Listas: Una Formalización Relacional

Pablo E. Martínez López* Gabriel A. Baum*

Resumen

Las Fork Álgebras constituyen un poderoso cálculo relacional para la derivación de programas. Esta clase de álgebras resulta de extender las álgebras relacionales con un nuevo operador, llamado *fork* que permite la introducción, por definición, de proyecciones. En este artículo damos una axiomatización del tipo de datos lista expresada en el lenguaje de las fork álgebras. La importancia de contar con una teoría de listas expresada en dicho lenguaje, con una notación uniforme y precisa, es imperiosa, ya que buena parte de la tarea de construcción formal de programas recae en el adecuado manejo de los tipos de datos. El método utilizado consiste en introducir constantes de relaciones junto con axiomas que las mismas deben cumplir para expresar las operaciones básicas sobre listas. Además, se prueban algunas propiedades conocidas utilizando los axiomas y se definen algunas operaciones derivadas para operar con listas.

Palabras clave: fork álgebras, construcción formal de programas, teoría de listas, tipos de datos.

*Universidad Nacional de La Plata. Departamento de Informática. LIFIA, C.C.11, Correo Central. 1900, La Plata, Provincia de Buenos Aires, República Argentina. E-mail: {fidel,gbaum}@info.unlp.edu.ar

Listas: Una Formalización Relacional

1 Introducción

Las Fork Álgebras constituyen un poderoso cálculo relacional para la derivación de programas. Esta clase de álgebras resulta de extender las álgebras relacionales con un nuevo operador, llamado *fork* que permite la introducción, por definición, de proyecciones. El cálculo abstracto asociado a las fork álgebras manipula términos relacionales sin variables sobre individuos. Sobre la base de este cálculo se ha desarrollado un formalismo para construcción formal de software. Dos cuestiones básicas para evaluar un formalismo de derivación de programas tienen que ver con sus aspectos formales (corrección y completitud) y su adecuación para razonar y derivar programas a partir de especificaciones. Estas cuestiones fundamentales han sido establecidas en términos de los teoremas de expresividad (que muestra que la teoría ecuacional de las fork álgebras tiene el poder expresivo del cálculo de predicados de primer orden) y representabilidad (que muestra que cualquier modelo de nuestro cálculo es isomorfo al modelo conjuntista de relaciones, que captura la semántica de entrada–salida de los programas). La potencia de nuestro cálculo para derivar y razonar acerca de programas ha sido ilustrada a través de una importante cantidad de ejemplos (ver [9, 14, 12, 23, 2, 1]). En ellos se puede ver que una buena parte de la tarea de contrucción de programas se basa en la adecuada definición y manipulación de tipos de datos (listas, árboles binarios, etc.). En los citados trabajos dichos tipos se utilizan asumiendo que se tiene un conocimiento intuitivo de ellos, y utilizando las propiedades conocidas de los mismos. En ese sentido, resulta de gran utilidad una formalización de la teoría de listas ([5]) expresada en el lenguaje de fork álgebras. Uno de los objetivos de este artículo es introducir notación que pueda ser utilizada en forma estándar a la hora de derivar programas, pero con una base cierta y definida que permita que las estructuras de datos no sean sólo una noción intuitiva. El método utilizado consiste en introducir nuevas constantes de relación junto con un conjunto de axiomas que las mismas deben cumplir para representar a las operaciones básicas sobre listas. A partir de estos axiomas se prueban algunas propiedades conocidas sobre listas y, utilizando las operaciones básicas introducidas, se definen algunas operaciones clásicas para manipularlas.

La especificación relacional de tipos de datos no es una idea nueva; uno de los primeros ejemplos parece ser un artículo de de Bakker y de Roever de 1972, [6]. Posteriormente, se realizó un importante desarrollo en el tema, [11, 7, 3, 4]; sin embargo el enfoque de estos trabajos se basa en un álgebra heterogénea, y nuestro interés es el de realizar esta tarea en el marco homogéneo que brindan las fork álgebras. El estudio de especificaciones relacionales en fork álgebras fue iniciado en [8], pero no fue continuado sistemáticamente, faltando hoy un desarrollo imprescindible en el tema. Finalmente, en [10] se caracterizan relacionalmente las listas en el marco de las fork álgebras, pero se introducen mediante las operaciones de hd y tl como primitivas (lo cual exige el uso de inducción para definir la operación conc), y exige como primitiva la interalización del conjunto de listas, que puede definirse a partir de las otras operaciones; sin embargo este método es de naturaleza funcional, y no aprovecha la totalidad del poder relacional de las fork álgebras. El enfoque que seguiremos es el de introducir las listas como *join-lists*, siguiendo las ideas de [5] y [18].

Este artículo se divide en dos secciones principales. En la Secc. 2 se presentan las fork álgebras, tanto propias como abstractas, basando su presentación en las álgebras relacionales, de las que se da una breve introducción, y se tratan los problemas de expresividad y representabilidad de las fork álgebras. En la Secc. 3 se presenta la axiomatización del tipo de datos lista expresada en términos del lenguaje introducido en la Secc. 2.

