operador de complementación serían ejemplos de operadores disponibles en la teoría de conjuntos y, por lo tanto, operadores disponibles para ser utilizados sobre nuestro modelo abstracto de representación, pero que no estarían a disposición de los usuarios de nuestro TAD, simplemente porque quien diseñó el TAD decidió no incluir tales operaciones. Igualmente, si en nuestro TAD hubiésemos decidido quedarnos sólo con el primer grupo de operaciones (i)-(v), sin el segundo grupo (vi)-(viii), entonces los operadores de unión, intersección y diferencia no habrían estado disponibles para los usuarios de nuestro TAD mientras sí habrían sido operadores disponibles para nuestro modelo abstracto de representación (de hecho, note más adelante que estos operadores son utilizados sobre el modelo abstracto para especificar las operaciones (i) y (ii) del primer grupo de operaciones de nuestro TAD).

Aclarada la sutil dualidad entre conjunto como TAD y conjunto como modelo abstracto de representación, mostramos en la figura **3.1.(c)** una especificación completa de nuestro TAD *Conjunto*. Al igual que todos los TADs presentados hasta ahora, nuestro TAD *Conjunto* es genérico, parametrizado por el tipo T de los elementos a ser almacenados en un conjunto.

Note que las precondiciones de *unir*, *intersectar* y *restar* son más fuertes de lo estrictamente

---

Especificación $\mathbb{A}$ de TAD *Conjunto* (T)

Modelo de Representación

const MAX : int
var $contenido$: set T

Invariante de Representación

$MAX > 0 \ \wedge \ \#\, contenido \leqslant MAX$

Operaciones

proc *crear* (**in** m : int ; **out** c : *Conjunto*)
{ **Pre** : $m > 0$ }
{ **Post** : $c.MAX = m \ \wedge \ c.contenido = \emptyset$ }

proc *agregar* (**in-out** c : *Conjunto* ; **in** $x : T$)
{ **Pre** : $x \notin c.contenido \ \Rightarrow \ \#\, c.contenido < c.MAX$ }
{ **Post** : $c.contenido = c_0.contenido \cup \{ x \}$ }

proc *eliminar* (**in-out** c : *Conjunto* ; **in** $x : T$)
{ **Pre** : true }
{ **Post** : $c.contenido = c_0.contenido - \{ x \}$ }

proc *extraer* (**in** c : *Conjunto* ; **out** $x : T$)
{ **Pre** : $c.contenido \neq \emptyset$ }
{ **Post** : $x \in c.contenido$ }

proc *pertenece* (**in** c : *Conjunto* ; **in** $x : T$; **out** p : boolean)
{ **Pre** : true }
{ **Post** : $p \equiv (x \in c.contenido)$ }

proc *vacio* (**in** c : *Conjunto* ; **out** v : boolean)
{ **Pre** : true }
{ **Post** : $v \equiv (c.contenido = \emptyset)$ }

proc *unir* (**in** $c0, c1$: *Conjunto* ; **in** m : int ; **out** c : *Conjunto*)
{ **Pre** : $\#\, c0.contenido + \#\, c1.contenido \leqslant m$ }
{ **Post** : $c.MAX = m \ \wedge \ c.contenido = c0.contenido \cup c1.contenido$ }

proc *intersectar* (**in** $c0, c1$: *Conjunto* ; **in** m : int ; **out** c : *Conjunto*)
{ **Pre** : $\#\, c0.contenido \leqslant m \ \wedge \ \#\, c1.contenido \leqslant m$ }
{ **Post** : $c.MAX = m \ \wedge \ c.contenido = c0.contenido \cap c1.contenido$ }

proc *restar* (**in** $c0, c1$: *Conjunto* ; **in** m : int ; **out** c : *Conjunto*)
{ **Pre** : $\#\, c0.contenido \leqslant m$ }
{ **Post** : $c.MAX = m \ \wedge \ c.contenido = c0.contenido - c1.contenido$ }

Fin TAD

Figura **3.1.(c)**: Especificación con modelo abstracto de un TAD *Conjunto*

necesario. Las precondiciones más débiles requeridas en cada caso serían

$$\#\,(c0.contenido \cup c1.contenido) \leqslant m$$

para *unir*,

$$\#\,(c0.contenido \cap c1.contenido) \leqslant m$$

para *intersectar*, y

$$\#\,(c0.contenido - c1.contenido) \leqslant m$$

para *restar*. Sin embargo, estas precondiciones serían más difíciles de garantizar por los clientes de nuestro TAD, previa llamada a las operaciones. La dificultad radicaría en que poder garantizar tales precondiciones requeriría conocer previamente el resultado que se desea obtener con la llamada, esto es, el resultado de la unión, el resultado de la intersección o el resultado de la diferencia. Las precondiciones que escogimos para nuestra especificación requieren sólo verificaciones simples en términos de los tamaños de los operandos de entrada (a diferencia de las más débiles que requerirían verificaciones en términos de los tamaños de ¡los resultados de salida!).

3.2. Ejercicios

3.2-a. *. . . especificar un TAD Pila. . .*

3.2-b. *. . . especificar un TAD ColaConPrioridades. . .*

3.2-c. *. . . especificar un TAD MultiConjunto. . .*

4. Implementación de un TAD

Las dos últimas secciones nos mostraron cómo especificar un nuevo TAD. Para esto se utilizó en cada caso un modelo *abstracto* de representación. Nótese que lo más importante en un TAD son sus operaciones con sus respectivas descripciones de comportamiento, mientras que el modelo abstracto en sí no tiene mayor relevancia. El único papel que jugó el modelo abstracto de representación en todos los ejemplos fue el permitir describir el comportamiento de las operaciones (notar que sin el modelo no habría habido manera de especificar los pares pre/post-condición de las operaciones).

Ahora, para implementar un TAD necesitamos ante todo escoger una estructura de datos con la cual poder almacenar la información correspondiente al TAD. Para construir tal estructura de datos, necesitaremos los posibles constructores de estructuras provistos por algún lenguaje de programación. En estas notas utilizaremos únicamente arreglos y registros, limitándonos a arreglos en los primeros ejemplos. (Algunos ejemplos muy interesantes de implementación de TADs incluyen el uso de estructuras recursivas de datos construidas con apuntadores/referencias, como listas enlazadas y árboles de diversos tipos; consideraremos a tales ejemplos fuera del alcance de estas notas.)

La estructura de datos que se decida utilizar para almacenar la información del TAD corresponderá en consecuencia a un nuevo modelo de representación para el TAD, al cual llamaremos *modelo concreto* de representación. Con tal nuevo modelo, las operaciones del TAD deben ser re-especificadas, y tal re-especificación debe ser consistente con la especificación original del TAD. La consistencia de la re-especificación del TAD con modelo concreto en relación con la especificación original del TAD con modelo abstracto será garantizada a través de una *relación de acoplamiento* entre los dos modelos (también llamada *invariante de acoplamiento* o *relación de abstracción*).

4.0. Modelo *Concreto* de Representación

Este nuevo modelo, como ya fue indicado, corresponde a la estructura de datos sobre la que se decida basar la construcción de la implementación del TAD. En el caso de nuestro TAD *Diccionario*, si queremos limitarnos a utilizar arreglos, podemos almacenar los pares del diccionario mediante dos arreglos: un arreglo de claves y un arreglo de valores, de forma tal que los pares se asocien componente a componente; esto es, la clave de la posición 0 se aparea con el valor de la posición 0, la clave de la posición 1 se aparea con el valor de la posición 1, y así sucesivamente.

Empezamos entonces la construcción de nuestro modelo concreto de representación de la siguiente forma:

Modelo de Representación
var *claves* : array [. . .) of *T0*
valores : array [. . .) of *T1* .

Note que el tamaño conveniente para estos arreglos debe ser la capacidad máxima indicada por la especificación original. Por ello, tomamos el atributo MAX del modelo abstracto y lo re-usamos en el modelo concreto, de nuevo como atributo constante en lugar de como atributo variable:

Modelo de Representación
const MAX : int
var *claves* : array [0..MAX) of *T0*
valores : array [0..MAX) of *T1* .

Por último, el tamaño conveniente para los arreglos puede ser MAX. Sin embargo, no siempre un diccionario almacenará la máxima cantidad posible de pares. Un diccionario almacenará a lo sumo tal cantidad máxima, pero usualmente tendrá menos. Necesitamos entonces un atributo más que indique cuál es la cantidad actual real de pares almacenados en el diccionario:

Modelo de Representación
const MAX : int
var *claves* : array [0..MAX) of *T0*
valores : array [0..MAX) of *T1*
tam : int .

Este último atributo *tam* lo manejaremos en combinación con los arreglos de forma tal que los pares del diccionario estén almacenados en las primeras *tam* posiciones de los arreglos. Esto es, en ambos arreglos el segmento con información relevante será [0..*tam*) , mientras lo que haya en el segmento [*tam*..MAX) de ambos arreglos será ignorado.

4.1. Invariante de Representación

Todo modelo de representación va acompañado de un invariante de representación (aunque sea el invariante trivial "true"). Por lo tanto, así como nuestro modelo abstracto de representación tenía su invariante, nuestro nuevo modelo concreto de representación también contará con su correspondiente invariante.

Al igual que antes, debemos analizar qué restricciones adicionales deben ser satisfechas por los atributos de nuestro nuevo modelo. Veamos: MAX, de nuevo, deberá ser positivo; *tam* deberá estar en el rango [0..MAX] , ambos extremos incluidos como las cantidades mínima y máxima posibles de pares a ser almacenadas en el diccionario; y, por último, no puede haber claves

repetidas en el segmento relevante, esto es, el segmento $[\,0..tam\,)$, del arreglo *claves*. Nuestro invariante de representación es entonces:

Invariante de Representación

$$MAX > 0 \;\wedge\; 0 \leqslant tam \leqslant MAX$$
$$\wedge\; (\forall\, i, j \,:\, 0 \leqslant i, j < tam \,:\, i \neq j \;\Rightarrow\; claves[i] \neq claves[j]\,) \quad .$$

4.2. Relación de Acoplamiento

He aquí un nuevo componente que ha de estar presente en la re-especificación del TAD, la cual estamos construyendo con miras a implementar el TAD. La especificación original del TAD con modelo abstracto es la especificación propiamente dicha, o contrato inicial de comportamiento, del TAD. La re-especificación del TAD con modelo concreto es una especie de puente intermedio entre el contrato inicial y la implementación final. En consecuencia, la re-especificación con modelo concreto debe ser consistente con la especificación original con modelo abstracto. Para poder analizar tal consistencia, se requiere indicar cómo el nuevo modelo representa al modelo anterior. Esto se logra por medio de una relación de acoplamiento (también llamada *invariante de acoplamiento* o *relación de abstracción*).

En la subsección anterior, al presentar el modelo concreto también dimos una breve descripción en lenguaje natural de cómo este nuevo modelo representaría a un diccionario. Tal descripción informal es lo que debemos usar para formalizar detalladamente cómo el nuevo modelo concreto representa a los componentes del modelo abstracto anterior.

Veamos, las claves conocidas por el diccionario estarán almacenadas en el arreglo *claves* y, como señalamos en la subsección anterior, el segmento relevante de este arreglo será $[\,0..tam\,)$. Por lo tanto, el conjunto *conoc* se construye formalmente a partir de nuestro modelo concreto de acuerdo con la siguiente fórmula:

$$conoc = \{\, i \,:\, 0 \leqslant i < tam \,:\, claves[i]\,\} \quad .$$

Igualmente, los pares del diccionario se encuentran en el mismo segmento de ambos arreglos, apareando los arreglos componente a componente:

$$tabla = \{\, i \,:\, 0 \leqslant i < tam \,:\, (claves[i], valores[i])\,\} \quad .$$

Por tanto, nuestra relación de acoplamiento completa es la siguiente:

Relación de Acoplamiento

$$conoc = \{\, i \,:\, 0 \leqslant i < tam \,:\, claves[i]\,\}$$
$$\wedge$$
$$tabla \;=\; \{\, i \,:\, 0 \leqslant i < tam \,:\, (claves[i], valores[i])\,\} \quad .$$

En cuanto al atributo MAX, note que éste está presente tanto en el modelo abstracto como en el modelo concreto. Supondremos siempre que cuando un atributo del modelo abstracto sea repetido en el modelo concreto, éste se representará a sí mismo; esto es, el atributo abstracto MAX es igual al atributo concreto MAX.

Debemos acá acotar que la notación que hemos utilizado recién para conjuntos definidos por comprensión difiere ligeramente de la notación de [Mor94]. En nuestra notación de tres partes hemos utilizado ":" para ambos separadores, mientras el texto en cuestión separa con "|" y "·", además de hacer siempre explícito al tipo o conjunto del que toma valores la variable generadora: allí el conjunto asociado a *conoc* antes especificado habría sido denotado como

$\{\, i : \mathbb{N} \mid 0 \leqslant i < tam \cdot claves[i] \,\}$. Nuestra desviación se debe a razones de consistencia con nuestra notación para cuantificaciones, que también usa doble separación con ":" y deja implícito el tipo de la variable generadora, desviándose exactamente en la misma forma de la notación para cuantificaciones de [Mor94]. Más detalles sobre la notación para conjuntos definidos por comprensión en nuestro texto de referencia, que además permite la omisión de la segunda o tercera parte mientras nosotros evitaremos hacer tal cosa, pueden ser consultados en [Mor94, sección 9.1.3].

4.3. Operaciones

Por último, las operaciones del TAD deben ser re-especificadas en términos del nuevo modelo concreto, de manera tal que se mantenga correspondencia con la intención expuesta en los pares pre/post-condición originales. Una vez hecho esto, sólo faltaría implementar los cuerpos de las subrutinas correspondientes a las operaciones del tipo, lo cual es un ejercicio algorítmico que no es de nuestro interés central en estas notas.

No es imprescindible que esta re-especificación concreta de las operaciones sea hecha explícitamente. Esto se debe a que la relación de acoplamiento induce implícitamente una cierta re-especificación para cada operación que automáticamente mantiene la correspondencia adecuada con la especificación original. Más adelante, cuando estudiemos las técnicas de refinamiento de datos que utilizaremos para analizar la consistencia entre especificación e implementación, en la sección **6**, veremos cuál es tal re-especificación concreta implícita y por qué es autómaticamente adecuada.

Sin embargo, a pesar de la existencia de tal re-especificación implícita automáticamente correcta para las operaciones, también es posible que hagamos nuestra propia re-especificación explícita. En este caso, para cada operación que re-especifiquemos explícitamente, debemos luego mostrar que hemos mantenido la intención de su especificación original. La posible conveniencia de optar por una re-especificación explícita estriba en el que ésta pueda entonces sugerir implementaciones particulares para una operación, en cuanto a la construcción del cuerpo de la subrutina correspondiente.

En estas notas, optaremos siempre por re-especificar explícitamente todas las operaciones, principalmente porque consideramos que esto ayuda a una mejor comprensión de las operaciones en el marco del modelo concreto. En algunos casos comentaremos si una re-especificación explícita sugiere o no cierta implementación particular; así mismo, en algunos casos comentaremos si una re-especificación explícita corresponde exactamente a la re-especificación implícita inducida por la relación de acoplamiento.

Crear En cuanto a la primera operación de creación o inicialización, tenemos la misma interfaz que antes para la misma:

proc *crear* (**in** m : int ; **out** d : *Diccionario*)
{ **Pre** : ... }
{ **Post** : ... } .

De hecho, para todas las operaciones, la interfaz de parámetros entrada/salida siempre debe mantenerse igual.

En cuanto a la precondición, lo consistente es mantener la misma anterior: m debe ser positivo. En relación con la postcondición, las restricciones sobre MAX siempre se mantendrán iguales ya que este atributo se mantuvo igual en el cambio de modelo. Ahora, en cuanto a que el diccionario

resultante deba ser vacío, en el nuevo modelo concreto esto corresponde a exigir que el atributo *tam* sea cero. Veamos entonces la re-especificación completa de la primera operación:

$$\begin{array}{l}\textbf{proc}\ \mathit{crear}\ (\ \textbf{in}\ m : \mathrm{int}\ ;\ \textbf{out}\ d : \mathit{Diccionario}\ ) \\ \quad \{\ \textbf{Pre}: \quad m > 0\ \} \\ \quad \{\ \textbf{Post}: \ d.\mathit{MAX} = m \ \wedge \ d.\mathit{tam} = 0\ \} \qquad .\end{array}$$

Vale la pena recordar que, al igual que anteriormente, el invariante de representación (concreto) sobre el diccionario de salida d se encuentra implícitamente anexo a la postcondición.

