LA IGUALDAD EN TIPOS ABSTRACTOS DE DATOS

Rodrigo CARDOSO R.
Universidad de los Andes
Departamento de Sistemas
Apartado Aéreo 4976
Bogotá, D.E.
COLOMBIA

Resumen: Se presenta una definición de igualdad en tipos abstractos de datos (TADs) que soluciona dificultades prácticas en la utilización de la metodología de diseño e implementación que se expone en [Car86].

La nueva definición hace más fácil para el diseñador el decidir que el TAD construído refleja de manera adecuada el universo que pretende modelar. El poner en claro la definición de igualdad permite además que la prueba mecánica de teoremas sobre el TAD (v.gr.adecuación al modelo, verificación de implementaciones) se pueda realizar con buenos resultados prácticos.

Palabras claves: Tipos abstractos de datos, relaciones de igualdad.

****

0. Introducción.

En [Car86] se describe una metodología de diseño de tipos abstractos de datos (TADs) guiada por el conocimiento de un universo de objetos que se quiere modelar. La labor de diseño va de la mano de un proceso de ajuste entre la abstracción y la realidad, la metodología pretende explicitar criterios par juzgar la bondad de este ajuste.

Al llevar a la práctica aquellas ideas (construyendo herramientas para diseñar e implementar TADs) se descubre una dificultad asociada con la forma de decidir en qué caso dos términos de un TAD son iguales. En **3.** se expone un problema típico con la metodología anterior y se muestra como las reformas propuestas solucionan la dificultad.

En **1.** se da un resumen de lo necesario de [Car86] para entender este artículo. Las reformas anunciadas se incluyen en **2.**

1. Conceptos básicos.

$\langle Y,F \rangle$ es un **tipo abstracto de datos** (TAD) cuando Y es un conjunto (construíble) de símbolos y F un conjunto finito de operaciones sobre estos símbolos y tal vez otros TADs ya entendidos. El conjunto Y es el **tipo de interés** del TAD, y sobre él se cuenta con un orden bien fundado $.<_Y.$. Mientras no haya confusión Y denotará el TAD $\langle Y,F \rangle$. Cuando Y es un conjunto de objetos explícitos y F un conjunto de operaciones calculables sobre ellos se dice que Y es euna **estructura computacional** (EC).

Una de las reformas al la metodología de [Car86] es exigir además una relación de igualdad $.=_Y.$ definida sobre Y, no necesariamente la correspondiente al orden $.<_Y.$, que permite una revisión más efectiva de la adecuación del TAD al universo de objetos que se pretende modelar, a la vez que hace posible la demostración de teoremas del TAD que no son derivables con la sola aplicación de reglas de reescritura.

El conjunto de operaciones F es la unión de los conjuntos de operaciones **iniciales, constructoras, simplificadoras, y analizadoras** (I, C, S, A respectivamente). Es cómodo llamar $FG = I \cup C$ el conjunto de operaciones **generadoras** y $FS = S \cup A$ el conjunto de operaciones **selectoras**

El tipo de interés Y es el conjunto de términos sin variables generados por FG. Cada f∈FG se define por autodenotación (v.gr. para c∈C con funcionalidad c: Y x X ---> Y, el valor de c(y,x) es justamente la cadena de símbolos 'c(y,x)').

Las funciones en FS son definidas inductivamente sobre la complejidad de la construcción del tipo de interés. Las definiciones son dadas mediante un conjunto finito de axiomas $\mathcal{A}$. Cada axioma es de la forma

<lado derecho> = <lado izquierdo>

El <lado derecho> es un término cuyo símbolo de operación más externo es el de una selectora, y para cada selectora hay suficientes axiomas para garantizar una definición recursiva de la misma, con lo que se asegura la completitud suficiente del TAD. Obsérvese que no se dan axiomas para operaciones generadoras.

2. La igualdad en un TAD.

En [Car86] se definió una relación de "equivalencia semántica" de elementos de un TAD Y que presenta problemas al tratar de usarse para mostrar ciertos teoremas del TAD [Bar87]. A continuación se da una nueva definición igualmente plausible y factible de utilizar.

Como hipótesis adicional se requiere de cada TAD Xi conocido que sirva como parámetro a Y el tener definida una relación de igualdad decidible $=_{Xi}$.