2 Fork Álgebras

En esta sección presentaremos los modelos que motivaron el uso de las fork álgebras como un cálculo de especificación y diseño de programas, las fork álgebras propias, daremos una axiomatización abstracta, que caracteriza completamente (salvo isomorfismo) dicha clase de modelos, presentaremos los teoremas de expresividad (Teo. 2.7) y representabilidad (Teo. 2.8), discutiremos algunas consecuencias que los mismos tienen en el uso de fork álgebras como lenguaje de construcción formal de programas, y definiremos abreviaturas de notación para algunos términos de uso común en la derivación de programas. Dado que las fork álgebras son extensión de las álgebras relacionales, que a su vez extienden a las álgebras de Boole, es conveniente recordar brevemente esta progresión, haciendo especial hincapié en las versiones concreta (o propia) y abstracta de cada una.

2.1 Conceptos básicos

Consideremos un conjunto W. Un *campo de conjuntos* sobre W es una estructura $\mathsf{SF} = <F, \cup, \cap, ', \emptyset, W>$ donde $F \subseteq \mathcal{P}(W)$ y W es cerrado para las operaciones $\cup$ (unión), $\cap$ (intersección), y ' (complemento respecto de W), y tal que $\emptyset, W \in F$.

La versión abstracta de un campo de conjuntos es un *álgebra de Boole*, o sea una estructura $\mathsf{BA} = <B, \vee, \wedge, ', 0, 1>$ que satisface el bien conocido conjunto de axiomas ([20]). Recordemos que sobre esta estructura se puede definir una relación $\preceq$, dada por $x \preceq y$ sii $x \vee y = y$, proveyendo al álgebra de Boole con una estructura subyacente de reticulado.

El teorema de Stone establece que toda álgebra de Boole es isomorfa a un campo de conjuntos. El orden parcial $\preceq$ se corresponde con la inclusión usual de conjuntos ($\subseteq$).

Consideremos ahora un conjunto $V \subseteq W \times W$ (o sea un conjunto de pares ordenados de elementos de W). Sobre los subconjuntos de V se pueden definir las operaciones de conversa ($^\smile$) y producto relacional ($|$) como $\mathsf{p}^\smile = \{<u,v>/<v,u> \in \mathsf{p}\}$ y $\mathsf{p} \mid \mathsf{q} = \{<u,w>/(\exists v \in W)(<u,v> \in \mathsf{p} \wedge <v,w> \in \mathsf{q})\}$; además, la relación diagonal (o relación identidad), $\Delta = \{<u,v>/<u,v> \in V \wedge u = v\}$ es un subconjunto de V. Un *álgebra de relaciones* (o álgebra relacional propia) es una estructura $\mathsf{PRA} = <P, \cup, \cap, |, ', ^\smile, \emptyset, V, \Delta>$, con $P \subseteq \mathcal{P}(V)$, tal que el reducto $<P, \cup, \cap, ', \emptyset, V>$ es un campo de conjuntos y P contiene a la relación diagonal y es cerrado por las operaciones $^\smile$ y $|$.

La versión abstracta de un álgebra de relaciones es llamada *álgebra relacional (abstracta)*, y consiste en una estructura $\mathsf{RA} = <A, +, \bullet, ;, ^-, ^\smile, 0, \infty, 1>$, tal que el reducto $<A, +, \bullet, ^-, 0, \infty>$ es un álgebra de Boole, el reducto $<A, ;, 1>$ es un monoide, y A cumple la regla de Schröeder ($\mathsf{R};\mathsf{S} \preceq \mathsf{T}$ sii $\mathsf{R}^\smile;\overline{\mathsf{T}} \preceq \overline{\mathsf{S}}$ sii $\overline{\mathsf{T}};\mathsf{S}^\smile \preceq \overline{\mathsf{R}}$) y la regla de Tarski ($\mathsf{R} \neq 0 \rightarrow \infty;\mathsf{R};\infty = \infty$), para todo R, S y T en A. Los

símbolos 0, ∞, 1, ; y $\smile$ denotan los elementos *cero*, *unidad*, *identidad*, y las operaciones producto relativo y conversa, respectivamente; el símbolo $\preceq$ corresponde al orden del reticulado subyacente y es llamado *inclusión relacional*.

En [21] se establece cuales son las álgebras relacionales representables mediante un álgebra de relaciones.

2.2 Fork Álgebras Propias

Las fork álgebras propias son álgebras de relaciones extendidas con un operador nuevo, llamado *fork*, diseñado para tratar con objetos complejos (con una estructura similar a la de los árboles binarios).

