Construcción y Optimización de Programas en Fork Álgebras

Gabriel Baum
LIFIA
Departamento de Informática
Facultad de Ciencias Exactas
Universidad Nacional de La Plata
gbaum@info.unlp.edu.ar

Marcelo Frías
Laboratório de Métodos Formais
Dpto. de Informática
Pontifícia Universidade Católica do Rio de Janeiro
mfrías@inf.puc-rio.br

Nazareno Aguirre Ricardo Medel
Área de Computación
Facultad de Ciencias Exactas, Físico-Químicas y Naturales
Universidad Nacional de Río Cuarto
{naguirre,rhmedel}@exa.unrc.edu.ar

Resumen

La creciente importancia de los factores críticos en el software hace que la utilización de métodos formales de desarrollo sea cada vez más frecuente. En este contexto, la construcción rigurosa de programas, concebida como un "álgebra de la programación" [BdM97] o bien como un "cálculo de programas" [FBH 97], constituye uno de los elementos de mayor relevancia, en la medida que provee técnicas, métodos y, más aún, estrategias generales que posibilitan obtener programas correctos por construcción a partir de especificaciones formales o semi formales. En este trabajo se presentan Estrategias de Construcción de Programas basadas en la Teoría de Primer Orden de las Fork Álgebras. En este contexto se describen estrategias, usuales en la Programación en Lógica y Funcional, como "tupling" y "generalización" que resultan de gran utilidad tanto como herramientas de solución de problemas, como técnicas de diseño de algoritmos y también como métodos generales de optimización de programas recursivos. Se analizan condiciones suficientes para la aplicación de estas estrategias a expresiones algebraicas que caracterizan algoritmos genéricos (es decir, clases de algoritmos) y se presentan varios ejemplos de aplicación de las mismas.

1. Introducción

En una visión idealizada y simplista, la construcción formal de programas podría describirse del siguiente modo: Se parte de una especificación declarativa y clara del problema a resolver y, mediante la utilización de un conjunto de reglas de transformación (que preservan la semántica), se deriva una especificación operacional (programa eficiente) que resuelve el problema en cuestión. Puesto que las reglas aplicadas preservan la semántica de la especificación, el programa resultante de la derivación es correcto por construcción y, en consecuencia, no necesita ser verificado o "testeado". Mas aún, buena parte de la tarea (digamos, las porciones de la derivación que no requieren trabajo creativo) podría perfectamente ser llevada a cabo de manera automática.

Lamentablemente, como se ha dicho, esta es una visión ingenua del proceso de construcción formal de programas. De hecho, dicho proceso difícilmente es lineal y depende en gran medida de la pericia y experiencia del desarrollador. Al igual que en cualquier otra aproximación al desarrollo de software, las diferentes etapas del desarrollo no pueden verse como compartimentos estancos, sino como fases de un proceso que interactúan y se intercalan dinámicamente.

A diferencia de los métodos informales y semi formales, los métodos formales posibilitan, a cada paso, extender un certificado de garantía respecto de la corrección del artefacto de software en desarrollo (por ejemplo, la prueba de validez de la regla de transformación aplicada). Sin embargo, hasta el momento, el desarrollo completo de un sistema de tamaño mediano utilizando métodos formales es una tarea virtualmente irrealizable a escala industrial debido a la carencia de herramientas automáticas o semi-automáticas que ayuden eficazmente a los desarrolladores y, en muchos casos, al grado de formación lógico-matemática que requieren. De todos modos, la aplicación de estos métodos es cada vez más habitual en el desarrollo de porciones críticas en un gran número de sistemas. El núcleo central de la construcción formal de programas consiste, esencialmente, en la capacidad de calcular programas de una manera completamente análoga a la tarea que desarrolla un matemático cuando resuelve un sistema de ecuaciones o prueba constructivamente un teorema.