Agregar Mantenemos la misma interfaz anterior:

$$\begin{array}{l}\textbf{proc}\ \mathit{agregar}\ (\ \textbf{in-out}\ d : \mathit{Diccionario}\ ;\ \textbf{in}\ c : \mathit{T0}\ ;\ \textbf{in}\ v : \mathit{T1}\ ) \\ \quad \{\ \textbf{Pre}: \quad \ldots\ \} \\ \quad \{\ \textbf{Post}: \ \ldots\ \} \qquad .\end{array}$$

En lo que a la precondición atañe, exigir que la clave c sea nueva se traduce en que ésta no debe estar en el segmento $[\,0..\mathit{tam}\,)$ del arreglo *claves*, y exigir que quede capacidad en el diccionario se re-expresa en términos del atributo *tam*. En cuanto a la postcondición, agregar el nuevo par al diccionario se expresa en términos de los contenidos de los dos arreglos del modelo de representación (concreto). Veamos la re-especificación:

$$\begin{array}{l}\textbf{proc}\ \mathit{agregar}\ (\ \textbf{in-out}\ d : \mathit{Diccionario}\ ;\ \textbf{in}\ c : \mathit{T0}\ ;\ \textbf{in}\ v : \mathit{T1}\ ) \\ \quad \{\ \textbf{Pre}: \quad \neg\,(\,\exists\, i \ :\ 0 \leqslant i < d.\mathit{tam} \ :\ d.\mathit{claves}[i] = c\,) \ \wedge \ d.\mathit{tam} < d.\mathit{MAX}\ \} \\ \quad \{\ \textbf{Post}: \ d.\mathit{tam} = d_0.\mathit{tam} + 1 \\ \qquad\qquad \wedge \ d.\mathit{claves} = d_0.\mathit{claves}\,(d_0.\mathit{tam} : c) \\ \qquad\qquad \wedge \ d.\mathit{valores} = d_0.\mathit{valores}\,(d_0.\mathit{tam} : v)\ \} \qquad .\end{array}$$

Note que en la postcondición estamos utilizando la notación $a(i : x)$ de construcción de un nuevo arreglo por reemplazo en a de la posición i con el valor x (ver definición en [Kal90, sección 10.1] o [Mez00, sección 6.2]). Esto es, $a(i : x)$ denota a un arreglo que coincide con a en todas las posiciones, excepto en i, posición en la que el nuevo arreglo contiene x. Formalmente, $a(i : x)[j] = a[j]$ para toda posición j diferente de i, y $a(i : x)[i] = x$. La postcondición arriba indicada para *agregar* expresa entonces que ambos arreglos $d.\mathit{claves}$ y $d.\mathit{valores}$ se mantienen intactos, salvo por la posición correspondiente a $d_0.\mathit{tam}$ (esto es, el valor inicial de $d.\mathit{tam}$), en la que se almacena el nuevo par (c, v).

Note que la postcondición dada es, en términos estrictamente lógicos, ligeramente más fuerte de lo necesario. Dado que el segmento $[\,\mathit{tam}..\mathit{MAX}\,)$ es irrelevante en ambos arreglos, su contenido no tiene por qué permanecer igual; si esta porción de los arreglos cambiase, igualmente cumpliríamos con lo deseado. Sería válido entonces sólo exigir la postcondición, ligeramente más débil, siguiente:

$$\begin{array}{l}\{\ \textbf{Post}: \ d.\mathit{tam} = d_0.\mathit{tam} + 1 \\ \qquad\qquad \wedge \ (\,\forall\, i \ :\ 0 \leqslant i < d_0.\mathit{tam} \ :\ d.\mathit{claves}[i] = d_0.\mathit{claves}[i] \\ \qquad\qquad\qquad\qquad\qquad\qquad \wedge \ d.\mathit{valores}[i] = d_0.\mathit{valores}[i]\,) \\ \qquad\qquad \wedge \ d.\mathit{claves}[d_0.\mathit{tam}] = c \ \wedge \ d.\mathit{valores}[d_0.\mathit{tam}] = v\ \} \qquad ,\end{array}$$

lo cual equivale a la postcondición anterior en el segmento relevante de los arreglos mientras se permite que el segmento irrelevante de los arreglos sea alterado arbitrariamente. En particular, esta nueva postcondición permitiría que tal "alteración arbitraria" del segmento irrelevante correspondiese a mantener los valores (irrelevantes) anteriores.

Como de costumbre, preferimos la postcondición que dimos inicialmente. En este caso, pues es más concisa y clara, y porque sabemos que la restricción innecesaria impuesta sobre los segmentos irrelevantes no involucrará costos innecesarios de ejecución al momento de implementar el cuerpo de la subrutina/operación.

Note también que la relación de acoplamiento entre nuestro modelo concreto y el modelo abstracto de la especificación original del TAD no impone ningún requerimiento de orden sobre los pares clave/valor en los arreglos. Por lo tanto, si estos pares fuesen reorganizados apropiadamente en los arreglos, el diccionario representado seguiría siendo el mismo. Por ejemplo, nuestra postcondición podría requerir la siguiente reorganización, correspondiente a insertar el nuevo par (c, v) en la posición 0 desplazando el resto de los elementos hacia la derecha, y el resultado aún sería el deseado:

$$\begin{array}{l}\{\ \textbf{Post}:\ d.tam = d_0.tam + 1 \\ \qquad\qquad \wedge\ \ d.claves[0] = c\ \ \wedge\ \ d.valores[0] = v \\ \qquad\qquad \wedge\ \ (\forall\, i\ :\ 1 \leqslant i < d.tam\ :\ d.claves[i] = d_0.claves[\,i-1\,] \\ \qquad\qquad\qquad\qquad\qquad\qquad \wedge\ \ d.valores[i] = d_0.valores[\,i-1\,])\ \ \}\ .\end{array}$$

Sin embargo, a pesar de que esta postcondición (concreta) es consistente con la postcondición (abstracta) original que se desea satisfacer, nuestros conocimientos de programación nos señalan que satisfacer esta postcondición será más costoso que satisfacer la postcondición (concreta) originalmente especificada (satisfacer esta nueva postcondición sería de complejidad lineal sobre el tamaño del diccionario, mientras que satisfacer la primera sería de complejidad constante).

En esta operación hemos visto un buen ejemplo de cómo una re-especificación explícita puede sugerir implementaciones particulares. Cada una de las postcondiciones antes propuestas sugiere una cierta implementación: colocar el nuevo par al final del segmento relevante de los arreglos, o colocar el nuevo par al inicio de tal segmento haciendo un corrimiento del resto de los pares.

Eliminar En este caso, la precondición puede reexpresarse en los nuevos términos sin mayor problema, de manera similar a como se hizo con la operación anterior *agregar*:

$$\begin{array}{l}\textbf{proc}\ \mathit{eliminar}\ (\ \textbf{in-out}\ d : \mathit{Diccionario}\ ;\ \textbf{in}\ c : \mathit{T0}\ ) \\ \quad \{\ \textbf{Pre}:\ \ (\exists\, i\ :\ 0 \leqslant i < d.tam\ :\ d.claves[i] = c\,)\ \ \} \\ \quad \{\ \textbf{Post}:\ \ \ldots\ \}\qquad .\end{array}$$

La postcondición es un poco más compleja. Al eliminar el par de ambos arreglos, no podemos dejar un "hueco" en medio del segmento relevante. Una posibilidad sería mover una posición hacia a la izquierda a todos los pares relevantes que estén a la derecha del par eliminado. Sin embargo, esto sería innecesariamente costoso, ya que no es necesario mantener a los elementos en el orden en el que estaban inicialmente en los arreglos. La manera menos costosa de "tapar el hueco" dejado por el par eliminado es trayendo al último par hacia el espacio dejado por el par eliminado. Re-especificamos *eliminar* entonces de esa forma:

$$\begin{array}{l}\textbf{proc}\ \mathit{eliminar}\ (\ \textbf{in-out}\ d : \mathit{Diccionario}\ ;\ \textbf{in}\ c : \mathit{T0}\ ) \\ \quad \{\ \textbf{Pre}:\ \ (\exists\, i\ :\ 0 \leqslant i < d.tam\ :\ d.claves[i] = c\,)\ \ \} \\ \quad \{\ \textbf{Post}:\ d.tam = d_0.tam - 1 \\ \qquad\qquad \wedge\ \ (\exists\, i\ :\ 0 \leqslant i < d_0.tam\ :\ d_0.claves[i] = c \\ \qquad\qquad\qquad\qquad \wedge\ \ d.claves = d_0.claves\,(i : d_0.claves[\,d_0.tam - 1\,]) \\ \qquad\qquad\qquad\qquad \wedge\ \ d.valores = d_0.valores\,(i : d_0.valores[\,d_0.tam - 1\,])\,)\ \ \}\ .\end{array}$$

Note que existen otras posibles reorganizaciones válidas para los arreglos, tal como señalamos anteriormente para la operación *agregar*. Sin embargo, la manera escogida parece ser la más sencilla de obtener el resultado deseado.

De nuevo, note que el haber utilizado para esta operación la alternativa de re-especificación explícita permitió proponer, en la postcondición, una manera particular de implementar la eliminación de un par.

Buscar y determinar existencia Ya hemos jugado lo suficiente con el nuevo modelo (concreto) como para que la especificación de las últimas dos operaciones no represente un problema. La especificación de éstas se encuentra en la figura **4.4.(d)**.

4.4. Armando el aparato completo... ahora como refinamiento

La re-especificación de nuestro TAD *Diccionario* con el modelo concreto ya está completa. Los componentes de esta re-especificación son casi los mismos de la especificación inicial anterior: modelo (ahora concreto) de representación, invariante de representación, y operaciones. Mas ahora hay un nuevo componente: la relación de acoplamiento. Esta relación enlaza el nuevo modelo (concreto) con el anterior modelo (abstracto), por lo cual esta re-especificación ha de establecerse desde un principio como dependiente de la anterior. He allí la necesidad de haber dado un nombre, $\mathbb{A}$, a la especificación anterior: para poder referirnos a ella posteriormente. Nuestra nueva especificación será llamada $\mathbb{B}$, y desde el encabezado de la especificación será declarada como una re-especificación. Diremos que la especificación $\mathbb{B}$ es un *refinamiento* de $\mathbb{A}$.

Formalmente, las técnicas que nos permitirán mostrar que nuestra re-especificación $\mathbb{B}$ con miras a implementar el TAD es consistente con la especificación original $\mathbb{A}$ son técnicas de *refinamiento de datos*. De allí que digamos que $\mathbb{B}$ es un refinamiento de $\mathbb{A}$ o, más específicamente , $\mathbb{B}$ es un *datos-refinamiento* de $\mathbb{A}$ (del término en inglés "*data-refinement*"), ya que lo que se ha modificado para acercarnos a una implementación es el espacio de datos, esto es, el modelo de representación.

La re-especificación completa del TAD *Diccionario* ha sido ensamblada en la figura **4.4.(d)**.

4.5. Finalizando la implementación

Una vez re-especificado el TAD con modelo concreto, lo único que resta hacer para completar la implementación es construir un cuerpo para cada subrutina/operación del TAD, de forma tal que el cuerpo sea correcto con respecto a la especificación pre/post-condición dada para cada operación. En estas notas no nos ocuparemos de la construcción de tales cuerpos de subrutinas y dejamos esta labor como ejercicio.

4.6. Ejercicios

4.6-a. *... ejercicios sobre reespecificación de las operaciones relajando las precondiciones...*

4.6-b. *... ejercicios sobre cambio de modelo concreto...*

4.6-c. *... ejercicio sobre TAD MultiDiccionario...*

Especificación $\mathbb{B}$ de TAD *Diccionario* (*T0*, *T1*) **, refinamiento de** $\mathbb{A}$

Modelo de Representación

const MAX : int
var *claves* : array [0..MAX) of *T0*
valores : array [0..MAX) of *T1*
tam : int

Invariante de Representación

$MAX > 0 \ \wedge \ 0 \leqslant tam \leqslant MAX$
$\wedge \ (\forall i, j \ : \ 0 \leqslant i, j < tam \ : \ i \neq j \ \Rightarrow \ claves[i] \neq claves[j] \,)$

Relación de Acoplamiento

$conoc = \{\, i \ : \ 0 \leqslant i < tam \ : \ claves[i] \,\}$
$\wedge$
$tabla \ = \ \{\, i \ : \ 0 \leqslant i < tam \ : \ (claves[i], valores[i]) \,\}$

Operaciones

proc *crear* (**in** m : int ; **out** d : *Diccionario*)
{ **Pre** : $m > 0$ }
{ **Post** : $d.MAX = m \ \wedge \ d.tam = 0$ }

proc *agregar* (**in-out** d : *Diccionario* ; **in** c : *T0* ; **in** v : *T1*)
{ **Pre** : $\neg\, (\, \exists\, i \ : \ 0 \leqslant i < d.tam \ : \ d.claves[i] = c \,) \ \wedge \ d.tam < d.MAX$ }
{ **Post** : $d.tam = d_0.tam + 1$
$\wedge \ d.claves = d_0.claves\, (d_0.tam : c)$
$\wedge \ d.valores = d_0.valores\, (d_0.tam : v)$ }

proc *eliminar* (**in-out** d : *Diccionario* ; **in** c : *T0*)
{ **Pre** : $(\, \exists\, i \ : \ 0 \leqslant i < d.tam \ : \ d.claves[i] = c \,)$ }
{ **Post** : $d.tam = d_0.tam - 1$
$\wedge \ (\, \exists\, i \ : \ 0 \leqslant i < d_0.tam \ : \ d_0.claves[i] = c$
$\wedge \ d.claves = d_0.claves\, (i : d_0.claves[\, d_0.tam - 1 \,])$
$\wedge \ d.valores = d_0.valores\, (i : d_0.valores[\, d_0.tam - 1 \,]) \,)$ }

proc *buscar* (**in** d : *Diccionario* ; **in** c : *T0* ; **out** v : *T1*)
{ **Pre** : $(\, \exists\, i \ : \ 0 \leqslant i < d.tam \ : \ d.claves[i] = c \,)$ }
{ **Post** : $(\, \exists\, i \ : \ 0 \leqslant i < d.tam \ : \ d.claves[i] = c \ \wedge \ d.valores[i] = v \,)$ }

proc *existe* (**in** d : *Diccionario* ; **in** c : *T0* ; **out** e : boolean)
{ **Pre** : true }
{ **Post** : $e \equiv (\, \exists\, i \ : \ 0 \leqslant i < d.tam \ : \ d.claves[i] = c \,)$ }

Fin TAD

Figura **4.4.(d)**: Re-especificación con modelo concreto del TAD *Diccionario*

5. Otros ejemplos de implementación de TADs (clásicos)

En esta sección veremos otros ejemplos de implementación de TADs, correspondientes a los ejemplos de especificación de la sección **3**. Así como los TADs entonces presentados fueron clásicos, las soluciones de implementación para ellos que acá presentaremos son clásicas en sí mismas.

5.0. Una implementación del TAD *Cola*

Como el TAD *Cola* especificado en la sección **3.0** corresponde a colas con capacidad acotada, es razonable utilizar un arreglo en el modelo concreto de representación para almacenar los elementos de una cola. Por supuesto, también es razonable utilizar como tamaño del arreglo a la capacidad máxima prescrita por la especificación (a pesar de que en el ejercicio **5.2-a** mencionaremos otra posibilidad de implementación en la que el tamaño del arreglo es otro). Nuestro modelo concreto empieza entonces así:

Modelo de Representación
const MAX : int
var *elems* : array $[0..MAX)$ of T .

Para manejar el contenido del arreglo, es natural pensar que, partiendo de una cola vacía, los elementos se empezarían a encolar en ella en las posiciones 0, 1, 2, y así sucesivamente, del arreglo. Esto sugiere agregar un índice entero al modelo concreto de representación que permita controlar qué segmento del arreglo contiene a la cola en cada momento, con lo que este índice siempre señalaría la posición en la cual se debe encolar al siguiente elemento. Llamemos *fin* a tal índice, puesto que éste señala el final de la cola, y tomemos entonces que el segmento $[0..fin)$ del arreglo correspondería a la cola. Nótese que la operación de encolar con este modelo sería entonces implementable con complejidad constante.

Hasta ahora, nuestro índice *fin* juega un papel similar al que jugó el índice *tam* en nuestra implementación del TAD *Diccionario*. Ahora bien, suponiendo que nos limitamos a utilizar un solo índice, como hicimos al implementar *Diccionario*, analicemos qué ocurrirá con la operación de desencolar. El elemento que debería ser eliminado de la cola en esta operación sería el ubicado en la posición 0 y entonces, si insistimos en que el segmento del arreglo que representa la cola sea $[0..fin)$, esto involucraría desplazar al resto de los elementos una posición hacia la izquierda, lo cual implicaría que la operación tuviese que ser implementada con complejidad lineal sobre el tamaño de la cola. Esto en sí mismo no parece ser un gran problema, y de hecho la operación de eliminación sugerida por la re-especificación de implementación en el caso de *Diccionario* resulta ser de complejidad lineal, pero en el caso de nuestro TAD *Cola* podemos obtener mejor complejidad. Para ello, en lugar de insistir en la posición 0 como inicio de la cola, basta introducir un segundo índice entero *inic* que indique el inicio de la cola, avanzando a éste una posición hacia la derecha cada vez que se desencole un elemento, en lugar de desplazar a los elementos dentro del arreglo. Esto permite que la operación de desencolar sea implementable con la misma complejidad constante con la que es implementable la operación de encolar.