Definición 2.1 Una *$\star$–Fork Álgebra Propia* sobre un conjunto W es una estructura

$$\star\text{–PFA} = <P, \cup, \cap, |, \nabla, ', ^\sim, \emptyset, V, \Delta, \star>$$

tal que:

1. $<P, \cup, \cap, |, ', ^\sim, \emptyset, V, \Delta>$ es un álgebra propia de relaciones
2. $\star$ es una operación binaria inyectiva de V en W
3. P es cerrada para la operación ∇ definida por

$$R \nabla S = \{<x, (y \star z)> / <x, y> \in R \text{ y } <x, z> \in S \text{ y } <y, z> \in V\}$$ ■

Es importante destacar que los "árboles" formados por la operación $\star$ pueden representar órdenes no bien fundados, y tienen entonces ramas infinitas. A pesar de que a primera vista puede parecer una desventaja, esta propiedad puede ser útil para modelizar procesos o computaciones infinitas, como se muestra en [9]. Además, estas estructuras arbóreas permiten manejar tantas variables como sea necesario al representar fórmulas de primer orden mediante términos del álgebra.

Las Fork Álgebras Propias se definen entonces de la siguiente manera:

Definición 2.2 Una *Fork Álgebra Propia* sobre un conjunto W es el reducto

$$\text{PFA} = <P, \cup, \cap, |, \nabla, ', ^\sim, \emptyset, V, \Delta>$$

de una $\star$–fork álgebra propia. ■

Observemos que una relación binaria puede representar la relación de entrada–salida de un programa secuencial. Las operaciones del álgebra representarán entonces a operaciones entre programas; por ejemplo, el producto relacional representará la composición secuencial, y la operación fork (∇) la formación de pares en los datos de salida de un programa.

Un punto de importancia al utilizar relaciones para representar programas es la forma en que se representan tipos de datos. Esto hace que la manera en que se representan conjuntos como relaciones sea de gran importancia, ya que los tipos de datos pueden entenderse como conjuntos de valores con ciertas propiedades. Existen varias maneras de realizar la "internalización" de un conjunto (ver [21, 23]); la

usada en este artículo consiste en representar al conjunto X mediante la relación $\{<x,x>/x \in X\}$, y se denominará identidad parcial Δ_X, por estar incluída propiamente en la relación Δ.

Otro concepto de importancia al considerar a las relaciones binarias como representación de programas son los programas "constantes", o sea aquellos que aceptan cualquier dato de entrada y producen siempre el mismo dato de salida. Dichas relaciones tienen la forma $\{<x,c>/<x,x> \in V\}$, para un elemento dado c. En la sección de fork álgebras abstractas daremos una caracterización abstracta de este tipo de relaciones.

2.3 Fork Álgebras Abstractas

Las fork álgebras propias utilizan variables para denotar dos clases de entidades: algunas denotan relaciones (como R y S en la Def. 2.1) y otras denotan individuos (como x, y, z, en la misma definición). Una caracterización abstracta de las fork álgebras propias como una clase de primer orden, en la cual las variables denotan sólo relaciones, se obtiene extendiendo las álgebras relacionales (versión abstracta de las álgebras de relaciones) con una operación que sea la contrapartida abstracta de la operación $\underline{\nabla}$.

Definición 2.3 Una *Fork Algebra* (*Abstracta*) es una estructura

$$\mathsf{AFA} = <A, +, \bullet, ;, \nabla, ^-, ^\smile, 0, \infty, 1>$$

(donde la operación ∇ es llamada *fork*) tal que:

1. $<A, +, \bullet, ;, ^-, ^\smile, 0, \infty, 1>$ es un álgebra relacional
2. $\mathsf{R} \nabla \mathsf{S} = (\mathsf{R};(1 \nabla \infty)) \bullet (\mathsf{S};(\infty \nabla 1))$
3. $(\mathsf{R} \nabla \mathsf{S});(\mathsf{T} \nabla \mathsf{Q})^\smile = (\mathsf{R};\mathsf{T}^\smile) \bullet (\mathsf{S};\mathsf{Q}^\smile)$
4. $(1 \nabla \infty)^\smile \nabla (\infty \nabla 1)^\smile \preceq 1$
5. $0 \neq \mathsf{T} \preceq ((1 \nabla \infty)^\smile; \infty) \nabla ((\infty \nabla 1)^\smile; \infty) \rightarrow$
$\rightarrow (\exists \mathsf{V})(\exists \mathsf{W})(0 \neq ((1 \nabla \infty)^\smile; \mathsf{V}) \nabla ((\infty \nabla 1)^\smile; \mathsf{W}) \preceq \mathsf{T})$ ■

Cada una de las operaciones de una fork álgebra abstracta intenta ser la contrapartida abstracta de las operaciones de una fork álgebra propia; en el Teo. 2.8 demostramos que efectivamente lo son.

Con respecto a la internalización de conjuntos, utilizaremos la versión abstracta de las identidades parciales, esto es, relaciones incluídas en 1. Así, anotamos 1_X para denotar la identidad parcial que internaliza al conjunto X.

La caracterización abstracta de las relaciones constantes es la siguiente:

Definición 2.4 Una relación R se dice *constante* sii cumple que $\mathsf{R} = \infty;\mathsf{R}$ y $\mathsf{R}^\smile;\mathsf{R} \preceq 1$. ■

La primer condición establece que la "entrada" es cualquier elemento del universo y la segunda establece que la "salida" es un único elemento.

A modo de ejemplo de las posibilidades de expresión de las fork álgebras, introducimos las nociones de funcionalidad e inyectividad de una relación.