Una particular clase de formalismos para construcción de programas esta constituida por aquellos basados en cálculos formales; estos formalismos están fundamentados en diversos tipos de lógicas. Las especificaciones son fórmulas, y a un cierto subconjunto de dichas fórmulas se le asigna un significado algorítmico, de manera que son interpretadas como programas en algún lenguaje de programación funcional, lógico o imperativo. Las reglas de derivación actúan como las reglas de inferencia de los sistemas lógicos correspondientes.

Las Fork Álgebras surgieron en las Ciencias de la Computación como resultado de la búsqueda de un cálculo para la construcción de programas basado en relaciones binarias. En este contexto, los programas son concebidos como relaciones binarias parciales entre datos de entrada y salida (o, eventualmente, entre estados iniciales y finales). Los cálculos funcionales han sido extensivamente utilizados para construcción de programas [BdM 93][BD 77][Jeu 94][Mee] pero lamentablemente sus lenguajes de especificación no son suficientemente declarativos. De hecho, las especificaciones son descripciones de funciones recursivas parciales (y entonces programas en lenguajes funcionales), que son optimizadas en algún sentido durante el proceso de derivación. Por otra parte, encontrar una especificación funcional no es siempre una tarea fácil, y en general la distancia entre la formulación original y su especificación es bastante mayor que lo deseable. Las relaciones poseen algunas ventajas sobre las funciones como argumento de especificación y derivación de programas. Las relaciones poseen operaciones, como la conversa y el complemento, que ni siquiera están definidas en los contextos funcionales. Estas operaciones hacen a las relaciones más expresivas que las funciones y como consecuencia los ambientes relacionales permiten especificaciones más claras y declarativas. Las relaciones han sido utilizadas en diversos campos de las ciencias de la computación (notablemente, en Bases de Datos) y en particular en la construcción de programas. En [BH 93][BdM 93][DGB 97] se introducen relaciones utilizando conceptos categóricos y se define un álgebra para la construcción de programas. En [BK 97] se utiliza el cálculo relacional para la construcción de algoritmos sobre grafos. Otras aplicaciones de relaciones binarias en Ciencias de la Computación se reportan en el libro [BKS 97].

El cálculo ecuacional de la Fork Algebras ha sido utilizado en construcción de programas desde hace algunos años [B+ 96][FBH 96][HV 91]. En [B+ 96][FBH 96] se introducen los conceptos fundamentales para caracterizar estrategias de solución de problemas en el

contexto de la Teoría de Primer Orden de las Fork Álgebras. En [FBH 97] se propone una metodología de construcción de programas basada estrategias y algoritmos genéricos.

En este trabajo presentamos la caracterización de dos estrategias generales para desarrollar programas, apuntando a mejorar su eficiencia: **tupling** y **generalización**. Ambas técnicas han sido extensivamente estudiadas en el contexto de la programación funcional y lógica. En el contexto de la Teoría de Primer Orden de las Fork Álgebras, estas nociones aparecen como fórmulas de primer orden sobre relaciones y su aplicación a expresiones algorítmicas de una forma general, que corresponde a una amplia clase de programas recursivos, las transforma en nuevas expresiones que son fácilmente interpretadas como programas más eficientes. El artículo está organizado del siguiente modo. En la Sección 2 se presenta el fundamento teórico del trabajo: las Álgebras de Relaciones y las Fork Álgebras. En la sección 3 se presentan las estrategias de *tupling* y generalización y, finalmente, la Sección 4 está dedicada a las conclusiones y trabajos futuros.

2. Fork Álgebras

Las Fork Álgebras propias son extensiones de las álgebras de relaciones binarias con un nuevo operador llamado fork ($\underline{\nabla}$). Este operador induce una estructura en el dominio subyacente de las Fork Álgebras propias. Los objetos, en lugar de ser relaciones binarias sobre un conjunto plano, son relaciones binarias sobre un dominio estructurado.