Tenemos entonces por ahora que el segmento del arreglo que representaría a la cola sería $[inic..fin)$. El índice *inic* señala por dónde desencolar elementos, mientras *fin* señala por dónde encolar nuevos elementos. Ahora bien, cuando al encolar un elemento el índice *fin* llegue al extremo derecho del arreglo, no tiene sentido declarar a la cola como llena si se tiene que *inic* ha avanzado desde su posición inicial de 0 debido a desencolamientos previos. Si diésemos la cola por llena en un caso así, estaríamos desaprovechando parte del arreglo que no está en uso, específicamente

el segmento $[\,0..inic\,)$ del arreglo, y además estaríamos irrespetando la especificación original, la cual exige que una cola sea considerada llena sólo cuando cuenta con MAX elementos. Esta situación la podemos manejar utilizando al arreglo de manera "circular", considerando que la última casilla del arreglo tiene como vecina a su derecha a la casilla inicial del arreglo. Así, luego de la posición $MAX - 1$, estaría ubicada la posición 0, lo que nos permite reusar el segmento $[\,0..inic\,)$ para continuar agregando elementos a la cola que en este momento se encontraría en el segmento $[\,inic..MAX\,)$. Llevando *fin* "circularmente" en este caso a 0 y continuando así otros encolamientos, terminaríamos con que la cola correspondería en tal momento al segmento $[\,inic..MAX\,)$ seguido del segmento $[\,0..fin\,)$ del arreglo.

Nuestro modelo concreto tendría entonces ahora la siguiente forma:

Modelo de Representación
const MAX : int
var *elems* : array $[\,0..MAX\,)$ of T
$inic, fin$: int ,

donde los índices enteros delimitarían la cola. Según lo discutido en el párrafo anterior, esta delimitación será en ocasiones continua convencional y en ocasiones "circular". Analicemos ahora cómo discriminar la utilización de un estilo u otro de delimitación.

Cuando se tenga que $inic < fin$, tendremos una cola almacenada de manera continua convencional en el segmento $[\,inic..fin\,)$ del arreglo. Por el contrario, cuando se tenga que $inic > fin$, tendremos una cola almacenada de manera "circular" en el segmento $[\,inic..MAX\,)$ seguido del segmento $[\,0..fin\,)$ del arreglo. Ahora bien, el caso $inic = fin$ es problemático, pues puede ser interpretado de dos maneras: se tiene una cola vacía en $[\,inic..fin\,)$, o se tiene una cola llena con MAX elementos en $[\,inic..MAX\,)$ seguido de $[\,0..fin\,)$. Para poder distinguir estas dos situaciones, utilizaremos un atributo booleano que nos indique si la cola está vacía o no, para así poder discenir qué tenemos cuando los dos índices enteros sean iguales (simétricamente, también podríamos haber utilizado un booleano que indicase si la cola está llena o no; el ejercicio **5.2-a** explora esta otra posibilidad). Esto completa nuestro modelo de la siguiente forma:

Modelo de Representación
const MAX : int
var *elems* : array $[\,0..MAX\,)$ of T
$inic, fin$: int
nada : boolean .

La figura **5.0.(e)** muestra la re-especificación completa de nuestro TAD *Cola* con modelo concreto de representación para implementación correspondiente a arreglo "circular".

En cuanto al invariante de representación, note que a los índices enteros no se les permite alcanzar el valor MAX (a diferencia de, por ejemplo, el índice entero *tam* de la implementación de *Diccionario*), ya que el manejo "circular" involucra considerar a 0 como el sucesor de $MAX - 1$. Además, la tercera y última restricción elimina la posibilidad de inconsistencia entre el booleano y los índices enteros: no puede ocurrir que la cola esté vacía y los índices sean diferentes.

La relación de acoplamiento discrimina si la cola está almacenada de manera continua o "circular", construyendo la secuencia correspondiente de manera apropiada. Las condiciones de discriminación han sido simplificadas en la figura **5.0.(e)**. Más detalladamente, estas condiciones corresponderían a las siguientes fórmulas: el almacenamiento continuo ocurre cuando se tiene

$$inic < fin \ \vee \ (inic = fin \ \wedge \ nada) \quad ,$$

Especificación $\mathbb{B}$ de TAD *Cola* (T), refinamiento de $\mathbb{A}$

Modelo de Representación

const MAX : int
var $elems$: array $[\,0..MAX\,)$ of T
$inic, fin$: int
$nada$: boolean

Invariante de Representación

$MAX > 0 \;\wedge\; 0 \leqslant inic, fin < MAX \;\wedge\; (nada \;\Rightarrow\; inic = fin)$

Relación de Acoplamiento

$(inic < fin \;\vee\; nada \;\Rightarrow\; contenido = \langle\, i \,:\, inic \rightarrow fin \,:\, elems[i] \,\rangle)$
$\wedge$
$(inic \geqslant fin \;\wedge\; \neg\, nada \;\Rightarrow$
$contenido = \langle\, i \,:\, inic \rightarrow MAX \,:\, elems[i] \,\rangle +\!\!+ \langle\, i \,:\, 0 \rightarrow fin \,:\, elems[i] \,\rangle)$

Operaciones

proc *crear* (**in** m : int ; **out** c : *Cola*)
{ **Pre** : $m > 0$ }
{ **Post** : $c.MAX = m \;\wedge\; c.inic = 0 \;\wedge\; c.fin = 0 \;\wedge\; c.nada$ }

proc *encolar* (**in-out** c : *Cola* ; **in** x : T)
{ **Pre** : $c.inic \neq c.fin \;\vee\; c.nada$ }
{ **Post** : $c.inic = c_0.inic \;\wedge\; c.fin = (c_0.fin + 1) \bmod c.MAX$
$\wedge\; c.elems = c_0.elems\,(c_0.fin : x) \;\wedge\; \neg\, c.nada$ }

proc *desencolar* (**in-out** c : *Cola*)
{ **Pre** : $\neg\, c.nada$ }
{ **Post** : $c.inic = (c_0.inic + 1) \bmod c.MAX \;\wedge\; c.fin = c_0.fin$
$\wedge\; c.elems = c_0.elems \;\wedge\; (c.nada \;\equiv\; (c.inic = c.fin))$ }

proc *primero* (**in** c : *Cola* ; **out** x : T)
{ **Pre** : $\neg\, c.nada$ }
{ **Post** : $x = c.elems[c.inic]$ }

proc *vacia* (**in** c : *Cola* ; **out** v : boolean)
{ **Pre** : true }
{ **Post** : $v \equiv c.nada$ }

Fin TAD

Figura **5.0.(e)**: Re-especificación con modelo concreto del TAD *Cola*

mientras el almacenamiento "circular" ocurre cuando se tiene

$$inic > fin \ \vee \ (inic = fin \ \wedge \ \neg \, nada) \quad .$$

Sin embargo, el invariante de representación nos permite obtener fórmulas más concisas. Para el almacenamiento continuo, simplificamos como se indica a continuación:

$$\begin{array}{ll} & inic < fin \ \vee \ (inic = fin \ \wedge \ nada) \\ \equiv & \quad \langle \text{ por invariante de representación, } \ nada \ \Rightarrow \ inic = fin \, \text{; lógica proposicional } \rangle \\ & inic < fin \ \vee \ nada \quad . \end{array}$$

Para el almacenamiento "circular", manipulamos de la siguiente forma:

$$\begin{array}{ll} & inic > fin \ \vee \ (inic = fin \ \wedge \ \neg \, nada) \\ \equiv & \quad \langle \text{ distributividad de disyunción sobre conjunción, relaciones aritméticas } \rangle \\ & inic \geqslant fin \ \wedge \ (inic > fin \ \vee \ \neg \, nada) \\ \equiv & \quad \left\langle \begin{array}{l} \text{por invariante de representación y contrarrecíproco, } \ inic \neq fin \ \Rightarrow \ \neg \, nada \, \text{;} \\ \text{por relaciones aritméticas, } \ inic > fin \ \Rightarrow \ inic \neq fin \, \text{; lógica proposicional} \end{array} \right\rangle \\ & inic \geqslant fin \ \wedge \ \neg \, nada \quad . \end{array}$$

Continuando con la relación de acoplamiento, una vez discriminado en qué segmento o segmentos está almacenada la cola, se determina la secuencia correspondiente al atributo abstracto *contenido*. Explicamos brevemente a continuación nuestra notación para definir secuencias por comprensión, que es de la forma $\langle \, x \, : \, a \rightarrow b \, : \, E \, \rangle$, con x una variable generadora, a y b enteros, y E una expresión del tipo deseado para los elementos de la secuencia. A diferencia de la notación para definir conjuntos por comprensión, el rango o componente medio que determina los valores a utilizar para la variable x no es una expresión booleana sino una secuencia, ya que para conjuntos el orden de tales valores no era relevante pero ahora para secuencias sí lo es. El rango $a \rightarrow b$ denota a la secuencia formada por los enteros del segmento $[\, a..b \,)$ en orden ascendente, de la cual toma valores x de manera ordenada. Para cada uno de tales valores se utiliza la expresión E para determinar los elementos de la secuencia en construcción.

De nuevo nos hemos desviado de la notación de [Mor94], donde al igual que antes se utiliza a "|" y "·" como separadores, pero además se mantiene el carácter booleano del rango cambiando en su lugar al tipo de la variable generadora por una secuencia en lugar de un conjunto. Nuestra $\langle \, x \, : \, a \rightarrow b \, : \, E \, \rangle$ sería en el texto en cuestión $\langle \, x : a \rightarrow b \mid \text{true} \cdot E \, \rangle$. Mantener el rango booleano da a [Mor94] mayor poder expresivo, pero nuestro limitado uso de secuencias definidas por comprensión permite que nos mantengamos con nuestra más sencilla versión. Más información sobre tal notación más poderosa para definir secuencias por comprensión en [Mor94, sección 9.3.3].

Volviendo a nuestra re-especificación del TAD, y pasando ahora a las pre/post-condiciones de las operaciones, note que bajo nuestro modelo concreto una cola c está llena exactamente cuando

$$c.inic = c.fin \ \wedge \ \neg \, c.nada \quad ,$$

mientras que está vacía exactamente cuando se cumple

$$c.nada$$

(fórmula más sencilla gracias al invariante de representación). La negación de estas dos condiciones es usada, respectivamente, en la precondición de *encolar* para asegurar la existencia de espacio

para el nuevo elemento, y en las precondiciones de *desencolar* y *primero* para asegurar la existencia de elementos.

El manejo "circular" para los índices se logra mediante aritmética modular con la operación de resto de división entera. Así, por ejemplo, avanzar el índice *fin* de una cola c se indica mediante

$$c.\mathit{fin} = (c_0.\mathit{fin} + 1) \bmod c.\mathit{MAX} \quad ,$$

lo cual corresponde a $c.\mathit{fin} = c_0.\mathit{fin} + 1$ si se tiene que $c_0.\mathit{fin} < c.\mathit{MAX} - 1$, y a $c.\mathit{fin} = 0$ si se tiene que $c_0.\mathit{fin} = c.\mathit{MAX} - 1$. Análogamente se especifica el avance del otro índice *inic*.

Por último, vale la pena comentar la importancia de no olvidar especificar en las postcondiciones el comportamiento del atributo booleano *nada* cuando alguna cola es creada o cambiada (esto es, cuando una cola es parámetro de salida o de entrada/salida). En los casos de *crear* y *encolar* es sencillo, pues el valor final es fijo. En el caso de *desencolar* depende del valor final de los índices enteros; note que es importante usar una equivalencia para determinar el valor del atributo booleano en todos los casos (la igualdad de los índices no puede confundirse en este caso con la posibilidad de cola llena pues se acaba de desencolar un elemento).

Tal como se adelantó al inicio de esta sección **5**, esta implementación de *Cola* con un arreglo "circular" es una solución clásica de implementación de este TAD. Otra presentación de esta solución puede conseguirse en [CLRS09, sección 10.1], aunque allí se usa la variante sin el booleano discutida en el ejercicio **5.2-a**.

5.1. Una implementación del TAD *Conjunto*

Escogemos para nuestro TAD *Conjunto* especificado en la sección **3.1** un modelo concreto de representación análogo al utilizado para el TAD *Diccionario*: un arreglo cuyo tamaño corresponda a la capacidad máxima y un índice que señale el segmento inicial del arreglo que realmente almacena a los elementos. Las figuras **5.1.(f)** y **5.1.(g)** muestran la re-especificación completa con modelo concreto. El invariante de representación y la relación de acoplamiento son análogos a los utilizados en el caso de *Diccionario*.

Note que la postcondición de *agregar* es más compleja que la utilizada en el caso de *Diccionario* (a pesar de tener un arreglo menos en el modelo). Esto se debe al carácter menos restrictivo de la precondición correspondiente, pues en el caso de *Diccionario* sólo se podía agregar claves no existentes previamente, mientras que ahora se permite que el elemento a agregar pueda estar o no previamente en la colección. Discriminar entre estas dos posibilidades genera el análisis de casos que puede verse en la postcondición, según el cual el modelo debe ser actualizado o no.

En el caso de la postcondición correspondiente en la especificación original con modelo abstracto, el análisis por casos no hizo falta pues todas las posibilidades son manejadas adecuadamente por el operador $\cup$ de unión de conjuntos. Esto podría haberse explotado también en esta re-especificación con modelo concreto, usando como postcondición a

$$\{\, i \,:\, 0 \leqslant i < c.\mathit{tam} \,:\, c.\mathit{elems}[i] \,\} \;=\; \{\, i \,:\, 0 \leqslant i < c_0.\mathit{tam} \,:\, c_0.\mathit{elems}[i] \,\} \cup \{\, x \,\} \quad .$$

Note que esta nueva propuesta de postcondición es una especie de traducción directa de la postcondición original, re-expresando lo referido al atributo abstracto *contenido* con los atributos concretos *elems* y *tam* según lo indicado por la relación de acoplamiento. Esto es un ejemplo de las re-especificaciones concretas implícitas inducidas por relaciones de acoplamiento de las que se habló en la sección **4.3** y que serán elaboradas con más detalle en la futura sección **6**.

Analicemos brevemente, en el caso de *agregar*, posibles ventajas y desventajas de la re-especificación implícita inducida por la relación de acoplamiento presentada en el párrafo anterior

---

Especificación $\mathbb{B}$ de TAD *Conjunto* (T), refinamiento de $\mathbb{A}$

Modelo de Representación

const MAX : int
var $elems$: array $[\,0..MAX\,)$ of T
tam : int

Invariante de Representación

$MAX > 0 \;\wedge\; 0 \leqslant tam \leqslant MAX$
$\wedge\; (\,\forall\, i, j \,:\, 0 \leqslant i, j < tam \,:\, i \neq j \;\Rightarrow\; elems[i] \neq elems[j]\,)$

Relación de Acoplamiento

$contenido = \{\, i \,:\, 0 \leqslant i < tam \,:\, elems[i] \,\}$

Operaciones

proc *crear* (**in** m : int ; **out** c : *Conjunto*)
{ **Pre** : $m > 0$ }
{ **Post** : $c.MAX = m \;\wedge\; c.tam = 0$ }

proc *agregar* (**in-out** c : *Conjunto* ; **in** $x : T$)
{ **Pre** : $\neg\,(\,\exists\, i \,:\, 0 \leqslant i < c.tam \,:\, c.elems[i] = x\,) \;\Rightarrow\; c.tam < c.MAX$ }
{ **Post** : $(\,(\,\exists\, i \,:\, 0 \leqslant i < c_0.tam \,:\, c_0.elems[i] = x\,) \;\Rightarrow$
$c.tam = c_0.tam \;\wedge\; c.elems = c_0.elems)$
$\wedge$
$(\neg\,(\,\exists\, i \,:\, 0 \leqslant i < c_0.tam \,:\, c_0.elems[i] = x\,) \;\Rightarrow$
$c.tam = c_0.tam + 1 \;\wedge\; c.elems = c_0.elems\,(c_0.tam : x))$ }

proc *eliminar* (**in-out** c : *Conjunto* ; **in** $x : T$)
{ **Pre** : true }
{ **Post** : $(\neg\,(\,\exists\, i \,:\, 0 \leqslant i < c_0.tam \,:\, c_0.elems[i] = x\,) \;\Rightarrow$
$c.tam = c_0.tam \;\wedge\; c.elems = c_0.elems)$
$\wedge$
$(\,(\,\exists\, i \,:\, 0 \leqslant i < c_0.tam \,:\, c_0.elems[i] = x\,) \;\Rightarrow$
$c.tam = c_0.tam - 1 \;\wedge$
$(\,\exists\, i \,:\, 0 \leqslant i < c_0.tam \,:\, c_0.elems[i] = x$
$\wedge\; c.elems = c_0.elems\,(i : c_0.elems[\,c_0.tam - 1\,])\,))$ }

proc *extraer* (**in** c : *Conjunto* ; **out** $x : T$)
{ **Pre** : $c.tam > 0$ }
{ **Post** : $(\,\exists\, i \,:\, 0 \leqslant i < c.tam \,:\, c.elems[i] = x\,)$ }

proc *pertenece* (**in** c : *Conjunto* ; **in** $x : T$; **out** p : boolean)
{ **Pre** : true }
{ **Post** : $p \equiv (\,\exists\, i \,:\, 0 \leqslant i < c.tam \,:\, c.elems[i] = x\,)$ }