Definición 2.5 Una relación R se dice *funcional* sii $\mathsf{R}^\smile;\mathsf{R} \preceq 1$, e *inyectiva* sii $\mathsf{R};\mathsf{R}^\smile \preceq 1$. ■

De la misma manera que éstas, otras propiedades pueden expresarse de manera abstracta en el lenguaje de las fork álgebras; en la Secc. 3 daremos algunos ejemplos adicionales.

2.4 Operaciones derivadas

En esta sección introduciremos algunas operaciones no fundamentales para las fork álgebras que serán utilizadas en los procesos de especificación y derivación. Las definiciones se dan en forma abstracta, pero describimos a continuación cual es la intención de cada una de estas operaciones desde el punto de vista de los modelos estándar.

Definición 2.6 Sean π, ρ, $\otimes$, 2, $Dom()$ y $Ran()$ las operaciones definidas por

1. $\pi = (1 \nabla \infty)^{\smile}$ (*primera proyección*)
2. $\rho = (\infty \nabla 1)^{\smile}$ (*segunda proyección*)
3. $\mathsf{R} \otimes \mathsf{S} = (\pi ; \mathsf{R}) \nabla (\rho ; \mathsf{S})$ (*producto*)
4. $2 = 1 \nabla 1$ (*duplicación de datos*)
5. $Dom(\mathsf{R}) = (\mathsf{R} ; \mathsf{R}^{\smile}) \bullet 1$ (*dominio de* R)
6. $Ran(\mathsf{R}) = (\mathsf{R}^{\smile} ; \mathsf{R}) \bullet 1$ (*rango de* R)

Las operaciones π y ρ representan a las relaciones de proyección respecto de la operación subyacente $\star$. Dado que la relación 1 representa a la relación identidad, la relación 2 representa a la operación de duplicación de sus datos de entrada, y $2^{\smile}$ resulta ser un filtro de igualdad sobre $<x \star y>$ tal que $x=y$. $Dom(\mathsf{R})$ y $Ran(\mathsf{R})$ son expresiones relacionales que caracterizan el *dominio* y el *rango* de la relación R. Estas operaciones satisfacen un cierto conjunto de propiedades, que se deducen de su definición. Sólo para mencionar algunas de ellas, podemos ver que π, ρ y 2 son relaciones funcionales e inyectivas, que para toda relación R, $Dom(\mathsf{R}) \preceq 1$ y $Ran(\mathsf{R}) \preceq 1$ y representan por tanto a conjuntos, que $Dom(\pi) = Dom(\rho) = 1 \otimes 1$, y que $Dom(\mathsf{R}) ; \mathsf{R} = \mathsf{R} ; Ran(\mathsf{R}) = \mathsf{R}$.

2.5 La Expresividad de las Fork Álgebras

El problema de expresividad consiste en saber si cualquier fórmula del cálculo de predicados de primer orden puede ser representado mediante un término relacional. La importancia de esta pregunta reside en que, de tener una respuesta negativa, el lenguaje involucrado no sería adecuado para la tarea de construcción formal de programas. El lenguaje de álgebras relacionales (ver Secc. 2.1) tiene operaciones como la conversa o el complemento, que son muy poderosas en el proceso de construcción de especificaciones (las dos permiten especificar problemas no-funcionales); sin embargo, su poder expresivo es muy limitado ([19]) y por lo tanto no sirve para la tarea de construcción de software. Las fork álgebras solucionan este problema, mediante la introducción del operador *fork*.

El teorema de expresividad de las fork álgebras establece que el poder expresivo de éstas abarca al del cálculo de predicados de primer orden. Fue demostrado por primera vez en [22] y se puede encontrar una demostración más detallada en [13].

Teorema 2.7 *Dado un lenguaje de primer orden $\mathcal{L}$ existe una función T que asigna a cada fórmula n-aria ϕ de $\mathcal{L}$ un fork-término* T, *tal que* T *es la internalización del conjunto* $\{x/\phi(x)\}$. ■

A pesar de que ya existen otros marcos con estas características (por ejemplo las álgebras *Cilíndricas* [16, 17] y las álgebras *Poliádicas* [15]), los mismos poseen desventajas para realizar especificación y desarrollo de programas cuando se las compara con las fork álgebras, pues tienen signaturas infinitas (cantidad infinita de cilindrificaciones y elementos diagonales, en el caso de las álgebras cilíndricas, o transformaciones, en el caso de las álgebras poliádicas).

2.6 La Representabilidad de las Fork Álgebras

El problema de la representabilidad consiste en saber si todos los modelos del álgebra abstracta son isomorfos a algún álgebra propia. Aunque el teorema de representabilidad pueda parecer interesante sólo para los teóricos, tiene un gran impacto en la especificación y diseño de programas en el marco de las fork álgebras. Un corolario inmediato del teorema dice que el conjunto de fórmulas de primer orden válidas en la clase de las PFA es exactamente el de la fórmulas válidas en la clase de las AFA; por lo tanto, cualquier propiedad válida en los modelos estándar es válida también en los abstractos, lo que nos permite que, a pesar de trabajar con términos del álgebra abstracta, podamos recurrir a la intuición que nos brindan las álgebras propias.