Con el objeto de definir la clase de las Fork Álgebras Propias (PFA), se define primero la clase de Full$\star$PFA como sigue:

Definición 1. *Una Full$\star$PFA es una estructura con dominio $\mathcal{P}(U \times U)$ y U*

$$\left\langle \mathcal{P}(U \times U), U, \cup, \cap, , \emptyset, U \times U, |, Id, {}^{T}, \underline{\nabla}, \star \right\rangle$$

tal que

1. *$\langle \mathcal{P}(U \times U), \cup, \cap, , \emptyset, U \times U \rangle$ es un Álgebra de Boole,*
2. *$\langle \mathcal{P}(U \times U), |, Id \rangle$ es un semi-grupo, con elemento neutro Id.*
3. *$|, Id, {}^{T}$ y indican la composición de relaciones binarias, la identidad sobre U, la conversa y el complemento respecto de $U \times U$,*
4. *$\star : U \times U \to U$ es una función inyectiva,*
5. *$R \underline{\nabla} S = \{\langle x, \star(y, z)\rangle : xRy \wedge xSz\}$.*

Definición 2. *Se define FullPFA como RdFull$\star$PFA, donde Rd toma reductos del tipo de similaridad $\left\langle \cup, \cap, , \emptyset, U \times U, |, Id, {}^{T}, \underline{\nabla} \right\rangle$, y se define la clase PFA como SPFullPFA donde S toma subálgebras y P toma la clausura respecto del producto directo.*

Como una instancia particular de la aplicación del operador fork, se puede considerar la relación $Id \underline{\nabla} Id$, que en una fork álgebra propia puede verse como una operación que produce dos copias de un elemento. Esta relación es denotada por *2*.

Dado un par de relaciones binarias, la operación llamada *cross* y denotada $\otimes$, realiza una especie de producto paralelo, y su definición es

$$R \otimes S = \{\langle \star(x, y), \star(w, z)\rangle : xRw \wedge ySz\}$$

Al igual que las álgebras relacionales son una versión abstracta de las álgebras de relaciones binarias, las Fork Álgebras propias tienen su contraparte abstracta, las Fork Álgebras Abstractas (AFA):

Definición 3. *Una Fork Álgebra Abstracta es una estructura algebraica*

$$\left\langle R, +, \cdot, , 0, 1, ; , 1', {}^{T}, \nabla \right\rangle$$

que satisface los siguientes axiomas:

Ax.1 $x;(y;z) = (x;y);z$

Ax.2 $(x+y);z = x;z + y;z$

Ax.3 $(x+y)^T = x^T + y^T$

Ax.4 $\left(x^T\right)^T = x$

Ax.5 $x, 1' = 1';x = x$

Ax.6 $(x;y)^T = y^T;x^T$

Ax.7 $x;y \cdot z = 0$ *sii* $z;y^T \cdot x = 0$ *sii* $x^T;z \cdot y = 0$

Ax.8 $r \nabla s = (r;(1'\nabla 1)) \cdot (s;(1 \nabla 1'))$

Ax.9 $(r \nabla s);(t \nabla q)^T = \left(r;t^T\right) \cdot \left(s;q^T\right)$

Ax.10 $(1'\nabla 1)^T \nabla (1 \nabla 1')^T \preceq 1'$

Las relaciones $(1'\nabla 1)^T$ y $(1 \nabla 1')^T$ se comportan como proyecciones, proyectando componentes cons-truídos con la función inyectiva $\star$. Se denotan π y ρ respectivamente. Denotaremos por $\preceq$ al orden parcial inducido por el retículo $\langle R, +, \cdot \rangle$.

Definición 4. *Una relación F es llamada* ***funcional*** *si satisface la fórmula $F^T;F \preceq 1'$.*

Una relación I es llamada ***inyectiva*** *si satisface la fórmula $I;I^T \preceq 1'$.*

Una relación D es llamada ***ideal izquierdo*** *si satisface la condición $D = 1;D$ y es llamada* ***ideal derecho*** *si satisface $D = D;1$.*

Una relación C se llama ***constante*** *si es funcional, ideal izquierdo y satisface la condición $C;1 = 1$.*

2.1. Filtros y Conjuntos

Los filtros son identidades parciales, es decir, relaciones F que satisfacen la condición $F \preceq 1'$; son llamados filtros porque pueden ser usados para "filtrar" la información de entrada a una relación. Por ejemplo, si F es un filtro y R es una relación arbitraria entonces $F;R$ restringe la entrada de R a F.