$\vdots$

Figura **5.1.(f)**: Re-especificación con modelo concreto del TAD *Conjunto* (inicio)

$\vdots$

proc *vacio* (**in** c : *Conjunto* ; **out** v : boolean)

$\{$ **Pre** : true $\}$

$\{$ **Post** : $v \equiv (c.tam = 0)$ $\}$

proc *unir* (**in** *c0*, *c1* : *Conjunto* ; **in** m : int ; **out** c : *Conjunto*)

$\{$ **Pre** : $c0.tam + c1.tam \leqslant m$ $\}$

$\{$ **Post** : $c.MAX = m$

$\wedge \ \{ i : 0 \leqslant i < c.tam : c.elems[i] \} =$

$\{ i : 0 \leqslant i < c0.tam : c0.elems[i] \}$

$\cup \ \{ i : 0 \leqslant i < c1.tam : c1.elems[i] \}$ $\}$

proc *intersectar* (**in** *c0*, *c1* : *Conjunto* ; **in** m : int ; **out** c : *Conjunto*)

$\{$ **Pre** : $c0.tam \leqslant m \ \wedge \ c1.tam \leqslant m$ $\}$

$\{$ **Post** : $c.MAX = m$

$\wedge \ \{ i : 0 \leqslant i < c.tam : c.elems[i] \} =$

$\{ i : 0 \leqslant i < c0.tam : c0.elems[i] \}$

$\cap \ \{ i : 0 \leqslant i < c1.tam : c1.elems[i] \}$ $\}$

proc *restar* (**in** *c0*, *c1* : *Conjunto* ; **in** m : int ; **out** c : *Conjunto*)

$\{$ **Pre** : $c0.tam \leqslant m$ $\}$

$\{$ **Post** : $c.MAX = m$

$\wedge \ \{ i : 0 \leqslant i < c.tam : c.elems[i] \} =$

$\{ i : 0 \leqslant i < c0.tam : c0.elems[i] \}$

$- \ \{ i : 0 \leqslant i < c1.tam : c1.elems[i] \}$ $\}$

Fin TAD

Figura **5.1.(g)**: Re-especificación con modelo concreto del TAD *Conjunto* (fin)

contra la re-especificación explícita utilizada en la figura **5.1.(f)**. La postcondición implícita es concisa, además de deducible automáticamente, ambas cosas puntos a su favor. Por otra parte, no sugiere casi nada para la implementación del cuerpo de la operación; sólo dice algo como "haga lo que quiera siempre y cuando el conjunto final representado sea el adecuado". Esto tiene la ventaja de mantener abiertas todas las posibilidades existentes de implementación del cuerpo, pero tiene la desventaja de no sugerir a ninguna de tales posibilidades como deseable. Por el contrario, la postcondición explícita presentada en la figura **5.1.(f)** sugiere cómo implementar el cuerpo: haga análisis por casos y luego actualice adecuadamente al modelo. Esto tiene la ventaja de guiar la implementación, pero la desventaja de descartar otras posibles implementaciones (por ejemplo, ubicar al nuevo elemento en otra posición diferente a la última). Podemos ver que no hay respuesta clara a qué es mejor al decidir entre re-especificaciones implícitas y re-especificaciones explícitas que restringen posibles implementaciones.

A la postcondición de *eliminar* le aplica analógamente todo el análisis hecho con *agregar*.

En cuanto a *unir*, *intersectar* y *restar*, note que las postcondiciones de la re-especificación explícita propuesta corresponden exactamente a la re-especificación implícita inducida por la relación de acoplamiento. En estos tres casos, cualquier otro intento de re-especificación de la postcondición sería considerablemente más complejo; se le sugiere que intente y se convenza de esto personalmente.

5.2. Ejercicios

5.2-a. *. . . cambiar modelo de implementación del TAD Cola; explorar varias posibilidades: cambio de interpretación de booleano, cambio de booleano por entero, cambio de booleano por casilla extra. . .*

5.2-b. *. . . una implementación del TAD ColaConPrioridades. . .*

5.2-c. *. . . una implementación del TAD Pila. . .*

5.2-d. *. . . cambiar modelo de implementación del TAD Conjunto, por ejemplo exigiendo que el arreglo esté ordenado y analizando implementaciones de unión e intersección. . .*

5.2-e. *. . . dos implementaciones del TAD MultiConjunto. . .*

6. Consistencia Implementación vs. Especificación

Tal como hemos señalado ya repetidamente, en estas notas consideramos a una especificación de TAD con modelo abstracto como el contrato original que se le exige a un TAD o, en otras palabras, como la especificación propiamente dicha del TAD, mientras consideramos a cualquier subsiguiente re-especificación del TAD con modelo concreto como un diseño intermedio entre el contrato original y una implementación final ya con todas las operaciones completamente construidas. Ahora bien, el sentido común indica que tal diseño intermedio debe ser consistente con el contrato original, para que tenga sentido continuar construyendo una implementación por esa vía.

Para mostrar tal consistencia entre una especificación con modelo abstracto de un TAD, o contrato original del TAD, y una re-especificación con modelo concreto del mismo, o diseño intermedio hacia una implementación completa, existen técnicas llamadas de *refinamiento de datos*.

El nombre proviene del hecho de que una especificación se ha de refinar, o transformar hacia una implementación, mediante un cambio en el modelo de datos utilizado: exactamente nuestro caso.

Entre las dos especificaciones a ser mostradas como consistentes tenemos como componentes comunes a (i) el modelo de representación, (ii) el invariante de representación, y (iii) las operaciones, mientras que el único componente no-común será precisamente el que servirá de puente, que es la relación de acoplamiento. El componente (i) es justamente lo que cambia: no tenemos nada que decir allí. Los componentes (ii) y (iii) deben guardar una cierta relación de consistencia; de esto nos ocuparemos en las siguientes subsecciones.

6.0. Invariante de Representación

Llamemos *InvA* al invariante de representación de la especificación original con modelo abstracto, llamemos *InvC* al invariante de representación de la re-especificación con modelo concreto, y por último llamemos *Ac* a la relación de acoplamiento entre ambos modelos.

Lo que necesitamos es asegurar que la implementación con modelo concreto garantice que los requerimientos iniciales con modelo abstracto sean satisfechos. Por lo tanto, lo que necesitamos garantizar es que el invariante concreto *InvC* implique que también se cumple el invariante abstracto *InvA*. Como estos invariantes están definidos en espacios diferentes, se requiere asumir como hipótesis al acoplamiento *Ac* para poder establecer la implicación deseada. Lo que ha de demostrarse es entonces

$$[\ Ac \wedge InvC \ \Rightarrow\ InvA\ ] \quad ,$$

lo cual expresa de manera más sencilla la implicación arriba sugerida en lenguaje natural, que formalizada literalmente sería $[\ Ac \ \Rightarrow\ (InvC \ \Rightarrow\ InvA)\ ]$.

Nota:

Utilizamos los corchetes $[\ldots]$ como cuantificación universal implícíta, de la misma manera que es utilizada en [Kal90], tal como fue propuesto inicialmente por Dijkstra y compañía en [DS90].

(Fin de nota.)

Para demostrar este requerimiento en el caso de nuestras especificación e implementación propuestas en las secciones anteriores para el TAD *Diccionario*, suponemos primero todas las hipótesis correspondientes a *Ac* e *InvC*:

$$\begin{array}{ll}
\text{(Hip0)} & conoc = \{\, i \,:\, 0 \leqslant i < tam \,:\, claves[i] \,\} \\
\text{(Hip1)} & tabla \;=\; \{\, i \,:\, 0 \leqslant i < tam \,:\, (claves[i], valores[i]) \,\} \\
\text{(Hip2)} & MAX > 0 \\
\text{(Hip3)} & 0 \leqslant tam \leqslant MAX \\
\text{(Hip4)} & (\,\forall\, i, j \,:\, 0 \leqslant i, j < tam \,:\, i \neq j \ \Rightarrow\ claves[i] \neq claves[j]\,) \quad ,
\end{array}$$

y debemos demostrar como consecuente a las afirmaciones de *InvA*:

$$\begin{array}{ll}
\text{(Cons0)} & MAX > 0 \\
\text{(Cons1)} & \#\, conoc \leqslant MAX \\
\text{(Cons2)} & conoc = \mathrm{dom}\, tabla \quad .
\end{array}$$

Procedamos entonces con las demostraciones correspondientes...

Primero, (Cons0) es consecuencia trivial de la hipótesis (Hip2).

Luego, en relación con (Cons1), razonamos de la siguiente forma:

$$
\begin{array}{ll}
 & \#\, conoc \\
= & \quad \langle \text{ hipótesis (Hip0) } \rangle \\
 & \#\,\{\, i \,:\, 0 \leqslant i < tam \,:\, claves[i] \,\} \\
= & \quad \left\langle \begin{array}{l} \text{hipótesis (Hip4) garantiza que todos los elementos generados} \\ \text{para el conjunto son diferentes y, por lo tanto, la cantidad de} \\ \text{ellos es exactamente igual a la cantidad de valores del rango} \end{array} \right\rangle \\
 & tam \\
\leqslant & \quad \langle \text{ hipótesis (Hip3) } \rangle \\
 & MAX \quad .
\end{array}
$$

Note que la hipótesis (Hip4) no es imprescindible ya que, de no haber contado con ésta, el segundo paso se habría podido dar con una desigualdad "$\leqslant$" en lugar de una igualdad "=". Esto se debe a que, de haber podido existir claves repetidas en el segmento $[\,0..tam\,)$ del arreglo *claves*, habría a lo sumo *tam* claves en el conjunto *conoc*, posiblemente menos.

Por último, para demostrar (Cons2) partimos del lado derecho (que es el más complejo) y llegamos al lado izquierdo de la siguiente forma:

$$
\begin{array}{ll}
 & \mathrm{dom}\, tabla \\
= & \quad \langle \text{ hipótesis (Hip1) } \rangle \\
 & \mathrm{dom}\,\{\, i \,:\, 0 \leqslant i < tam \,:\, (claves[i], valores[i]) \,\} \\
= & \quad \langle \text{ "dom" extrae las primeras componentes } \rangle \\
 & \{\, i \,:\, 0 \leqslant i < tam \,:\, claves[i] \,\} \\
= & \quad \langle \text{ hipótesis (Hip0) } \rangle \\
 & conoc \quad .
\end{array}
$$

Listo: hemos demostrado el requerimiento de consistencia $[\; Ac \wedge InvC \;\Rightarrow\; InvA \;]$ correspondiente a los invariantes de representación para nuestras especificaciones del TAD *Diccionario*.

6.1. Operaciones

Para cada una de las operaciones se deberá mostrar que su re-especificación con modelo concreto es consistente con la especificación original con modelo abstracto. Tomemos entonces una operación cualquiera y supongamos que su especificación original estaba dada por el par pre/post-condición *PreA* y *PostA* ("$\ldots A$" por abstractas) mientras la re-especificación está dada por *PreC* y *PostC* ("$\ldots C$" por concretas).

Primera versión (excesivamente fuerte) de la consistencia Lo ideal es que ambas especificaciones de la operación digan exactamente lo mismo, por lo que el objetivo pareciera ser demostrar

$$[\quad \ldots \;\; PreA \equiv PreC \;]$$

y

$$[\quad \ldots \quad PostA \equiv PostC \ ] \quad .$$

Sin embargo, no tiene sentido intentar demostrar estas equivalencias por sí solas, ya que las fórmulas involucradas se refieren a modelos diferentes. Esto es, *PreA* y *PostA* están formuladas sobre el modelo abstracto y, por lo tanto, no tiene sentido intentar demostrar que son equivalentes a *PreC* y *PostC*, que están formuladas sobre el modelo concreto. Debemos entonces echar mano de nuestro puente: la relación de acoplamiento entre ambos modelos.

Si el nuevo TAD en definición y construcción es T, los modelos abstracto y concreto son utilizados sobre los parámetros de tipo T de la operación. Las precondiciones sólo hacen referencia a parámetros que son dados como entrada a la operación, por lo que la conexión debe construirse haciendo uso de la relación de acoplamiento aplicada sobre todo parámetro que sea dato de entrada. Con tal puente como hipótesis tiene entonces sentido intentar demostrar la consistencia deseada entre las dos precondiciones:

$$[\ (\forall\, x \,:\, x \text{ es parámetro } \textbf{in} \text{ o } \textbf{in-out} \text{ de tipo } T \,:\, x.Ac\ ) \ \Rightarrow \ (PreA \equiv PreC)\ ] \quad ,$$

entendiendo que $x.Ac$ denota las condiciones exigidas por la relación de acoplamiento Ac particularizadas para los atributos del objeto x. Por otra parte, las postcondiciones pueden referirse a: (i) los parámetros de salida, (ii) tanto el estado final como el estado inicial de los parámetros de entrada/salida, y (iii) los parámetros de entrada (a los cuales, recuerde, por convención consideramos constantes). La hipótesis puente debe entonces en este caso considerar a todos estos parámetros:

$$\begin{aligned}[\ &(\forall\, x \,:\, x \text{ es parámetro } \textbf{in}, \textbf{out} \text{ o } \textbf{in-out} \text{ de tipo } T \,:\, x.Ac\ ) \\ &\quad \wedge \ (\forall\, x \,:\, x \text{ es parámetro } \textbf{in-out} \text{ de tipo } T \,:\, x_0.Ac\ ) \ \Rightarrow \ (PostA \equiv PostC)\ ] \quad ,\end{aligned}$$

donde, de nuevo, $x.Ac$ se refiere a la relación de acoplamiento formulada sobre los atributos del objeto x e, igualmente, $x_0.Ac$ se refiere al acoplamiento formulado sobre los atributos de x_0, esto es, el valor inicial del parámetro (de entrada/salida) x.

Hay un último detalle a considerar relacionado con los invariantes de representación, el cual, al ser discutido, puede además dilucidar una potencial duda que las personas lectoras podrían tener en este momento. Recuerde que los invariantes de representación sobre los parámetros de tipo T de las operaciones son considerados parte implícita de la precondición o de la postcondición (dependiendo de si el parámetro es de entrada, de salida, o de entrada/salida). Sin embargo, en las demostraciones de refinamiento de datos basta sólo considerar los componentes explícitos en los pares pre/post-condición, lo cual significa que arriba nos referíamos con *PreA*, *PostA*, *PreC* y *PostC* a las pre/post-condiciones <u>explícitas</u> de las operaciones. Más aún, los componentes implícitos de los pares pre/post-condición, esto es, los relacionados con invariantes de representación, no sólo no hace falta incluirlos en los consecuentes a demostrar en las reglas de refinamiento de datos, sino que pueden ser supuestos como hipótesis en tales demostraciones. Esto es:

$$\begin{aligned}[\ &(\forall\, x \,:\, x \text{ es parámetro } \textbf{in} \text{ o } \textbf{in-out} \text{ de tipo } T \,:\, x.Ac \wedge x.InvA \wedge x.InvC\ ) \\ &\qquad\qquad \Rightarrow \ (PreA \equiv PreC)\ ]\end{aligned}$$

y

$$\begin{aligned}[\ &(\forall\, x \,:\, x \text{ es parámetro } \textbf{in}, \textbf{out} \text{ o } \textbf{in-out} \text{ de tipo } T \,:\, x.Ac \wedge x.InvA \wedge x.InvC\ ) \\ &\quad \wedge \ (\forall\, x \,:\, x \text{ es parámetro } \textbf{in-out} \text{ de tipo } T \,:\, x_0.Ac \wedge x_0.InvA \wedge x_0.InvC\ ) \\ &\qquad\qquad \Rightarrow \ (PostA \equiv PostC)\ ] \quad ,\end{aligned}$$

donde, al igual que antes con $x.Ac$ y $x_0.Ac$, las expresiones $x.InvA$, $x.InvC$, $x_0.InvA$ e $x_0.InvC$ se refieren a los invariantes de representación particularizados a los objetos x y x_0. Note además que, gracias a haber demostrado previamente que en presencia de la relación de acoplamiento el invariante concreto *InvC* implica al invariante abstracto *InvA*, en las hipótesis antes referidas es formalmente equivalente el listar solamente al invariante concreto *InvC*. Por mayor claridad, mantenemos las reglas con la enumeración completa de ambos invariantes, con lo que además se facilita el tener mayor conciencia sobre todas las hipótesis disponibles para las demostraciones.