Teorema 2.8 *Toda fork álgebra completa y atomística es isomorfa a una fork álgebra propia.* ■

2.7 Consecuencias de la Expresividad y la Representabilidad

Es generalmente aceptado que el lenguaje de la lógica de primer orden es adecuado para especificar problemas computacionales (entendidos como una relación entrada–salida). Gracias al teorema de expresividad, sabemos que cada especificación posible puede traducirse a un término del álgebra, equivalente en el sentido que representa la misma relación de entrada–salida; el teorema provee, de paso, un método para realizar dicha traducción.

Por otro lado, el modo algebraico de proceder en el marco de un cálculo ecuacional concreta la idea de que la construcción rigurosa de programas puede transformarse en un auténtico cálculo de programas a partir de especificaciones. El teorema de representabilidad asegura que toda manipulación hecha en el cálculo ecuacional de las fork álgebras abstractas es correcta respecto del modelo elemental intuitivo (relaciones binarias de entrada–salida).

En conclusión, las fork álgebras abstractas constituyen un cálculo relacional potente y adecuado que permite sentar las bases para el desarrollo de una metodología de construcción formal de software.

3 Listas Relacionales

En esta sección presentamos relacionalmente el tipo de datos lista, introduciendo constantes de relación y axiomas que éstas deben cumplir para representar las operaciones básicas sobre listas. Además se prueban

algunas propiedades conocidas sobre listas utilizando los axiomas y se introducen algunas operaciones derivadas, cómo términos que utilizan las operaciones básicas.

3.1 Listas No Vacías

Una lista no vacía sobre un conjunto S se construye mediante dos operaciones básicas:

- una operación unaria $\hat{}$, que para cada elemento del conjunto S construye una lista con ese único elemento, y
- una operación binaria $+$, que dadas dos listas de elementos de S, construye la lista que resulta de colocar cada uno de los elementos de la segunda al final de la primera.

Para expresar relacionalmente al conjunto de las listas no vacías sobre un conjunto S, al que llamaremos $\mathcal{L}_S^+$, introducimos dos nuevas constantes relacionales: tip y conc. La relación tip expresará a la operación $\hat{}$ y la relación conc expresará a la operación $+$. Para la operación tip tenemos los siguientes axiomas:

Axioma 3.1 *El símbolo* tip *debe satisfacer:*

1. $\mathsf{tip}^\smile;\mathsf{tip} \preceq 1$ *(*tip *es funcional);*
2. $\mathsf{tip};\mathsf{tip}^\smile = 1_S$ *(*tip *es inyectiva).*

El Ax. 3.1-2 implica que $Dom(\mathsf{tip}) = 1_S$, estableciendo que tip está definida para todo elemento de S. Por otro lado, la operación $\hat{}$ nos permite definir el subconjunto de $\mathcal{L}_S^+$ que está compuesto por las listas que contienen un solo elemento; este subconjunto será llamado $\mathcal{L}_S^1$, y su internalización como un término de las fork álgebras es:

Definición 3.2 $1_{\mathcal{L}_S^1} = Ran(\mathsf{tip}) = \mathsf{tip}^\smile;\mathsf{tip}$. (por Def. 2.6-6 y Ax. 3.1-1) ∎

Para el símbolo conc tenemos los siguientes axiomas:

Axioma 3.3 *El símbolo* conc *debe satisfacer:*

1. $\mathsf{conc}^\smile;\mathsf{conc} \preceq 1$ *(*conc *es funcional);*
2. $Dom(\mathsf{conc}) = (Ran(\mathsf{conc}) + Ran(\mathsf{tip}) \otimes Ran(\mathsf{conc}) + Ran(\mathsf{tip}))$ *(el "tipo" de* conc*);*
3. $\mathsf{conc};\mathsf{conc}^\smile = \mathsf{assoc}$ *(*conc *es asociativo).*

donde assoc es una abreviatura definida de la siguiente manera:

Definición 3.4 Sean $\mathsf{reassoc}$ y assoc los siguientes términos:

1. $\mathsf{reassoc} = ((\pi \nabla(\rho;\pi))\nabla(\rho;\rho))$
2. $\mathsf{assoc} = (1 \otimes \mathsf{conc}^\smile);\mathsf{reassoc};(\mathsf{conc} \otimes 1) + (\mathsf{conc}^\smile \otimes 1);\mathsf{reassoc}^\smile;(1 \otimes \mathsf{conc}) + Dom(\mathsf{conc})$ ∎

En los modelos propios, la operación reassoc permite transformar el par $<x,<y,z>>$ en el par $<<x,y>,z>$, permitiendo expresar la ley asociativa de conc. Es fácil demostrar que assoc es una relación de equivalencia sobre el conjunto de pares de listas. Estas abreviaturas permiten simplificar la notación y por lo tanto la legibilidad de los términos relacionales.