Existe una clara relación entre filtros de Álgebras de Relaciones Binarias y conjuntos; un filtro F caracteriza unívocamente al conjunto $\{x : xFx\}$. Así mismo, un conjunto S caracteriza unívocamente a un filtro, la relación binaria $\{\langle x, x\rangle : x \in S\}$. Denotamos al filtro asociado al conjunto S como $1'_S$. Dado un filtro F, por $\neg F$ denotamos al término $\overline{F} \cdot 1'$. Nótese que si $F = 1'_S$ para algún conjunto S, entonces $\neg F = 1'_{\overline{S}}$, es el filtro asociado al complemento del conjunto S.

Los filtros son utilizados en Construcción de Programas para modelar guardas en construcciones similares al if-then-else o case. Consideremos el siguiente ejemplo:

```
function ISZERO (nat n) : bool
begin
    if n=0 then return(true)
           else return(false)
end;
```

Esta función puede ser representada relacionalmente por la ecuación siguiente:

$$ISZERO = 1'_{n=0}; C_{true} + 1'_{n>0}; C_{false}$$

donde $1'_{n=0}$ es el filtro $\{\langle 0,0\rangle\}$ y $1'_{n>0}$ es el filtro $\{\langle x,x\rangle : x > 0\}$.

Las propiedades más importantes de los filtros se encuentran en [FBH 97].

2.2. Propiedades Aritméticas de las Álgebras Relacionales y Fork Álgebras

Daremos a continuación algunas propiedades útiles para el cálculo de programas. Sus demostraciones y muchas otras propiedades pueden encontrarse en [FBH 97].

Teorema 1. *Las siguientes propiedades son válidas en cualquier álgebra de relaciones:*

1. $Dom(R); R = R$ *y* $R; Ran(R) = R$, *para toda relación* R.
2. *Cualesquiera sean* R *y* S,

$$Dom(R + S) = Dom(R) + Dom(S), \text{ y } Ran\,(R + S) = Ran\,(R) + Ran\,(S)\,.$$

3. *Si* $R \preceq 1'$ *entonces* $\breve{R} = R$ *(Es decir, las identidades parciales son relaciones simétricas).*
4. *Si* F *es una relación funcional entonces* $F; (R \cdot S) = (F; R) \cdot (F; S)$, *cualesquiera sean* R *y* S.
5. *Si* F *es una relación funcional,* $G \preceq F$ *y* $Dom(G) = Dom(F)$ *entonces* $G = F$.
6. $(R \cdot S)^T = R^T \cdot S^T$, *cualesquiera sean* R *y* S.
7. $(R + S)^T = R^T + S^T$, *cualesquiera sean* R *y* S.
8. *Si* I *es una relación inyectiva entonces* $(R \cdot S)\,; I = (R; I) \cdot (S; I)$, *cualesquiera sean las relaciones* R *y* S.■

Teorema 2. *Las siguientes propiedades son verdaderas en cualquier Fork Álgebra:*