El hecho de poder suponer los invariantes de representación como hipótesis tiene que ver con una propiedad llamada *inducción sobre tipos de datos* (ver, por ejemplo, [Lis01, sección 5.7.1]). Esto es: si

- toda operación que produce inicialmente como parámetros de salida a objetos de tipo T garantiza que tales objetos cumplan con el invariante de representación (caso base), y
- toda operación que recibe como entrada objetos de tipo T y produce como salida otros objetos de tipo T, al suponer que los de entrada cumplen con el invariante de representación, garantiza que los nuevos objetos de salida también lo cumplan (caso inductivo),

entonces todo objeto de tipo T producido por tal conjunto de operaciones cumplirá siempre el invariante de representación. Por ello, tales invariantes pueden ser supuestos como ciertos al demostrar la equivalencia entre las precondiciones abstracta/concreta y las postcondiciones abstracta/concreta.

Cápsula breve sobre la presencia/ausencia de invariantes de representación

- Los invariantes de representación deben ser considerados *presentes implícitamente* en las pre/post-condiciones al momento de implementar las operaciones. Esto es, al construir los cuerpos de los procedimientos/funciones correspondientes a las operaciones, las pre/post-condiciones *con* los invariantes de representación son el contrato a cumplir.
- Los invariantes de representación deben ser considerados *ausentes* de las pre/post-condiciones al momento de demostrar las condiciones de consistencia correspondientes a refinamiento de datos (ya que los invariantes de representación pueden ser supuestos como ciertos gracias a inducción de tipos de datos).

Segunda versión (más débil) de la consistencia La versión ya presentada de la consistencia entre los pares pre/post-condición abstractos vs. concretos es excesivamente fuerte. Esto se debe a que no hace falta que los consecuentes a demostrar sean equivalencias: $PreA \equiv PreC$ y $PostA \equiv PostC$. Para que el refinamiento de datos sea correcto no hace falta que estas equivalencias sean válidas, y de hecho en muchos casos estas equivalencias no serán ciertas (luego veremos que en nuestro ejemplo del TAD *Diccionario* algunas de ellas no son ciertas). Basta con que se cumpla sólo una de las dos implicaciones presente en tales equivalencias. Sin embargo, entender cuál es la implicación requerida en cada caso requiere un análisis cuidadoso.

Recordemos un par de reglas sobre tripletas de Hoare que son de conocimiento común:

si la tripleta $\{ P \}\ I\ \{ Q \}$ es válida
y también se tiene $[P' \Rightarrow P]$,
entonces la tripleta $\{ P' \}\ I\ \{ Q \}$ también es válida ;

y

si la tripleta $\{ P \}\ I\ \{ Q \}$ es válida
y también se tiene $[Q \Rightarrow Q']$,
entonces la tripleta $\{ P \}\ I\ \{ Q' \}$ también es válida .

Éstas pueden ser combinadas en una única regla, como se especifica a continuación:

si la tripleta $\{ P \}\ I\ \{ Q \}$ es válida
y se tienen las implicaciones $[P' \Rightarrow P]$ y $[Q \Rightarrow Q']$,
entonces la tripleta $\{ P' \}\ I\ \{ Q' \}$ también es válida .

Demos la siguiente lectura "empresarial" a esta última regla combinada: "teniendo las implicaciones en cuestión, al cumplir con el contrato P/Q también se está entonces cumpliendo con el contrato P'/Q'". Y ahora volteemos los contratos y demos la siguiente re-lectura a la regla: "teniendo las implicaciones en cuestión, si el contrato original que se desea cumplir es P'/Q' entonces basta con cumplir el contrato P/Q."

Extrapolando esta última lectura a nuestro problema de refinamiento de datos, esto es, garantizar la consistencia entre la especificación abstracta y la especificación concreta de una operación, el contrato que debemos cumplir es $PreA/PostA$ y aspiramos que baste cumplir con una implementación del contrato $PreC/PostC$. De acuerdo con la regla arriba planteada lo que necesitamos entonces es cumplir con las implicaciones

$$[\quad \ldots \ \ PreA \ \Rightarrow \ PreC \ ]$$

y

$$[\quad \ldots \ \ PostC \ \Rightarrow \ PostA \ ] \quad .$$

Mas, como hay incompatibilidad en los espacios de datos utilizados entre las pre/post-condiciones abstractas y las concretas, necesitamos como hipótesis todos los puentes ya antes planteados a través de la relación de acoplamiento.

Nuestra nueva versión de las reglas sería entonces:

$$[\ (\forall x : x \text{ es parámetro } \textbf{in} \text{ o } \textbf{in-out} \text{ de tipo } T : x.Ac \wedge x.InvA \wedge x.InvC\ ) \Rightarrow\ (PreA \Rightarrow PreC)\ ]$$

y

$$[\ (\forall x : x \text{ es parámetro } \textbf{in}, \textbf{out} \text{ o } \textbf{in-out} \text{ de tipo } T : x.Ac \wedge x.InvA \wedge x.InvC\ ) \wedge\ (\forall x : x \text{ es parámetro } \textbf{in-out} \text{ de tipo } T : x_0.Ac \wedge x_0.InvA \wedge x_0.InvC\ ) \Rightarrow\ (PostC \Rightarrow PostA)\ ] \quad .$$

Estas nuevas reglas son más débiles y, por lo tanto, más fáciles de satisfacer.

Por último, hay una hipótesis adicional que aún podemos agregar a la regla de las postcondiciones. Mientras se demuestra la correspondencia entre postcondiciones, puede utilizarse también el hecho de que los parámetros de entrada (que son, recuerde, constantes) cumplían inicialmente con la precondición y que esto también ocurría para el valor inicial de los parámetros de entrada/salida. Por lo tanto, denotando Pre_0 al resultado de tomar la precondición Pre y sustituir en ella a todo parámetro de entrada/salida x por x_0, nuestra última versión de la regla (la más completa) para postcondiciones es:

$$[\ (\forall x : x \text{ es parámetro } \textbf{in}, \textbf{out} \text{ o } \textbf{in-out} \text{ de tipo } T : x.Ac \wedge x.InvA \wedge x.InvC\ ) \wedge\ (\forall x : x \text{ es parámetro } \textbf{in-out} \text{ de tipo } T : x_0.Ac \wedge x_0.InvA \wedge x_0.InvC\ ) \wedge\ PreA_0 \Rightarrow\ (PostC \Rightarrow PostA)\ ] \quad .$$

Resumen Para cada una de las operaciones de un TAD T cuya especificación original en términos del modelo abstracto esté dada por el par pre/post-condición $PreA/PostA$ y cuya re-especificación en términos del modelo concreto esté dada por el par pre/post-condición $PreC/PostC$, la consistencia entre ambas especificaciones debe ser garantizada demostrando

$$[\ (\forall\, x\ :\ x \text{ es parámetro } \textbf{in} \text{ o } \textbf{in-out} \text{ de tipo } T\ :\ x.Ac \wedge x.InvA \wedge x.InvC\ ) \Rightarrow\ (PreA \Rightarrow PreC)\ ]$$

y

$$[\ (\forall\, x\ :\ x \text{ es parámetro } \textbf{in}, \textbf{out} \text{ o } \textbf{in-out} \text{ de tipo } T\ :\ x.Ac \wedge x.InvA \wedge x.InvC\ ) \wedge\ (\forall\, x\ :\ x \text{ es parámetro } \textbf{in-out} \text{ de tipo } T\ :\ x_0.Ac \wedge x_0.InvA \wedge x_0.InvC\ ) \wedge\ PreA_0 \Rightarrow\ (PostC \Rightarrow PostA)\ ]\ ,$$

siendo Ac la relación de acoplamiento, $InvA$ el invariante de representación correspondiente al modelo abstracto, e $InvC$ el invariante de representación correspondiente al modelo concreto.

Crear En el caso de esta operación en nuestro TAD *Diccionario* de ejemplo, la precondición abstracta y la concreta son idénticas. Por lo tanto, ambas son equivalentes, aún sin suponer ninguna de las hipótesis de las que podemos disponer.

El caso de las postcondiciones no es tan directo. Teniendo que d es el único parámetro de tipo *Diccionario* de la operación, y siendo éste de salida (**out**), podemos tomar como hipótesis a $d.Ac$, $d.InvA$ y $d.InvC$. Tal como veremos en la demostración, sólo la hipótesis $d.Ac$ y una parte de $d.InvC$ terminan resultando necesarias para la demostración, por lo cual listamos a continuación sólo a tales componentes que necesitaremos:

$$\begin{array}{ll} \text{(Hip0)} & d.conoc = \{\, i\ :\ 0 \leqslant i < d.tam\ :\ d.claves[i]\,\} \\ \text{(Hip1)} & d.tabla\ =\ \{\, i\ :\ 0 \leqslant i < d.tam\ :\ (d.claves[i], d.valores[i])\,\} \\ \text{(Hip2)} & 0 \leqslant d.tam \leqslant d.MAX \quad . \end{array}$$

Suponiendo entonces las hipótesis señaladas, razonamos de la siguiente forma sólo sobre las porciones de las postcondiciones que no son idénticas sintácticamente:

$$\begin{array}{ll} & d.conoc = \emptyset\ \wedge\ d.tabla = \emptyset \\ \equiv & \quad \langle \text{ hipótesis (Hip0) e (Hip1) } \rangle \\ & \{\, i\ :\ 0 \leqslant i < d.tam\ :\ d.claves[i]\,\} = \emptyset \\ & \quad \wedge\ \{\, i\ :\ 0 \leqslant i < d.tam\ :\ (d.claves[i], d.valores[i])\,\} = \emptyset \\ \equiv & \quad \left\langle \begin{array}{l} \text{vacuidad de conjuntos definidos por comprensión:} \\ \text{ocurre si y sólo si el rango de generación es vacío} \end{array} \right\rangle \\ & \neg\,(\,\exists\, i\ ::\ 0 \leqslant i < d.tam\,) \\ \equiv & \quad \langle \text{ aritmética } \rangle \\ & d.tam \leqslant 0 \\ \equiv & \quad \langle \text{ por hipótesis (Hip2) se tiene } d.tam \geqslant 0 \ \rangle \\ & d.tam = 0 \quad . \end{array}$$

Nótese que en este caso hemos logrado demostrar la versión fuerte de la consistencia (según refinamiento de datos), ya que las postcondiciones resultaron equivalentes.

Note también que esta demostración la habríamos podido hacer de manera más simple. Recuerde que discutimos anteriormente (sección **2.2**) que, gracias a la presencia implícita de invariantes de representación en las pre/post-condiciones, podíamos disponer de varias versiones equivalentes de tales pre/post-condiciones. En el caso de la postcondición abstracta de *crear*, una de tales versiones equivalentes a la original (de nuevo ignorando la porción de la postcondición referente a $d.MAX$) era $d.conoc = \emptyset$. Utilizando esta versión, la demostración anterior habría sido un poco más sencilla.

He aquí la razón por la que se señaló anteriormente que era positivo el contar con versiones más breves y versiones más extensas y detalladas, pero equivalentes, de alguna especificación (ver, por ejemplo, la discusión final en la especificación de la operación *crear* en la sección **2.2**). Las más detalladas antes insistimos en utilizarlas en las especificaciones, gracias a que éstas, siendo más explícitas, son más claras y, por lo tanto, ayudan más a la comprensión de la especificación. Por otra parte, las más concisas son más útiles en las demostraciones de consistencia por refinamiento de datos, ya que pueden convertirse en consecuentes a demostrar más sencillos. (Sin embargo, si tal pre/post-condición no es el consecuente a demostrar sino una hipótesis, de nuevo sería mejor utilizar la versión más detallada y explícita.)

Agregar Para razonar sobre las precondiciones, podemos tomar como hipótesis toda la información del único parámetro d de tipo *Diccionario* de la operación, ya que éste es de entrada/salida (**in-out**); tal información es $d.Ac$, $d.InvA$ y $d.InvC$. De nuevo, detallamos a continuación sólo las porciones de estas hipótesis que terminaremos utilizando:

$$\begin{array}{ll} \text{(Hip0)} & d.conoc = \{\, i \,:\, 0 \leqslant i < d.tam \,:\, d.claves[i] \,\} \\ \text{(Hip1)} & (\,\forall\, i, j \,:\, 0 \leqslant i, j \leqslant d.tam \,:\, i \neq j \;\Rightarrow\; d.claves[i] \neq d.claves[j]\,) \quad . \end{array}$$

Razonamos ahora sobre las porciones claves de las precondiciones de la siguiente forma:

$$\begin{array}{ll} & c \in d.conoc \\ \equiv & \quad \langle \text{ hipótesis (Hip0) } \rangle \\ & c \in \{\, i \,:\, 0 \leqslant i < d.tam \,:\, d.claves[i] \,\} \\ \equiv & \quad \langle \text{ pertenencia a conjuntos definidos por comprensión } \rangle \\ & (\,\exists\, i \,:\, 0 \leqslant i < d.tam \,:\, d.claves[i] = c\,) \quad , \end{array}$$

y

$$\begin{array}{ll} & \#\, d.conoc < d.MAX \\ \equiv & \quad \langle \text{ hipótesis (Hip0) } \rangle \\ & \#\, \{\, i \,:\, 0 \leqslant i < d.tam \,:\, d.claves[i] \,\} \,<\, d.MAX \\ \equiv & \quad \left\langle \begin{array}{l} \text{gracias a hipótesis (Hip1) todos los elementos generados para} \\ \text{el conjunto son diferentes y, por lo tanto, la cantidad de ellos} \\ \text{es exactamente la cantidad de valores del rango: } d.tam \end{array} \right\rangle \\ & d.tam < d.MAX \quad . \end{array}$$

Discutamos un par de detalles sobre esta demostración. Primero, el razonamiento sobre la parte izquierda de las precondiciones optamos por realizarlo sin las negaciones involucradas. Esto es válido pues la misma negación se encontraba a ambos lados y, por propiedades de lógica proposicional, sabemos que es equivalente el razonar "cancelando" las negaciones a ambos lados, y fue

conveniente pues prescindir de las negaciones simplificó (aunque sólo ligeramente) las fórmulas involucradas en la demostración. Segundo, en el razonamiento sobre la parte derecha de las precondiciones note lo importante de haber utilizado la hipótesis (Hip1); de no haber contado con ella, sólo habríamos podido asegurar la desigualdad

$$\#\{\, i \,:\, 0 \leqslant i < d.tam \,:\, d.claves[i] \,\} \leqslant d.tam$$

debido a la posibilidad de elementos repetidos, en cuyo caso de la última equivalencia sólo habríamos podido asegurar la implicación "$\Leftarrow$" mas no la implicación "$\Rightarrow$". Esto nos habría estropeado la consistencia según refinamiento de datos deseada, ya que la regla exige que se cumpla la implicación desde la precondición abstracta hacia la concreta.

Para razonar sobre las postcondiciones, sobre nuestro único parámetro d de tipo *Diccionario*, en vista de que éste es de entrada/salida (**in-out**), podemos tomar como hipótésis a $d.Ac$, $d.InvA$ y $d.InvC$, y también a $d_0.Ac$, $d_0.InvA$ y $d_0.InvC$. A continuación las hipótesis que realmente se utilizarán:

$$\begin{array}{ll} \text{(Hip0)} & d.tabla \;=\{\, i \,:\, 0 \leqslant i < d.tam \,:\, (d.claves[i], d.valores[i]) \,\} \\ \text{(Hip1)} & d_0.tabla = \{\, i \,:\, 0 \leqslant i < d_0.tam \,:\, (d_0.claves[i], d_0.valores[i]) \,\} \\ \text{(Hip2)} & 0 \leqslant d_0.tam \leqslant d_0.MAX \quad . \end{array}$$

Por otra parte, para demostrar que la precondición concreta implica a la abstracta utilizaremos la técnica de suponer el antecedente de la implicación a demostrar y mostraremos que así se puede concluir el consecuente de ésta. Agregamos entonces las hipótesis correspondientes a la postcondición concreta:

$$\begin{array}{ll} \text{(Hip3)} & d.tam = d_0.tam + 1 \\ \text{(Hip4)} & d.claves = d_0.claves\,(d_0.tam : c) \\ \text{(Hip5)} & d.valores = d_0.valores\,(d_0.tam : v) \quad . \end{array}$$

Finalmente, esta vez sí utilizaremos una de nuestras postcondiciones abreviadas. Como postcondición concreta utilizaremos la versión reducida que sólo se refiere al atributo *tabla* y no al atributo *conoc* (ver comentarios luego de la especificación de *agregar* en la sección **2.2**). El consecuente a demostrar es entonces

$$\text{(Cons)} \qquad d.tabla \;=\; d_0.tabla \cup \{\, (c, v) \,\} \quad .$$

Probamos (Cons) partiendo de su lado izquierdo hasta llegar a su lado derecho, razonando como

sigue:

$$
\begin{array}{ll}
 & d.tabla \\
= & \quad \langle \text{ hipótesis (Hip0) } \rangle \\
 & \{\, i \,:\, 0 \leqslant i < d.tam \,:\, (d.claves[i], d.valores[i]) \,\} \\
= & \quad \langle \text{ hipótesis (Hip3) } \rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam + 1 \,:\, (d.claves[i], d.valores[i]) \,\} \\
= & \quad \left\langle \begin{array}{l} \text{separación de término, gracias a que } \; d_0.tam \geqslant 0 \; \text{ por hipótesis (Hip2)} \\ \text{(de lo contrario el rango sería vacío y no habría término a separar)} \end{array} \right\rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam \,:\, (d.claves[i], d.valores[i]) \,\} \,\cup\, \{\, (d.claves[d_0.tam], d.valores[d_0.tam]) \,\} \\
= & \quad \left\langle \begin{array}{l} \text{gracias a hipótesis (Hip4), tenemos que} \\ d.claves[i] = d_0.claves[i] \; \text{ para todo } i \text{ tal que } \; 0 \leqslant i < d_0.tam \,, \\ \text{y también tenemos que } \; d.claves[d_0.tam] = c \end{array} \right\rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam \,:\, (d_0.claves[i], d.valores[i]) \,\} \,\cup\, \{\, (c, d.valores[d_0.tam]) \,\} \\
= & \quad \left\langle \begin{array}{l} \text{de manera similar, gracias a hipótesis (Hip5), tenemos que} \\ d.valores[i] = d_0.valores[i] \; \text{ para todo } i \text{ tal que } \; 0 \leqslant i < d_0.tam \,, \\ \text{y también tenemos que } \; d.valores[d_0.tam] = v \end{array} \right\rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam \,:\, (d_0.valores[i], d_0.valores[i]) \,\} \,\cup\, \{\, (c, v) \,\} \\
= & \quad \langle \text{ hipótesis (Hip1) } \rangle \\
 & d_0.tabla \,\cup\, \{\, (c, v) \,\} \quad .
\end{array}
$$

Listo.