La operación + nos permite definir el subconjunto de $\mathcal{L}_S^+$ que está compuesto por listas generadas a partir de otras listas, al que llamaremos $\mathcal{L}_S^{>1}$:

Definición 3.5 $1_{\mathcal{L}_S^{>1}} = Ran(\mathsf{conc}) = \mathsf{conc}^\smile;\mathsf{conc}$. (por Def. 2.6-6 y Ax. 3.3-1) ■

El inciso 3 del Ax. 3.3 es una forma de establecer que la operación conc es asociativa, como se ve en la siguiente propiedad:

Proposición 3.6 *Ax. 3.3-3* $\rightarrow (1 \otimes \mathsf{conc});\mathsf{conc} = \mathsf{reassoc};(\mathsf{conc} \otimes 1);\mathsf{conc}$ ■

Para expresar que ninguna lista compuesta por un único elemento es igual a una lista obtenida mediante +, agregamos el siguiente axioma:

Axioma 3.7 $\mathsf{tip};\mathsf{conc}^\smile = 0$ ■

Este axioma implica que $1_{\mathcal{L}_S^1} \bullet 1_{\mathcal{L}_S^{>1}} = 0$ y por lo tanto que la siguiente definición establece una partición del conjunto $\mathcal{L}_S^+$, lo cual será muy útil a la hora de derivar programas, ya que la estrategia de trivialización requiere contar con una partición del dominio. La internalización del conjunto $\mathcal{L}_S^+$ de las listas no vacías sobre el conjunto S se define entonces:

Definición 3.8 $1_{\mathcal{L}_S^+} = 1_{\mathcal{L}_S^1} + 1_{\mathcal{L}_S^{>1}}$ ■

Esta definción nos permite reescribir Ax. 3.3-2 como $Dom(\mathsf{conc}) = \left(1_{\mathcal{L}_S^+} \otimes 1_{\mathcal{L}_S^+}\right)$, que deja claro por qué en el Ax. 3.3-2 el comentario hablaba del "tipo" de conc.

Para expresar la inducción sobre listas, usaremos la mínima solución de la ecuación recursiva

$$\mathsf{destr} = \mathsf{tip}^\smile + \mathsf{conc}^\smile;(\mathsf{destr} \otimes \mathsf{destr})$$

en el siguiente axioma:

Axioma 3.9 $Dom(\mathsf{destr}) = 1_{\mathcal{L}_S^+}$ *(*$1_{\mathcal{L}_S^+}$ *es un tipo inductivo)*

que expresa que toda lista se obtiene como concatenación de una cantidad finita de listas unitarias.

3.2 Listas con Lista Vacía

El tipo de las listas que pueden ser vacías se define de la misma manera que el de las listas no vacías, pero donde se establece que la operación + tiene un neutro; este tipo será llamado como $\mathcal{L}_S^*$, y se obtiene extendiendo $\mathcal{L}_S^+$ con un nuevo elemento, llamado [], que satisface $[]+\mathbf{b} = \mathbf{b}+[]= \mathbf{b}$ para todo $\mathbf{b} \in \mathcal{L}_S^+$. En la teoría relacional, introducimos el símbolo nil para representar a [].

Axioma 3.10 *El símbolo* nil *satisface:*

1. $\mathsf{nil} = \infty ; \mathsf{nil}, \; y$
   $\mathsf{nil}^{\smile} ; \mathsf{nil} \preceq 1$ *(*nil *es una constante);*

2. $\mathsf{nil} ; \mathsf{tip}^{\smile} = 0, \; y$
   $\mathsf{nil} ; \mathsf{conc}^{\smile} = 0$ *(*nil *es un elemento "nuevo").*

El subconjunto de $\mathcal{L}_S^*$ compuesto sólo por [] (denotado $\mathcal{L}_S^0$) se internaliza:

Definición 3.11 $1_{\mathcal{L}_S^0} = Ran(\mathsf{nil}) = \mathsf{nil}^{\smile} ; \mathsf{nil}$ (por Def. 2.6-6 y Ax. 3.10-1) ■

Esta última definición y el Ax. 3.10-2 nos permiten internalizar el conjunto $\mathcal{L}_S^*$ como una partición:

Definición 3.12 $1_{\mathcal{L}_S^*} = 1_{\mathcal{L}_S^0} + 1_{\mathcal{L}_S^+}$ ■

La operación $+$ de $\mathcal{L}_S^*$ (extensión de la operación $+$ de $\mathcal{L}_S^+$), debe ser expresada por un nuevo símbolo relacional, append, definido por:

Definición 3.13 $\mathsf{append} = \left(1_{\mathcal{L}_S^0} \nabla 1_{\mathcal{L}_S^0}\right)^{\smile} + \left(1_{\mathcal{L}_S^0} \otimes 1_{\mathcal{L}_S^+}\right) ; \rho + \left(1_{\mathcal{L}_S^+} \otimes 1_{\mathcal{L}_S^0}\right) ; \pi + \mathsf{conc}$ ■

Con esta definición, podemos probar que nil es neutro para append.

Proposición 3.14 $(1 \nabla \mathsf{nil}) ; \mathsf{append} = (\mathsf{nil} \nabla 1) ; \mathsf{append} = 1_{\mathcal{L}_S^*}$

Habiendo introducido las listas a través de las relaciones nil, tip y conc, podemos escribir algunas funciones clásicas sobre listas.