Notación: Para $y = (y_1,...,y_{m-1}) \in Y^{m-1}$, $y_0 \in Y$, $1 \le j \le m$:

$$[y,y_0]_j = (y_1,...,y_{j-1},y_0,y_{j+1},...,y_m).$$

2.1 Definición

Sean t1, t2 ∈ F_* (términos sin variables),

y1, y2 ∈ Y,

f ∈ FS, con funcionalidad f: Y^m x $dom_X(f)$ ----> Z ($dom_X(f)$ es un producto cartesiano X1 x ... x Xr de TADs conocidos, Z = Y si f ∈ S, y Z = Xi si f ∈ A),

K un subconjunto de FS.

a) t1 →* t2 ssi t1 reduce a t2 por aplicación de axiomas

t1 =* t2 ssi Existe t tal que t1 →* t y t2 →* t.

b) $y1 =_f y2$ ssi Para todo $\mathbf{y} \in Y^{m-1}$, $x \in dom_X(f)$, $1 \le j \le m$:

$$f([\mathbf{y},y_1]_j,x) =^* f([\mathbf{y},y_2]_j,x), \text{ si } f \in S,$$
$$f([\mathbf{y},y_1]_j,x) =_{Xi} f([\mathbf{y},y_2]_j,x), \text{ si } f \in A.$$

c) $y1 =_K y2$ ssi Para todo $f \in K$: $y1 =_f y2$.

d) $y1 \equiv y2$ ssi $y1$ es **fuertemente equivalente** a $y2$
ssi $y1 =_{FS} y2$.

e) $y1 =_{K,f} y2$ ssi Para todo $\mathbf{y} \in Y^{m-1}$, $x \in dom_X(f)$, $1 \le j \le m$.

$$f([\mathbf{y},y_1]_j,x) =_K f([\mathbf{y},y_2]_j,x).$$

f) $.=_K.$ es una **igualdad** ssi

Para todo $f \in FS$: $y1 =_K y2 \Rightarrow y1 =_{K,f} y2$.

En este caso se dice que K es un **caracterizador**.

□

Con la anterior definición es concebible que en Y se puedan considerar varias relaciones de igualdad. Además de ser relación de equivalencia, una igualdad satisface una propiedad de sustitución entre términos relacionados, i.e. si dos términos son iguales, cualquier expresión que dependa de uno de ellos evalúa al mismo valor si se utiliza el otro en su lugar.

En [Car86] solo se aceptaba una igualdad, la aquí llamada equivalencia fuerte $.\equiv.$. Claramente vale para cualquier K, subconjunto de FS:

$$y1 \equiv y2 \Rightarrow y1 =_K y2,$$

y en particular para los caracterizadores. El concepto es aquí más general y exige del diseñador una última decisión para completar la definición del TAD Y: escoger un caracterizador que defina la igualdad $.=_Y.$.

La elección de un caracterizador no es tan difícil en la práctica, porque el diseñador debería tener claro qué propiedades son las que quiere observar para distinguir un objeto de otro en el universo que desea modelar. En últimas puede escoger siempre a FS como caracterizador, y la igualdad asociada es la equivalencia fuerte.

El ideal es elegir un caracterizador K tal que $y1 =_K y2$ si y solo si y1, y2 nombran un mismo objeto del mundo. Por supuesto esta es una propiedad apenas intuitivamente constatable, y en el mejor de los casos la demostración de una tal adecuación depende de una teoría para el mundo cuyo formalismo es mucho más general que el de los TADs, de modo que hay que usar conocimiento adicional para probar que ésto vale. A continuación se ilustra mediante un ejemplo cómo puede funcionar este método.

3. Un ejemplo.

El siguiente TAD modela los conjuntos finito de elementos de un TAD X.

Tipo Set [X]

```
* empty:              ----> Set
* insert:  Set x X    ----> Set
  delete:  Set x X    ----> Set
  has:     Set x X    ----> bool
  union:   Set x Set  ----> Set
```

Axiomas

(1) delete(empty,x) = empty

(2) delete(insert(s,x1),x2) = **if** $x1 =_X x2$ **then** delete(s,x2)
else insert(delete(s,x2),x1)
fi

(3) has(empty,x) = **false**

(4) has(insert(s,x1),x2) = **if** $x1 =_X x2$ **then true**
else has(s,x2)
fi

(5) union(empty,s) = s

(6) union(insert(s1,x),s2) = insert(union(s1,s2),x)

finTipo

Para $a,b \in X$, el conjunto $\{a,b\}$ tiene dos nombres en Set[X]:

$$s1 = insert(insert(empty,a),b)$$
$$s2 = insert(insert(empty,b),a).$$

En [Car86] se sugirió que era posible demostrar que $s1 \equiv s2$ Los términos a cada lado de la equivalencia son irreducibles, porque no hay axiomas para generadoras. Si se intenta una demostración que chequee si s1 y s2 se portan en la misma forma para cada selectora, tal intento resulta fallido [Bar87].