Dejamos como ejercicio (ver ejercicio **6.4-a**) el responder a la siguiente pregunta: ¿qué habría pasado si hubiésemos tratado de demostrar que la postcondición abstracta también implica a la concreta?

Eliminar, buscar y determinar existencia Dejaremos como ejercicio demostrar la consistencia bajo refinamiento de datos del resto de las operaciones de nuestro TAD *Diccionario*. Sin embargo, haremos una última demostración correspondiente a la consistencia de las postcondiciones de *eliminar*, que es la demostración más compleja de lo que resta. Sobre el único parámetro d de tipo *Diccionario* de la operación, podemos tomar como hipótésis a $d.Ac$, $d.InvA$ y $d.InvC$, de las cuales únicamente terminaremos utilizando a sus componentes

$$
\begin{array}{ll}
\text{(Hip0)} & d.tabla = \{\, i \,:\, 0 \leqslant i < d.tam \,:\, (d.claves[i], d.valores[i]) \,\} \\
\text{(Hip1)} & 0 \leqslant d.tam \leqslant d.MAX \quad ,
\end{array}
$$

y también podemos tomar como hipótesis, por ser d parámetro de entrada/salida (**in-out**), a $d_0.Ac$, $d_0.InvA$ y $d_0.InvC$, de las cuales utilizaremos a

$$
\begin{array}{ll}
\text{(Hip2)} & d_0.tabla = \{\, i \,:\, 0 \leqslant i < d_0.tam \,:\, (d_0.claves[i], d_0.valores[i]) \,\} \\
\text{(Hip3)} & (\,\forall\, i, j \,:\, 0 \leqslant i, j < d_0.tam \,:\, i \neq j \;\Rightarrow\; d_0.claves[i] \neq d_0.claves[j]\,) \quad .
\end{array}
$$

Por último, para concluir bajo estas hipótesis que la postcondición concreta implica a la postcondición abstracta, agregaremos a la primera a nuestras hipótesis para luego proceder a demostrar

la segunda. Suponemos entonces también a

$$\begin{array}{ll} \text{(Hip4)} & d.tam = d_0.tam - 1 \\ \text{(Hip5)} & (\exists\, i \,:\, 0 \leqslant i < d_0.tam \,:\, d_0.claves[i] = c \\ & \qquad \wedge\ d.claves = d_0.claves\,(i : d_0.claves[\,d_0.tam - 1\,]) \\ & \qquad \wedge\ d.valores = d_0.valores\,(i : d_0.valores[\,d_0.tam - 1\,]) \quad , \end{array}$$

y nuestro consecuente a demostrar será

$$\text{(Cons)} \qquad d.tabla \;=\; d_0.tabla \;-\; \{\,(c, d_0.tabla\, c)\,\} \quad .$$

Ahora bien, antes de proceder con (Cons), extraigamos la información de (Hip5) tomando un testigo $\hat{\imath}$ para el existencial (ver "metateorema de testigo" para existenciales como hipótesis en, por ejemplo, [GS94, sección ... ?]) de manera tal de poder disponer de tal información directamente y usemos entonces las hipótesis

$$\begin{array}{ll} \text{(Hip6)} & 0 \leqslant \hat{\imath} < d_0.tam \\ \text{(Hip7)} & d_0.claves[\hat{\imath}] = c \\ \text{(Hip8)} & d.claves = d_0.claves\,(\hat{\imath} : d_0.claves[\,d_0.tam - 1\,]) \\ \text{(Hip9)} & d.valores = d_0.valores\,(\hat{\imath} : d_0.valores[\,d_0.tam - 1\,]) \quad . \end{array}$$

Iniciemos ahora la demostración de (Cons) intentando hacer lo mismo que hicimos en el caso de las postcondiciones de *agregar*, esto es, partiendo del lado izquierdo y buscando re-expresar toda la información de d en términos de d_0 hasta llegar al lado derecho:

$$\begin{array}{ll} & d.tabla \\ = & \quad \langle\ \text{hipótesis (Hip0)}\ \rangle \\ & \{\, i \,:\, 0 \leqslant i < d.tam \,:\, (d.claves[i], d.valores[i])\,\} \\ = & \quad \left\langle \begin{array}{l} \text{necesitamos re-expresar los atributos de } d \text{ en términos de atributos de } d_0\text{:} \\ \text{sólo podemos hacerlo fácilmente para el atributo } tam \text{ utilizando (Hip4),} \\ \text{mientras para el resto de los atributos requeriremos un análisis por casos} \end{array} \right\rangle \\ & \{\, i \,:\, 0 \leqslant i < d_0.tam - 1 \,:\, (d.claves[i], d.valores[i])\,\} \quad . \end{array} \qquad \text{(I)}$$

Para continuar la transformación de expresiones sobre d en expresiones sobre d_0, podemos (debemos) aplicar las hipótesis (Hip8) e (Hip9) sobre los atributos arreglos *claves* y *valores*. Pero luego de hacer esto no podremos simplificar las expresiones resultantes salvo que logremos distinguir cuándo la indexación de los arreglos de d por medio de i coincide o no con el índice de modificación $\hat{\imath}$ utilizado en (Hip8) e (Hip9). Tal necesidad de distinción de índices puede ser resuelta separando el término $\hat{\imath}$ del resto de los elementos, pero esto tiene un problema: no necesariamente $\hat{\imath}$ se encuentra en el rango $0 \leqslant i < d_0.tam - 1$ utilizado en (I).

La hipótesis (Hip6) determina los posibles valores de $\hat{\imath}$ y muestra que no necesariamente se encuentra en el rango de (I). Sin embargo, (Hip6) también nos muestra que la pertenencia de $\hat{\imath}$ a tal rango sólo depende de su igualdad o no con el valor $d_0.tam - 1$. Continuemos entonces nuestro razonamiento separando estos dos casos, empezando por el caso en que podremos extraer $\hat{\imath}$ del rango en cuestión debido a su pertenencia a éste.

Caso $\hat{\imath} \neq d_0.tam - 1$:

$$
\begin{array}{ll}
 & \text{(I)} \\
= & \left\langle \begin{array}{l} \text{separación de término, pues (Hip6) y la hipótesis sobre } \hat{\imath} \text{ de este caso} \\ \text{garantizan la pertenencia de } \hat{\imath} \text{ al rango de generación del conjunto} \end{array} \right\rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam - 1 \;\wedge\; i \neq \hat{\imath} \,:\, (d.claves[i], d.valores[i]) \,\} \\
 & \quad \cup\ \{\, (d.claves[\hat{\imath}], d.valores[\hat{\imath}]) \,\} \\
= & \left\langle \begin{array}{l} \text{hipótesis (Hip8) e (Hip9) nos permiten ahora re-expresar a} \\ d.claves \text{ y } d.valores \text{ en términos de } d_0.claves \text{ y } d_0.valores \end{array} \right\rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam - 1 \;\wedge\; i \neq \hat{\imath} \,:\, (d_0.claves[i], d_0.valores[i]) \,\} \\
 & \quad \cup\ \{\, (d_0.claves[\, d_0.tam - 1\,], d_0.valores[\, d_0.tam - 1\,]) \,\} \\
= & \left\langle \begin{array}{l} \text{separación de término (en sentido contrario: “agregación” de término), pues} \\ \text{las hipótesis (Hip1) e (Hip4) más la hipótesis sobre } \hat{\imath} \text{ de este caso garantizan} \\ \text{que } i := d_0.tam - 1 \text{ pertenece al rango resultante del cual se separa/agrega} \end{array} \right\rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam \;\wedge\; i \neq \hat{\imath} \,:\, (d_0.claves[i], d_0.valores[i]) \,\} \quad ,
\end{array}
$$

y en este punto pareciera que no podemos hacer ninguna otra simplificación o manipulación razonable, por lo que nos detenemos acá y analizamos el otro caso con la esperanza de llegar a la misma expresión (para así cerrar el análisis separado de casos).

Caso $\hat{\imath} = d_0.tam - 1$:

$$
\begin{array}{ll}
 & \text{(I)} \\
= & \langle \text{ re-expresión del rango, gracias a la hipótesis sobre } \hat{\imath} \text{ de este caso } \rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam \;\wedge\; i \neq \hat{\imath} \,:\, (d.claves[i], d.valores[i]) \,\} \\
= & \left\langle \begin{array}{l} \text{hipótesis (Hip8) e (Hip9) nos permiten de nuevo re-expresar} \\ \text{a } d.claves \text{ y } d.valores \text{ en términos de } d_0.claves \text{ y } d_0.valores \end{array} \right\rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam \;\wedge\; i \neq \hat{\imath} \,:\, (d_0.claves[i], d_0.valores[i]) \,\} \quad ,
\end{array}
$$

con lo cual hemos exitosamente alcanzado nuestro objetivo de llegar a una conclusión común en los dos casos analizados.

Nuestra conclusión común, producto de la manipulación inicial que condujo a (I) y de los dos últimos razonamientos, es entonces

$$
d.tabla \;=\; \{\, i \,:\, 0 \leqslant i < d_0.tam \;\wedge\; i \neq \hat{\imath} \,:\, (d_0.claves[i], d_0.valores[i]) \,\} \qquad \text{(II)}
$$

y, tal como se mencionó antes, pareciera no haber otra simplificación o manipulación razonable que hacer a este expresión, por lo que nos detenemos en este punto de la manipulación del lado izquierdo de (Cons) y nos dedicamos ahora a su lado derecho con el objetivo de llegar al mismo resultado (y así completar la demostración).

Para manipular el lado derecho de (Cons), empezamos sólo con una parte de éste:

$$
\begin{array}{ll}
 & d_0.tabla \\
= & \quad \langle \text{ hipótesis (Hip2) } \rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam \,:\, (d_0.claves[i], d_0.valores[i]) \,\} \\
= & \quad \left\langle \begin{array}{l} \text{separación de término, gracias a que (Hip6) garantiza} \\ \text{que } \hat{\imath} \text{ se encuentra en el rango de generación del conjunto} \end{array} \right\rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam \;\wedge\; i \neq \hat{\imath} \,:\, (d_0.claves[i], d_0.valores[i]) \,\} \\
 & \quad \cup\; \{\, (d_0.claves[\hat{\imath}], d_0.valores[\hat{\imath}]) \,\} \\
= & \quad \left\langle \begin{array}{l} \text{por hipótesis (Hip7) tenemos, primero, que } d_0.claves[\hat{\imath}] = c \text{ y, luego,} \\ \text{combinada con (Hip2), también tenemos que } d_0.valores[\hat{\imath}] = d_0.tabla\, c \end{array} \right\rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam \;\wedge\; i \neq \hat{\imath} \,:\, (d_0.claves[i], d_0.valores[i]) \,\} \;\cup\; \{\, (c, d_0.tabla\, c) \,\} \quad .
\end{array}
$$

Con esta conclusión parcial, manipulamos ahora el lado derecho completo de (Cons) y lo mostramos igual al lado izquierdo de (Cons) como sigue:

$$
\begin{array}{ll}
 & d_0.tabla \;-\; \{\, (c, d_0.tabla\, c) \,\} \\
= & \quad \left\langle \begin{array}{l} \text{por el cálculo anterior y las siguientes propiedades de conjuntos: para} \\ X, Y \text{ y } Z \text{ cualesquiera tenemos } (X \cup Y) - Z = (X - Z) \cup (Y - Z) \\ \text{y, por lo tanto, también tenemos } (X \cup Y) - Y = X - Y \end{array} \right\rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam \;\wedge\; i \neq \hat{\imath} \,:\, (d_0.claves[i], d_0.valores[i]) \,\} \;-\; \{\, (c, d_0.tabla\, c) \,\} \\
= & \quad \left\langle \begin{array}{l} \text{por hipótesis (Hip6) e (Hip7) combinadas con (Hip3), en el primer} \\ \text{conjunto no puede haber un par con } c \text{ como primera componente} \end{array} \right\rangle \\
 & \{\, i \,:\, 0 \leqslant i < d_0.tam \;\wedge\; i \neq \hat{\imath} \,:\, (d_0.claves[i], d_0.valores[i]) \,\} \\
= & \quad \langle \text{ por (II) } \rangle \\
 & d.tabla \quad .
\end{array}
$$

¡Listo!

6.2. Otros ejemplos de consistencia

Bajo las reglas vistas en esta sección, es posible mostrar que nuestras re-especificaciones para implementación de los TADs *Cola* y *Conjunto* presentadas en la sección **5** son consistentes con las especificaciones originales presentadas en la sección **3**. Buena parte de la demostración de consistencia para estos dos ejemplos es mostrada en el apéndice **A**.

6.3. Re-especificación implícita de operaciones

⟨⟨∗∗ *OJO, sección sólo en borrador. . .* ∗∗⟩⟩

Explicar de nuevo a qué se refiere esto. . .

Simple sustitución gracias a igualdad ⟨⟨∗∗ *OJO, sección sólo en borrador. . .* ∗∗⟩⟩ *Explicar que relaciones de acoplamiento que definen a los atributos abstractos mediante igualdades, esto es, funcionales, se usan para re-especificar simplemente por sustitución. . .*

Dar un ejemplo... Para operación de agregar, precondición...

$$c \notin \{\, i \,:\, 0 \leqslant i < d.tam \,:\, d.claves[i] \,\} \quad \wedge \quad \#\{\, i \,:\, 0 \leqslant i < d.tam \,:\, d.claves[i] \,\} < d.MAX$$

y postcondición (versión breve)...

$$\{\, i \,:\, 0 \leqslant i < d.tam \,:\, (d.claves[i], d.valores[i]) \,\} = \{\, i \,:\, 0 \leqslant i < d_0.tam \,:\, (d_0.claves[i], d_0.valores[i]) \,\} \cup \{\, (c, v) \,\}$$

Notar que con frecuencia en las demostraciones esto corresponde al primer paso; esto ocurrió con la precondición de agregar... También, por ejemplo, con la postcondición de crear en la demostración correspondiente...

$$\begin{array}{l} d.MAX = m \\ \quad \wedge \quad \{\, i \,:\, 0 \leqslant i < d.tam \,:\, d.claves[i] \,\} = \emptyset \\ \quad \wedge \quad \{\, i \,:\, 0 \leqslant i < d.tam \,:\, (d.claves[i], d.valores[i]) \,\} = \emptyset \end{array}$$

Notar que otras relaciones de acoplamiento podrían complicar la cosa... Por ejemplo, en Cola sí se tienen igualdades que definen lo abstracto, pero hay que involucrar permanentemente al análisis de casos... En otros casos la relación de acoplamiento podría ser no-funcional, lo cual elimina la posibilidad de esta sencilla sustitución; en nuestros ejemplos, esto nunca ocurrirá, y en la práctica es muy poco común...

Justificación general de re-especificación implícita por sustitución *⟨⟨∗∗ OJO, sección sólo en borrador... ∗∗⟩⟩ En esta sección se demuestra por qué la sustitución recién explicada siempre funciona bien... La lectura de esta sección no es imprescindible; vale la pena principalmente para quienes gustan de razonamientos y manipulaciones lógicas, o de quienes quieran entender por qué la sustitución arriba propuesta cumple con las reglas de consistencia...*

Explicar método... A partir de las reglas se "despejará" a la pre/post-condición concreta deseada, primero separándola del resto y luego eliminando los atributos abstractos de las variables libres de la fórmula correspondiente... Esto se hará simplificando, con un solo parámetro de entrada para la precondición y con un solo parámetro de salida para la postcondición... Además, se elimina la precondición de las hipótesis de la regla de postcondiciones...