Definición 3.15 Sean hd, tl, init y last las operaciones definidas por

1. $\mathsf{hd} = \mathsf{conc}^{\smile} ; (\mathsf{tip}^{\smile} \otimes 1) ; \pi + \mathsf{tip}^{\smile}$ (primer elemento)
2. $\mathsf{tl} = \mathsf{conc}^{\smile} ; (\mathsf{tip}^{\smile} \otimes 1) ; \rho + \mathsf{tip}^{\smile} ; \mathsf{nil}$ (todos, menos el primero)
3. $\mathsf{last} = \mathsf{conc}^{\smile} ; (1 \otimes \mathsf{tip}^{\smile}) ; \rho + \mathsf{tip}^{\smile}$ (último elemento)
4. $\mathsf{init} = \mathsf{conc}^{\smile} ; (1 \otimes \mathsf{tip}^{\smile}) ; \pi + \mathsf{tip}^{\smile} ; \mathsf{nil}$ (todos, menos el último)

Se puede probar que las cuatro son operaciones funcionales y que su dominio es igual a $1_{\mathcal{L}_S^+}$.

3.3 Un ejemplo: rev

A modo de ejemplo del uso de la axiomatización presentada, construiremos la función que calcula el reverso de una lista. Para ello daremos una especificación axiomática del símbolo relacional rev y luego derivaremos una ecuación recursiva cuya mínima solución será el término buscado.

Para simplificar la notación en la especificación de rev, daremos la definición de una abreviatura:

Definición 3.16 $\mathsf{swap} = (\rho \nabla \pi)$

En los modelos propios esta operación permite transformar al par $<x, y>$ en el par $<y, x>$.

La especificación establece que rev es inyectiva y como afecta a las listas obtenidas mediante los distintos constructores.

- $\mathsf{rev} ; \mathsf{rev}^{\smile} = 1_{\mathcal{L}_S^*}$ (rev es inyectiva)

- $\mathsf{nil};\mathsf{rev} = \mathsf{nil}$ (la lista vacía permanece invariante por rev)
- $\mathsf{tip};\mathsf{rev} = \mathsf{tip}$ (las listas unitarias permanecen invariantes por rev)
- $\mathsf{conc};\mathsf{rev} = (\mathsf{rev} \otimes \mathsf{rev});\mathsf{swap};\mathsf{conc}$

El primer inciso implica que $Dom(\mathsf{rev}) = 1_{\mathcal{L}_S^*}$. Para derivar una solución para esta especificación, multiplicamos los tres incisos restantes por $\mathsf{nil}^\smile$, $\mathsf{tip}^\smile$ y $\mathsf{conc}^\smile$, respectivamente y, utilizando las Defs. 3.11, 3.2 y 3.5, obtenemos que $1_{\mathcal{L}_S^0};\mathsf{rev} = 1_{\mathcal{L}_S^0}$, que $1_{\mathcal{L}_S^1};\mathsf{rev} = 1_{\mathcal{L}_S^1}$, y que $1_{\mathcal{L}_S^{>1}};\mathsf{rev} = \mathsf{conc}^\smile;(\mathsf{rev} \otimes \mathsf{rev});\mathsf{swap};\mathsf{conc}$. Sumando miembro a miembro y utilizando la distributividad de + sobre ;, tenemos $(1_{\mathcal{L}_S^0} + 1_{\mathcal{L}_S^1} + 1_{\mathcal{L}_S^{>1}});\mathsf{rev} = 1_{\mathcal{L}_S^0} + 1_{\mathcal{L}_S^1} + \mathsf{conc}^\smile;(\mathsf{rev} \otimes \mathsf{rev});\mathsf{swap};\mathsf{conc}$ y, finalmente, aplicando la Def. 3.12 y propiedades de dominios, obtenemos una ecuación recursiva cuya mínima solución es el término buscado para rev:

$$\mathsf{rev} = 1_{\mathcal{L}_S^0} + 1_{\mathcal{L}_S^1} + \mathsf{conc}^\smile;(\mathsf{rev} \otimes \mathsf{rev});\mathsf{swap};\mathsf{conc}$$

El Ax. 3.9 implica que esta ecuación tiene solución.

4 Conclusiones y Trabajos Futuros

Hemos dado una axiomatización del tipo de datos lista en el marco de las fork álgebras, presentándolas como *join-lists*. Con ella hemos probado algunas propiedades conocidas y hemos definido algunas funciones clásicas sobre listas.

A lo largo del trabajo hemos utilizado numerosas veces la conversa de relaciones como medio de definir internalizaciones de conjuntos y funciones, establecer propiedades, etc., haciendo uso de toda la potencia de un cálculo relacional. Además, todas las internalizaciones de conjuntos fueron definidas en función de las operaciones básicas, mostrando que no es necesario incluírlas como primitivas, y establecieno los "tipos" de las distintas operaciones en un marco homogéneo.