Por ejemplo, debería suceder, para cualquier $c \in X$, que:

$$delete(s1,c) \equiv delete(s2,c).$$

Ahora bien.

$$\begin{aligned} delete(s1,c) &= delete(insert(insert(empty,a),b), c) \\ &= \textbf{if } b =_X c \textbf{ then } delete(\ insert(empty,a),c) \\ &\qquad \textbf{else } insert(delete(insert(empty,a),c),b) \\ &\quad \textbf{fi} \end{aligned}$$

Y después de reescribir:

$$\begin{aligned} delete(s1,c) = \ & \textbf{if } \ b =_X c \ \wedge \ a =_X c \ \textbf{then } empty \\ & \textbf{elsf } b =_X c \ \wedge \ a \neq_X c \ \textbf{then } insert(empty,a) \\ & \textbf{elsf } b \neq_X c \ \wedge \ a =_X c \ \textbf{then } insert(empty,b) \\ & \textbf{elsf } b \neq_X c \ \wedge \ a \neq_X c \ \textbf{then } insert(insert(empty,a),b) \\ & \textbf{fi} \end{aligned}$$

Análogamente:

$$\begin{aligned} delete(s2,c) = \ & \textbf{if } \ b =_X c \ \wedge \ a =_X c \ \textbf{then } empty \\ & \textbf{elsf } b =_X c \ \wedge \ a \neq_X c \ \textbf{then } insert(empty,a) \\ & \textbf{elsf } b \neq_X c \ \wedge \ a =_X c \ \textbf{then } insert(empty,b) \\ & \textbf{elsf } b \neq_X c \ \wedge \ a \neq_X c \ \textbf{then } insert(insert(empty,b),a) \\ & \textbf{fi} \end{aligned}$$

Y para demostrar que las dos expresiones reducen a lo mismo en todos los casos se requiere que $s1 \equiv s2$ (obsérvese el caso $b \neq_X c \ \wedge \ a \neq_X c$).

Se mostrará ahora que K = {has} es un caracterizador para Set[X].

Hay que probar la siguiente proposición:

3.1 Proposición:

$s1 =_{has} s2 \Rightarrow$ a) Para todo $x \in X$: $del(s1,x) =_{has} del(s2,x)$

b) Para todo $s \in Set$: $union(s1,s) =_{has} union(s2,s)$ y $union(s,s1) =_{has} union(s,s2)$.

La demostración puede ahora llevarse a cabo utilizando las definiciones y reglas de reescritura adecuadas. Hay otra manera de probar el resultado, apoyándose en el conocimiento que se puede tener del modelo pretendido, i.e. los conjuntos finitos de elementos de X. Se ilustrará cómo puede adelantarse una tal demostración.

Para el diseñador puede ser natural la siguiente definición de una función de representación, que establece qué conjunto se quiere representar con un nombre del TAD:

$$\Phi : Set[X] \dashrightarrow 2^X$$

definida recursivamente así:

$\Phi(empty) = \{\}$

$\Phi(insert(s,x)) = \Phi(s) \cup \{x\}$.

Y ahora es posible demostrar el siguiente resultado:

3.2 Lema:

a) $has(s,x)$ ssi $x \in \Phi(s)$.

b) $\Phi(delete(s,x)) = \Phi(s) \setminus \{x\}$

c) $\Phi(union(s1,s2)) = \Phi(s1) \cup \Phi(s2)$

d) $s1 =_{has} s2$ ssi $\Phi(s1) = \Phi(s2)$.

La prueba de **3.1** sigue fácilmente de este resultado

4. Conclusiones

La nueva definición de igualdad mejora los mecanismos de prueba de teoremas disponibles para un TAD. Para el diseñador es además más explícita la manera de constatar, al menos en forma intuitiva, en qué medida el TAD diseñado abstrae el universo de objetos que se pretende modelar.

El hecho de evitar proporcionar axiomas para operaciones generadoras permite hacer inducción sobre el tipo de interés, tanto para definir el significado de las operaciones selectoras como para llevar a cabo demostraciones de teoremas sobre el TAD.

Cuando el diseñador acepta que un caracterizador K define un igualdad satisfactoria en un TAD Y, está al mismo tiempo aceptando que el universo observable es isomorfo a $Y/=_K$. También se está entendiendo que el universo es un modelo inicial del TAD Y, en el sentido de que dos objetos son iguales en el universo si se puede demostrar la igualdad de los nombres correspondientes en el TAD, y esta igualdad se define como ausencia de diferencias en las características que el observador considera relevantes.

Debe ser claro, además, que entre más pequeño sea un conjunto caracterizador la igualdad asociada es más sencilla de utilizar. Por ésto es recomendable buscar caracterizadores minimales. Así, si K es un caracterizador y $f \in K$, si $.=_K.$ discrimina tanto como $.=_{K,f}.$, entonces es preferible elegir a $K \setminus \{f\}$ como caracterizador.

BIBLIOGRAFIA

[Bar87] Barón, R., "Una herramienta para el diseño de tipos abstractos de datos", Tesis de grado, Universidad de los Andes, Bogotá, 1987.

[Bau82] Bauer, F.L., Wössner H., Algorithmic Language and Program Development, Springer Verlag, 1982.

[Car86] Cardoso, R., "Diseño e implementación de tipos abstractos de datos", CLEI 1986, Montevideo

****