Para precondición...

$$
\begin{array}{ll}
 & [\ x.Ac \wedge x.InvA \wedge x.InvC \ \Rightarrow\ (PreA \ \Rightarrow\ PreC)\ ] \\
\equiv & \quad \langle \text{ regla probada de invariantes } \rangle \\
 & [\ x.Ac \wedge x.InvC \ \Rightarrow\ (PreA \ \Rightarrow\ PreC)\ ] \\
\equiv & \quad \langle \text{ lógica proposicional, para despejar } PreC) \rangle \\
 & [\ x.Ac \wedge x.InvC \wedge PreA \ \Rightarrow\ PreC\ ] \\
\equiv & \quad \langle \text{ lógica proposicional, para agrupar atributos abstractos } \rangle \\
 & [\ x.InvC \wedge (x.Ac \wedge PreA) \ \Rightarrow\ PreC\ ] \\
\equiv & \quad \langle \text{ cuantificación universal explícita } \rangle \\
 & (\forall\, x.a\,, x.c\,, w\ ::\ x.InvC \wedge (x.Ac \wedge PreA) \ \Rightarrow\ PreC\,) \\
\equiv & \quad \langle \text{ regla de cuantificaciones } \rangle \\
 & (\forall\, x.c\,, w\ ::\ (\exists\, x.a\ ::\ x.InvC \wedge (x.Ac \wedge PreA)\,) \ \Rightarrow\ PreC\,) \\
\equiv & \quad \langle \text{ reglas de cuantificaciones } \rangle \\
 & (\forall\, x.c\,, w\ ::\ x.InvC \wedge (\exists\, x.a\ :\ x.Ac\ :\ PreA\,) \ \Rightarrow\ PreC\,) \\
\equiv & \quad \langle \text{ re-introducción de cuantificación universal implícita } \rangle \\
 & [\ x.InvC \wedge (\exists\, x.a\ :\ x.Ac\ :\ PreA\,) \ \Rightarrow\ PreC\ ] \quad .
\end{array}
$$

y para postcondición...

$$
\begin{array}{ll}
 & [\ x.Ac \wedge x.InvA \wedge x.InvC \wedge PreA_0 \ \Rightarrow\ (PostC \ \Rightarrow\ PostA)\ ] \\
\Leftarrow & \quad \langle \text{ por lógica proposicional, para simplificar las hipótesis eliminando la precondición } \rangle \\
 & [\ x.Ac \wedge x.InvA \wedge x.InvC \ \Rightarrow\ (PostC \ \Rightarrow\ PostA)\ ] \\
\equiv & \quad \langle \text{ regla probada de invariantes } \rangle \\
 & [\ x.Ac \wedge x.InvC \ \Rightarrow\ (PostC \ \Rightarrow\ PostA)\ ] \\
\equiv & \quad \langle \text{ lógica proposicional, para despejar } PostC \rangle \\
 & [\ PostC \ \Rightarrow\ (x.Ac \wedge x.InvC \ \Rightarrow\ PostA)\ ] \\
\equiv & \quad \langle \text{ lógica proposicional, para agrupar atributos abstractos } \rangle \\
 & [\ PostC \ \Rightarrow\ (x.InvC \ \Rightarrow\ (x.Ac \ \Rightarrow\ PostA))\ ] \\
\equiv & \quad \langle \text{ cuantificación universal explícita } \rangle \\
 & (\forall\, x.a\,, x.c\,, w\ ::\ PostC \ \Rightarrow\ (x.InvC \ \Rightarrow\ (x.Ac \ \Rightarrow\ PostA))\,) \\
\equiv & \quad \langle \text{ regla de cuantificaciones } \rangle \\
 & (\forall\, x.c\,, w\ ::\ PostC \ \Rightarrow\ (\forall\, x.a\ ::\ x.InvC \ \Rightarrow\ (x.Ac \ \Rightarrow\ PostA))\,) \\
\equiv & \quad \langle \text{ reglas de cuantificaciones } \rangle \\
 & (\forall\, x.c\,, w\ ::\ PostC \ \Rightarrow\ (x.InvC \ \Rightarrow\ (\forall\, x.a\ :\ x.Ac\ :\ PostA\,))\,) \\
\equiv & \quad \langle \text{ re-introducción de cuantificación universal implícita } \rangle \\
 & [\ PostC \ \Rightarrow\ (x.InvC \ \Rightarrow\ (\forall\, x.a\ :\ x.Ac\ :\ PostA\,))\ ] \quad .
\end{array}
$$

Los "despejes" sugieren tomar para PreC a...

$$x.InvC \wedge (\exists\, x.a\ :\ x.Ac\ :\ PreA\,)$$

y para PostC a...

$$x.InvC \ \Rightarrow\ (\forall\, x.a\ :\ x.Ac\ :\ PostA\,)$$

Y, como en las pre/post-condiciones se supone implícitamente al invariante de representación, al reemplazar a InvC por true *y simplificar, tenemos que los "despejes" terminan sugiriendo tomar como par pre/post-condición PreC/PostC a...*

$$(\exists\, x.a \,:\, x.Ac \,:\, PreA\,)$$

y...

$$(\forall\, x.a \,:\, x.Ac \,:\, PostA\,)$$

Esto justifica las sustituciones antes utilizadas... Si el acoplamiento define lo abstracto con igualdades, por ejemplo, en nuestro análisis simplificado habría alguna función f tal que...

$$a \;=\; f\, c$$

Aplicado a nuestro parámetro x de arriba, es...

$$x.a \;=\; f\, x.c$$

y aplicando entonces regla de un punto a las cuantificaciones, terminamos con la siguiente sugerencia para PreC/PostC, calculada por simple sustitución sobre PreA/PostA, que automáticamente cumple con las reglas de consistencia por construcción...

$$PreA\,[\,x.a := f\, x.c\,]$$

y...

$$PostA\,[\,x.a := f\, x.c\,]$$

6.4. Ejercicios

6.4-a. *... buscar contraejemplo para la implicación de la postcondición abstracta hacia la concreta de agregar...*

6.4-b. *... completar el resto de las demostraciones de consistencia...*

6.4-c. *...?*

7. Un buen ejemplo (clásico) final: un TAD *Grafo*

...

8. Más ejemplos de TADs (un par no-clásicos)

8.1. Un TAD *RegistroEstudiantil*

... tomarlo del examen viejo en que lo usé...

8.2. Un TAD *ConexionYTarifas*

. . . tomar el ejemplo de la empresa LLÉVATELO *de uno de los exámenes viejos. . .*

8.3. Ejercicios

8.3-a. *. . . cambiar modelo abstracto de representación del TAD RegistroEstudiantil. . .*

8.3-b. *. . . cambiar modelo concreto de representación del TAD RegistroEstudiantil. . .*

8.3-c. *. . . inventar otro TAD similar a RegistroEstudiantil y pedir especificación de éste. . .*

8.3-d. *. . . inventar cambios similares para el ejemplo de* LLÉVATELO*. . .*

Referencias

[CLRS09] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 3rd edition, 2009.

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

[GS94] D. Gries and F.B. Schneider. *A Logical Approach to Discrete Math.* Texts and Monographs in Computer Science. Springer-Verlag, 1994.

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

[Lis01] B. Liskov with J. Guttag. *Program Development in Java – Abstraction, Specification, and Object-Oriented Design.* Addison-Wesley, 2001.

[Mez00] O. Meza. Introducción a la programación. Departamento de Computación y Tecnología de la Información, Universidad Simón Bolívar, disponible vía `www.ldc.usb.ve/~meza/`, 2000.

[Mit92] R. Mitchell. *Abstract Data Types and Modula-2 – A worked example of design using data abstraction.* Prentice Hall, 1992.

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

A. Otros ejemplos de consistencia

⟨⟨** *OJO, sección sólo en borrador. . .* **⟩⟩

En esta sección mostramos parte de la consistencia de los otros ejemplos que se han venido desarrollando. . .

A.0. Consistencia de la implementación del TAD *Cola*

⟨⟨** *OJO, sección sólo en borrador...* **⟩⟩

Regla de invariantes... Suponer hipótesis...

(Hip0) $inic < fin \ \vee \ nada \ \Rightarrow \ contenido = \langle\, i \,:\, inic \to fin \,:\, elems[i] \,\rangle$

(Hip1) $inic \geqslant fin \ \wedge \ \neg\, nada \ \Rightarrow$
$contenido = \langle\, i \,:\, inic \to MAX \,:\, elems[i] \,\rangle +\!\!+ \langle\, i \,:\, 0 \to fin \,:\, elems[i] \,\rangle$

(Hip2) $MAX > 0$

(Hip3) $0 \leqslant inic, fin < MAX$

(Hip4) $nada \ \Rightarrow \ inic = fin$,

y demostrar consecuentes...

(Cons0) $MAX > 0$

(Cons1) $\#\, contenido \leqslant MAX$.

Propiedad de secuencias que será usada repetidamente...

$$\#\,\langle\, x \,:\, a \to b \,:\, E \,\rangle \;=\; b - a \qquad \text{si } \ a \leqslant b$$

La parte interesante es (Cons1)... Acoplamiento pide análisis por casos... Primer caso, suponemos $inic < fin \ \vee \ nada$ *...*

$$\begin{array}{ll}
 & \#\, contenido \\
= & \quad \langle \text{ suposición de primer caso e (Hip0) } \rangle \\
 & \#\,\langle\, i \,:\, inic \to fin \,:\, elems[i] \,\rangle \\
= & \quad \left\langle \begin{array}{l} \text{propiedad de secuencias antes enunciada,} \\ \text{condición requerida } \ inic \leqslant fin \ \text{ demostrada luego} \end{array} \right\rangle \\
 & fin - inic \\
< & \quad \langle \text{ por (Hip3) tenemos } \ fin < MAX \ \rangle \\
 & MAX - inic \\
\leqslant & \quad \langle \text{ por (Hip3) tenemos } \ -inic \leqslant 0 \ \rangle \\
 & MAX
\end{array}$$

lo cual demuestra $\#\, contenido < MAX$ *, lo cual implica* $\#\, contenido \leqslant MAX$ *en este primer caso... La condición que se ofreció demostrar falta...*

$$\begin{array}{ll}
 & inic \leqslant fin \\
\equiv & \quad \langle \text{ relaciones aritméticas } \rangle \\
 & inic < fin \ \vee \ inic = fin \\
\Leftarrow & \quad \langle \text{ hipótesis (Hip4) } \rangle \\
 & inic < fin \ \vee \ nada \\
\equiv & \quad \langle \text{ hipótesis de primer caso } \rangle \\
 & \text{true}
\end{array}$$

Segundo caso, suponemos $inic \geqslant fin \ \wedge \ \neg\, nada \ldots$

$$
\begin{array}{ll}
 & \#\, contenido \\
= & \quad \langle \text{ suposición de segundo caso e (Hip1) } \rangle \\
 & \#\,(\,\langle\, i \,:\, inic \rightarrow MAX \,:\, elems[i] \,\rangle +\!\!+ \langle\, i \,:\, 0 \rightarrow fin \,:\, elems[i] \,\rangle\,) \\
= & \quad \langle \text{ propiedad de secuencias } \rangle \\
 & \#\,\langle\, i \,:\, inic \rightarrow MAX \,:\, elems[i] \,\rangle \,+\, \#\,\langle\, i \,:\, 0 \rightarrow fin \,:\, elems[i] \,\rangle \\
= & \quad \left\langle \begin{array}{l} \text{propiedad de secuencias antes enunciada, condiciones} \\ \text{requeridas } inic \leqslant MAX \text{ y } 0 \leqslant fin \text{ se tienen por (Hip3)} \end{array} \right\rangle \\
 & MAX - inic + fin \\
\leqslant & \quad \langle \text{ por hipótesis de segundo caso tenemos } fin \leqslant inic \ \rangle \\
 & MAX - inic + inic \\
= & \quad \langle \text{ aritmética } \rangle \\
 & MAX
\end{array}
$$

lo cual demuestra $\#\, contenido \leqslant MAX$ *también en el segundo caso...*

Vemos sólo operación encolar...

Precondición de encolar... Suponemos hipótesis c.Ac, c.InvA y c.InvC... Listamos sólo las que se usarán...

$$
\begin{array}{lll}
\text{(Hip0)} & c.inic \geqslant c.fin \ \wedge \ \neg\, c.nada \ \Rightarrow & \\
 & \qquad c.contenido = \langle\, i \,:\, c.inic \rightarrow c.MAX \,:\, c.elems[i] \,\rangle +\!\!+ & \\
 & \qquad\qquad \langle\, i \,:\, 0 \rightarrow c.fin \,:\, c.elems[i] \,\rangle & \\
\text{(Hip1)} & 0 \leqslant c.inic, c.fin < c.MAX \quad , &
\end{array}
$$

Se debe demostrar...

$$\#\, c.contenido < c.MAX \ \Rightarrow \ c.inic \neq c.fin \ \vee \ c.nada$$

Para habilitar el uso de los casos de la relación de acoplamiento, es mejor intentar demostrar el contrarrecíproco...

$$c.inic = c.fin \ \wedge \ \neg\, c.nada \ \Rightarrow \ \#\, c.contenido \geqslant c.MAX$$

Suponemos entonces nueva hipótesis...

$$
\begin{array}{ll}
\text{(Hip2)} & c.inic = c.fin \\
\text{(Hip3)} & \neg\, c.nada
\end{array}
$$

y demostramos consecuente...

$$\text{(Cons)} \qquad \#\, c.contenido \geqslant c.MAX \quad .$$

Notar que este consecuente es, gracias a hipótesis del invariante abstracto, no listada antes explícitamente, equivalente a igualdad... De hecho, se demostrará igualdad, pero como ésta implica a

lo indicado en (Cons) no usamos la hipótesis nombrada, por innecesaria... Demostramos...

$$
\begin{array}{ll}
 & \#\, c.contenido \\
= & \quad \langle \text{ por (Hip0), (Hip2) e (Hip3) } \rangle \\
 & \#\,(\,\langle\, i \,:\, c.inic \to c.MAX \,:\, c.elems[i] \,\rangle \mathbin{+\!\!+} \langle\, i \,:\, 0 \to c.fin \,:\, c.elems[i] \,\rangle\,) \\
= & \quad \langle \text{ propiedad de secuencias } \rangle \\
 & \#\,\langle\, i \,:\, c.inic \to c.MAX \,:\, c.elems[i] \,\rangle \,+\, \#\,\langle\, i \,:\, 0 \to c.fin \,:\, c.elems[i] \,\rangle \\
= & \quad \left\langle \begin{array}{l} \text{propiedad de secuencias antes enunciada, condiciones} \\ \text{requeridas } c.inic \leqslant c.MAX \text{ y } 0 \leqslant c.fin \text{ se tienen por (Hip1)} \end{array} \right\rangle \\
 & c.MAX - c.inic + c.fin \\
= & \quad \langle \text{ por (Hip2) y aritmética } \rangle \\
 & c.MAX
\end{array}
$$

Quedó demostrado (Cons)...

Postcondición de encolar... Suponemos de nuevo hipótesis c.Ac, c.InvA y c.InvC, pero también $c_0.Ac$, $c_0.InvA$ y $c_0.InvC$, y además $PreA_0$... Notar que por lo demostrado para precondiciones también podemos entonces contar con hipótesis $PreC_0$... Listamos sólo lo que se terminará utilizando... De la información de c tenemos...

$$
\begin{array}{ll}
\text{(Hip0)} & c.inic < c.fin \;\vee\; c.nada \;\Rightarrow \\
 & \qquad c.contenido = \langle\, i \,:\, c.inic \to c.fin \,:\, c.elems[i] \,\rangle \\
\text{(Hip1)} & c.inic \geqslant c.fin \;\wedge\; \neg\, c.nada \;\Rightarrow \\
 & \qquad c.contenido = \langle\, i \,:\, c.inic \to c.MAX \,:\, c.elems[i] \,\rangle \mathbin{+\!\!+} \\
 & \qquad\qquad \langle\, i \,:\, 0 \to c.fin \,:\, c.elems[i] \,\rangle \\
\text{(Hip2)} & 0 \leqslant c.inic, c.fin < c.MAX
\end{array}
$$

De c_0 utilizaremos lo mismo, más la parte final del invariante de representación concreto...

$$
\begin{array}{ll}
\text{(Hip3)} & c_0.inic < c_0.fin \;\vee\; c_0.nada \;\Rightarrow \\
 & \qquad c_0.contenido = \langle\, i \,:\, c_0.inic \to c_0.fin \,:\, c_0.elems[i] \,\rangle \\
\text{(Hip4)} & c_0.inic \geqslant c_0.fin \;\wedge\; \neg\, c_0.nada \;\Rightarrow \\
 & \qquad c_0.contenido = \langle\, i \,:\, c_0.inic \to c_0.MAX \,:\, c_0.elems[i] \,\rangle \mathbin{+\!\!+} \\
 & \qquad\qquad \langle\, i \,:\, 0 \to c_0.fin \,:\, c_0.elems[i] \,\rangle \\
\text{(Hip5)} & 0 \leqslant c_0.inic, c_0.fin < c_0.MAX \\
\text{(Hip6)} & c_0.nada \;\Rightarrow\; c_0.inic = c_0.fin
\end{array}
$$

Nótese que $c_0.MAX$ es siempre trivialmente igual a $c.MAX$ por tratarse de un atributo constante... De la hipótesis $PreA_0$, como referido antes, gracias a la información sobre c_0 y la consistencia ya demostrada entre postcondiciones, obtenemos también a $PreC_0$, que listamos entonces ahora como hipótesis...

$$
\begin{array}{ll}
\text{(Hip7)} & c_0.inic \neq c_0.fin \;\vee\; c_0.nada
\end{array}
$$

Por último, suponemos PostC para luego demostrar PostA... Esto nos da nuestras últimas hipó-

tesis...