Posibles continuaciones de este trabajo son demostrar que esta axiomatización es monomórfica (i.e. todos sus modelos son isomorfos), extender la axiomatización para abarcar otros tipos de datos (árboles binarios, bags, etc.), estudiar homomorfismos sobre listas y su uso en la definición de funciones y/o generalizar este método para tratar con cualquier tipo de datos.

Referencias

[1] Gabriel A. Baum, Marcelo Fabián Frias, Armando M. Haeberer, and Pablo E. Martínez López. Construcción formal de programas con fork álgebras. In *24 Jornadas Argentinas de Informática e Investigación Operativa (JAIIO)*, Agosto 1995.

[2] Gabriel A. Baum, Marcelo Fabián Frias, Armando M. Haeberer, and Pablo E. Martínez López. From specifications to programs: A fork–algebraic approach to bridge the gap. In *Document 731 HKO-4 of the IFIP W.G. 2.1, Hong Kong*. January 1995.

[3] Rudolf Berghammer. Relational specification of data types and programs. Report 9109, Univ. der Bundeswehr München, Fakultät fur Informatik, 1991.

[4] Rudolf Berghammer and Gunther Schmidt. Relational specifications. Report, Univ. der Bundeswehr München, Fakultät fur Informatik, 1992.

[5] Richard S. Bird. An introduction to the theory of lists. In M.Broy, editor, *Logic of Programming and Calculus of Discrete Design, vol.F36 of NATO ASI series*, pages 5–42. Springer–Verlag, 1987.

[6] J. W. de Bakker and W. P. de Roever. A calculus for recursive program schemes. In *Proc. IRIA Symp. on Automata, Formal Languajes and Programming*, pages 167–196, Amsterdam, North Holland, 1972.

[7] J. Desharnais and N.H. Madhavji. Abstract relational specification. In M. Broy and J.B. Jones, editors, *Proc. TC 2 Working Conference on Programming Concepts and Methods, North Holland*, pages 267–284, 1990.

[8] Marcelo Fabián Frias and N. G. Aguayo. Natural specifications vs. abstract specifications. a relational approach. In *Proceedings of SOFSEM '94*, Milovy, Czech Republic, November 1994.

[9] Marcelo Fabián Frias, N. G. Aguayo, and B. Novak. Development of graph algorithms with fork algebras. In *Proceedings of the XIX Latinamerican Conference on Informatics*, pages 529–554, 1993.

[10] Isabel García. Lists in fork algebras. Res. rept., PUC-Rio, 1995. To appear.

[11] Thomas F. Gritzner. *Die Axiomatik abstrakter Relationenalgebren: Darstellung der Grundlagen und Anwendung auf das Unschärfeproblem relationaler Produkte.* PhD thesis, Techn. Univ. München, Institut für Informatik, 1989.

[12] Armando M. Haeberer, Gabriel A. Baum, and Gunther Schmidt. On the smooth calculation of relational recursive expressions out of first–order non–constructive specifications involving quantifiers. In *Proceedings of the Intl. Conference on Formal Methods in Programming and Their Applications, LNCS 735*, pages 281–298, 1993.

[13] Armando M. Haeberer, Marcelo Fabián Frias, Gabriel A. Baum, and Paulo A. S. Veloso. Fork algebras and program construction. Res. rept. PUC-Rio, 1994.

[14] Armando M. Haeberer and P. A. S. Veloso. Partial relations for program derivation: Adequacy, inevitability and expressiveness. In *Constructing Programs from Specifications - Proceedings of the IFIP TC2 Working Conference on Constructing Programs from Specifications*, pages 319–371, North Holland, 1991. IFIP WG. 2.1.

[15] P. R. Halmos. *Algebraic Logic.* New York: Chelsea, 1962.

[16] L. Henkin, J. D. Monk, and Alfred Tarski. *Cylindric Algebras, part I*, volume 64. Studies in Logic and the Foundations of Mathematics, North Holland, 1971.

[17] L. Henkin, J. D. Monk, and Alfred Tarski. *Cylindric Algebras, part II*, volume 115. Studies in Logic and the Foundations of Mathematics, North Holland, 1985.

[18] Johan Jeuring. Algorithms from theorems. In M. Broy and C.B. Jones, editors, *Proc. TC 2 Working Conference on Programming Concepts and Methods*, pages 247–266. Elsevier Science (North Holland), 1990.

[19] L. Löwenheim. Über Möglichkeiten im Relativkalkül. In *Math. Ann. vol. 76*, pages 447–470, 1915.

[20] Lía Oubiña and Rubén Zucchello. *Estructuras Algebraicas.* Editorial Exacta, March 1994.

[21] Gunther Schmidt and T. Ströhlein. *Relations and Graphs.* EATCS Monographs on Theoretical Computer Science. Springer–Verlag, 1993.

[22] Paulo A. S. Veloso and Armando M. Haeberer. A finitary relational algebra for classical first-order logic, vol. 20, no. 2. In *Bull. Section of Logic*, pages 52–62. Polish Acad. Sciences, 1991.

[23] Paulo A. S. Veloso, Armando M. Haeberer, and Gabriel A. Baum. Formal program construction within an extended