(Hip8) $c.inic = c_0.inic$
(Hip9) $c.fin = (c_0.fin + 1) \bmod c.MAX$
(Hip10) $c.elems = c_0.elems\,(c_0.fin : x)$
(Hip11) $\neg\, c.nada$

Debemos entonces demostrar el consecuente...

(Cons) $c.contenido = c_0.contenido +\!\!+ \langle x \rangle$

Trabajar con este consecuente exige distinguir los casos correspondientes a (Hip0) vs. (Hip1) para estudiar a $c.contenido$ *, y en cada uno de estos casos distinguir las posibilidades correspondientes a (Hip3) vs. (Hip4) de* $c_0.contenido$ *... Cuando se esté haciendo tal análisis de casos, la aritmética modular de (Hip9) se estudiará también por casos... La hipótesis (Hip5) da una cota superior para* $c_0.fin$ *que permite separar los casos correspondientes a (Hip9) como sigue (recordar que* $c.MAX$ *y* $c_0.MAX$ *coinciden)...*

(Hip12) $c_0.fin < c.MAX - 1 \;\Rightarrow\; c.fin = c_0.fin + 1$
(Hip13) $c_0.fin = c.MAX - 1 \;\Rightarrow\; c.fin = 0$

Empezamos finalmente a demostrar (Cons)... Para manipular a $c.contenido$ *, debemos distinguir los casos dados por (Hip0) e (Hip1)... Gracias a (Hip11), tenemos que* $c.nada$ *es falso, por lo que los dos casos se basan sólo en la relación entre* $c.inic$ *y* $c.fin$ *...*

Vamos con caso (A): $c.inic < c.fin$ *... En este caso (al igual que en los siguientes casos que vendrán), debemos analizar las posibilidades para* $c_0.contenido$ *según (Hip3) e (Hip4), y analizar las posibilidades para* $c.fin$ *según (Hip12) e (Hip13)... Empezamos mostrando que en este caso (A) no puede ocurrir el caso correspondiente a (Hip13)...*

$c.inic < c.fin \;\wedge\; c_0.fin = c.MAX - 1$
$\Rightarrow$ $\langle$ por (Hip13) $\rangle$
$c.inic < c.fin \;\wedge\; c.fin = 0$
$\Rightarrow$ $\langle$ Leibniz $\rangle$
$c.inic < 0$
$\equiv$ $\langle$ por (Hip2) $\rangle$
false

Por lo tanto...

$c.inic < c.fin$
$\Rightarrow$ $\langle$ por reducción al absurdo del cálculo anterior $\rangle$
$c_0.fin \neq c.MAX - 1$
$\equiv$ $\langle$ por (Hip5) y ser MAX atributo constante $\rangle$
$c_0.fin < c.MAX - 1$ (I)

Vemos entonces que $c.fin$ *sólo puede estar definido en este caso (A) según lo estipulado en (Hip12)... En cuanto a los posibles casos según (Hip3) y (Hip4), tenemos...*

$$
\begin{array}{ll}
 & c.inic < c.fin \\
\equiv & \quad \langle \text{ por (Hip8), y por (Hip12) con (I) } \rangle \\
 & c_0.inic < c_0.fin + 1 \\
\equiv & \quad \langle \text{ relaciones aritméticas } \rangle \\
 & c_0.inic \leqslant c_0.fin \qquad \text{(II)} \\
\equiv & \quad \langle \text{ por (Hip7) } \rangle \\
 & c_0.inic \leqslant c_0.fin \ \wedge \ (c_0.inic \neq c_0.fin \ \vee \ c_0.nada) \\
\equiv & \quad \langle \text{ distributividad de conjunción sobre disyunción } \rangle \\
 & (c_0.inic \leqslant c_0.fin \ \wedge \ c_0.inic \neq c_0.fin) \ \vee \ (c_0.inic \leqslant c_0.fin \ \wedge \ c_0.nada) \\
\Rightarrow & \quad \langle \text{ relaciones aritméticas, lógica proposicional } \rangle \\
 & c_0.inic < c_0.fin \ \vee \ c_0.nada \qquad \text{(III)}
\end{array}
$$

Esto último, (III), determina el caso para c_0*... Las tres consecuencias (I), (II) y (III) de la condición del caso (A) han sido marcadas para posterior uso... Finalizamos la demostración de (Cons) en este caso (A)...*

$$
\begin{array}{ll}
 & c.contenido \\
= & \quad \langle \text{ por (Hip0) con hipótesis de caso (A) } \rangle \\
 & \langle\, i \,:\, c.inic \rightarrow c.fin \,:\, c.elems[i] \,\rangle \\
= & \quad \langle \text{ por (Hip8), y por (Hip12) con (I) } \rangle \\
 & \langle\, i \,:\, c_0.inic \rightarrow c_0.fin + 1 \,:\, c.elems[i] \,\rangle \\
= & \quad \langle \text{ separación de término, gracias a que (II) garantiza que el rango no es vacío } \rangle \\
 & \langle\, i \,:\, c_0.inic \rightarrow c_0.fin \,:\, c.elems[i] \,\rangle \mathbin{+\!\!+} \langle\, c.elems[c_0.fin] \,\rangle \\
= & \quad \langle \text{ por (Hip10) podemos re-expresar a } c.elems \text{ en términos de } c_0.elems \ \rangle \\
 & \langle\, i \,:\, c_0.inic \rightarrow c_0.fin \,:\, c_0.elems[i] \,\rangle \mathbin{+\!\!+} \langle x \rangle \\
= & \quad \langle \text{ por (Hip3) con (III) } \rangle \\
 & c_0.contenido \mathbin{+\!\!+} \langle\, x \,\rangle
\end{array}
$$

Vamos ahora con caso (B): $c.inic \geqslant c.fin$ *... De nuevo debemos analizar las posibilidades para* $c_0.contenido$ *según (Hip3) e (Hip4), y también las posibilidades para* $c.fin$ *según (Hip12) e (Hip13)... Atacamos primero lo último, ya que esto determina lo primero...*

Vamos entonces con sub-caso (B.0): $c_0.fin < c.MAX - 1$ *... Discernimos ahora en qué caso*

estamos para $c_0.\mathit{contenido}\ldots$

$$
\begin{array}{ll}
 & c.\mathit{inic} \geqslant c.\mathit{fin} \\
\equiv & \quad \langle \text{ por (Hip8), y por (Hip12) con hipótesis de caso (B.0) } \rangle \\
 & c_0.\mathit{inic} \geqslant c_0.\mathit{fin} + 1 \\
\equiv & \quad \langle \text{ relaciones aritméticas } \rangle \\
 & c_0.\mathit{inic} > c_0.\mathit{fin} \qquad \text{(IV)} \\
\equiv & \quad \langle \text{ relaciones aritméticas } \rangle \\
 & c_0.\mathit{inic} \geqslant c_0.\mathit{fin} \ \wedge \ c_0.\mathit{inic} \neq c_0.\mathit{fin} \\
\Rightarrow & \quad \langle \text{ por contrarrecíproco de (Hip6) } \rangle \\
 & c_0.\mathit{inic} \geqslant c_0.\mathit{fin} \ \wedge \ \neg\, c_0.\mathit{nada} \qquad \text{(V)}
\end{array}
$$

Esto determina el caso para $c_0\ldots$ *Por lo tanto...*

$$
\begin{array}{ll}
 & c.\mathit{contenido} \\
= & \quad \langle \text{ por (Hip1) con hipótesis de caso (B) } \rangle \\
 & \langle\, i \,:\, c.\mathit{inic} \to c.\mathit{MAX} \,:\, c.\mathit{elems}[i] \,\rangle +\!\!+ \langle\, i \,:\, 0 \to c.\mathit{fin} \,:\, c.\mathit{elems}[i] \,\rangle \\
= & \quad \langle \text{ por (Hip8), y por (Hip12) con hipótesis de caso (B.0) } \rangle \\
 & \langle\, i \,:\, c_0.\mathit{inic} \to c.\mathit{MAX} \,:\, c.\mathit{elems}[i] \,\rangle +\!\!+ \langle\, i \,:\, 0 \to c_0.\mathit{fin} + 1 \,:\, c.\mathit{elems}[i] \,\rangle \\
= & \quad \langle \text{ separación de término, rango no-vacío gracias a } 0 \leqslant c_0.\mathit{fin} \text{ de (Hip5) } \rangle \\
 & \langle\, i \,:\, c_0.\mathit{inic} \to c.\mathit{MAX} \,:\, c.\mathit{elems}[i] \,\rangle +\!\!+ \langle\, i \,:\, 0 \to c_0.\mathit{fin} \,:\, c.\mathit{elems}[i] \,\rangle +\!\!+ \langle\, c.\mathit{elems}[c_0.\mathit{fin}] \,\rangle \\
= & \quad \left\langle \begin{array}{l} \text{por (Hip10) podemos re-expresar a } c.\mathit{elems} \text{ en términos de } c_0.\mathit{elems}\,, \\ \text{gracias a (IV) se tiene que } c_0.\mathit{fin} \text{ no ocurre en el primer rango} \end{array} \right\rangle \\
 & \langle\, i \,:\, c_0.\mathit{inic} \to c.\mathit{MAX} \,:\, c_0.\mathit{elems}[i] \,\rangle +\!\!+ \langle\, i \,:\, 0 \to c_0.\mathit{fin} \,:\, c_0.\mathit{elems}[i] \,\rangle +\!\!+ \langle x \rangle \\
= & \quad \langle \text{ por (Hip4) con (V), recordar que } \mathit{MAX} \text{ es atributo constante } \rangle \\
 & c_0.\mathit{contenido} +\!\!+ \langle\, x \,\rangle
\end{array}
$$

Finalizamos con sub-caso (B.1): $c_0.\mathit{fin} = c.\mathit{MAX} - 1\ldots$ *De nuevo, lo primero será discernir qué caso define a* $c_0.\mathit{contenido}\ldots$

$$
\begin{array}{ll}
 & c_0.\mathit{fin} = c.\mathit{MAX} - 1 \\
\Rightarrow & \quad \left\langle \begin{array}{l} \text{por (Hip5) se tiene que } c_0.\mathit{inic} \leqslant c_0.\mathit{MAX} - 1\,, \\ \text{recordar que } \mathit{MAX} \text{ es atributo constante, Leibniz} \end{array} \right\rangle \\
 & c_0.\mathit{inic} \leqslant c_0.\mathit{fin} \qquad \text{(VI)} \\
\Rightarrow & \quad \langle \text{ por el mismo cálculo entre (II) y (III) del caso (A) } \rangle \\
 & c_0.\mathit{inic} < c_0.\mathit{fin} \ \vee \ c_0.\mathit{nada} \qquad \text{(VII)}
\end{array}
$$

Esto determina el caso para c_0 y por tanto...

$$
\begin{array}{ll}
 & c.contenido \\
= & \quad \langle \text{ por (Hip1) e hipótesis de caso (B) } \rangle \\
 & \langle\, i \,:\, c.inic \rightarrow c.MAX \,:\, c.elems[i] \,\rangle \mathbin{+\!\!+} \langle\, i \,:\, 0 \rightarrow c.fin \,:\, c.elems[i] \,\rangle \\
= & \quad \langle \text{ por (Hip8), y por (Hip13) con hipóetsis de caso (B.1) } \rangle \\
 & \langle\, i \,:\, c_0.inic \rightarrow c.MAX \,:\, c.elems[i] \,\rangle \mathbin{+\!\!+} \langle\, i \,:\, 0 \rightarrow 0 \,:\, c.elems[i] \,\rangle \\
= & \quad \langle \text{ por hipótesis de caso (B.1) y concatenación con secuencia vacía } \rangle \\
 & \langle\, i \,:\, c_0.inic \rightarrow c_0.fin + 1 \,:\, c.elems[i] \,\rangle \\
= & \quad \langle \text{ separación de término, gracias a que (VI) garantiza que el rango no es vacío } \rangle \\
 & \langle\, i \,:\, c_0.inic \rightarrow c_0.fin \,:\, c.elems[i] \,\rangle \mathbin{+\!\!+} \langle\, c.elems[c_0.fin] \,\rangle \\
= & \quad \langle \text{ por (Hip10) podemos re-expresar a } c.elems \text{ en términos de } c_0.elems \ \rangle \\
 & \langle\, i \,:\, c_0.inic \rightarrow c_0.fin \,:\, c_0.elems[i] \,\rangle \mathbin{+\!\!+} \langle x \rangle \\
= & \quad \langle \text{ por (Hip3) con (VII) } \rangle \\
 & c_0.contenido \mathbin{+\!\!+} \langle\, x \,\rangle
\end{array}
$$

Resto de la consistencia queda como ejercicio...

A.1. Consistencia de la implementación del TAD *Conjunto*

*⟨⟨** OJO, sección sólo en borrador... **⟩⟩*

En lugar de demostrar toda la consistencia según todas las reglas vistas, demostramos un par de lemas que luego ayudarían a demostrar todo lo demás... Esto ilustra las bondades de seleccionar un par de buenos lemas que puedan ser re-utilizados con frecuencia...

Para un objeto x cualquiera de tipo Conjunto tomamos las hipótesis $x.Ac$, $x.InvA$ e $x.InvC$ y demostramos los siguientes lemas, que relacionan expresiones sobre atributos del modelo abstracto con expresiones sobre atributos del modelo concreto...

(Lema0) $\quad \#\, x.contenido \;=\; x.tam$

(Lema1) $\quad y \in x.contenido \;\equiv\; (\,\exists\, i \,:\, 0 \leqslant i < x.tam \,:\, x.elems[i] = y\,)$

Para las demostraciones, de todas las hipótesis listamos ahora sólo lo que terminaremos usando explícitamente...

(Hip0) $\quad x.contenido \;=\; \{\, i \,:\, 0 \leqslant i < x.tam \,:\, x.elems[i] \,\}$

(Hip1) $\quad (\,\forall\, i, j \,:\, 0 \leqslant i, j < x.tam \,:\, i \neq j \;\Rightarrow\; x.elems[i] \neq x.elems[j]\,)$

Procedemos ahora a demostrar los lemas... Empezamos con (Lema0)...

$$
\begin{array}{ll}
 & \#\, x.contenido \\
= & \quad \langle \text{ hipótesis (Hip0) } \rangle \\
 & \#\, \{\, i \,:\, 0 \leqslant i < x.tam \,:\, x.elems[i] \,\} \\
= & \quad \left\langle \begin{array}{l} \text{hipótesis (Hip1) garantiza que todos los elementos generados} \\ \text{para el conjunto son diferentes y, por lo tanto, la cantidad de} \\ \text{ellos es exactamente igual a la cantidad de valores del rango} \end{array} \right\rangle \\
 & x.tam
\end{array}
$$

Continuamos con (Lema1)...

$$
\begin{array}{ll}
 & y \in x.contenido \\
\equiv & \quad \langle \text{ hipótesis (Hip0) } \rangle \\
 & y \in \{\, i : 0 \leqslant i < x.tam : x.elems[i] \,\} \\
\equiv & \quad \langle \text{ pertenencia a conjuntos definidos por comprensión } \rangle \\
 & (\exists\, i : 0 \leqslant i < x.tam : x.elems[i] = y) \quad ,
\end{array}
$$

Listos los lemas...

Dar algunos ejemplos de cómo los lemas ayudan... Para precondición de agregar, ambos lemas con $x, y := c, x$ *; para postcondiciones de extraer y pertenece, (Lema1) con* $x, y := c, x$ *; para precondiciones de unir, intersectar y restar, (Lema0) con* $x := c0$ *y* $x := c1$ *... Incluso, para postcondiciones de crear y vacio, y también para precondición de extraer, (Lema0) con* $x := c$ *más la equivalencia de teoría de conjuntos entre* $A = \emptyset$ *y* $\#A = 0$ *para cualquier conjunto* A*...*

Resto de la consistencia queda como ejercicio...

J.N. Ravelo / Diciembre 2011 (corregido y aumentado de versiones anteriores de febrero 2009 y febrero 2010)