1. $(R \otimes S) \cdot (T \otimes U) = (R \cdot T) \otimes (S \cdot U)$, *cualesquiera sean las relaciones* R, S, T *y* U.
2. $(R \bigtriangledown S)\,; (T \otimes U) = (R; T) \bigtriangledown (S; U)$, *cualesquiera sean las relaciones* R, S, T *y* U.
3. $(R \otimes S)\,; (T \otimes U) = (R; T) \otimes (S; U)$, *cualesquiera sean las relaciones* R, S, T *y* U.
4. *Las relaciones* π *y* ρ *son funcionales.*
5. *Si* F *es un filtro entonces* $F; R \bigtriangledown S = F; (R \bigtriangledown S)$, *cualesquiera sean* R *y* S.
6. *Si* F *es una relación funcional, entonces* $F; (S \bigtriangledown T) \preceq (F; S) \bigtriangledown (F; T)$, *cualesquiera sean* S *y* T.
7. $(R \bigtriangledown S)\,; \pi = Dom(S)\,; R$ *y* $(R \bigtriangledown S)\,; \rho = Dom(R)\,; S$, *cualesquiera sean* R *y* S.
8. $(R + S) \otimes T = (R \otimes T) + (S \otimes T)$ *y* $R \otimes (S + T) = (R \otimes S) + (R \otimes T)$, *cualesquiera sean* R, S *y* T.
9. $(R \otimes 1')\,; \pi = \pi; R$ *y* $(1' \otimes R)\,; \rho = \rho; R$, *para toda* R.
10. $(R \otimes S)^T = R^T \otimes S^T$, *cualesquiera sean* R *y* S.■

3. Estrategias de Transformación en Fork Álgebras

Una vez que se obtiene una especificación descriptiva de un problema, comienza la etapa de derivación de una solución algorítmica para esa especificación. Es claro que esto no puede realizarse siguiendo sólo un razonamiento matemático mecánico, sino que se necesita de ideas, intuición y experiencia.

Se necesita entonces de algunas direcciones que indiquen qué caminos seguir para alcanzar los algoritmos deseados. Las estrategias de transformación de programas muestran estas direcciones.

Una vez que se han conseguido especificaciones operacionales de los problemas, muchas veces es necesario mejorar su eficiencia. Las estrategias que daremos a continuación ayudan a derivar versiones más eficientes de especificaciones operacionales (recursivas) ya obtenidas.

3.1. Tupling

En el contexto de la transformación de programas lógicos, la estrategia denominada *Predicate Tupling* [PrP96] permite, al igual que las estrategias *Tupling* y *Composition* en el contexto de Programación Funcional, evitar las visitas múltiples a una misma estructura de datos y la construcción de estructuras de datos intermedias.

Esta estrategia consiste en seleccionar algunos átomos $A_1, \ldots, A_n$ en el cuerpo de una cláusula C. Se introduce un nuevo predicado $newp$ definido por:

$$newp(X_1, \ldots, X_k) \leftarrow A_1, \ldots, A_n$$

y luego se busca una definición recursiva para este predicado, realizando pasos de *unfolding*, transformaciones y *folding* usando la definición de $newp$. Generalmente esta estrategia se aplica cuando $A_1, \ldots, A_n$ comparten variables.

Aunque en el cálculo de programas basado en Fork álgebras no se tienen variables individuales, el hecho de compartir datos puede pensarse como la utilización del operador fork (∇) en algún lugar en la definición de una relación. Podemos definir entonces una estrategia similar a *Predicate Tupling* en el contexto relacional, que llamaremos simplemente *Tupling*, de la siguiente manera:

> *Dada una ecuación de la forma:*
>
> $$R = S; (R_1 \bigtriangledown \ldots \bigtriangledown R_n); T$$
>
> *Se define* $NEWR = (R_1 \bigtriangledown \ldots \bigtriangledown R_n)$ *y se busca una definición recursiva para* $NEWR$*,realizando pasos de unfolding, transformaciones y finalmente folding utilizando* $NEWR$*.*

Consideremos el problema de calcular el mínimo y el máximo de una lista de naturales. Supongamos tener las siguientes especificaciones, que calculan el mínimo y el máximo de una lista respectivamente:

$$\begin{aligned} MIN &= 1'_{L=1}; hd + 1'_{L>1}; (hd \bigtriangledown tl); (1' \otimes MIN); MINNUM \\ MAX &= 1'_{L=1}; hd + 1'_{L>1}; (hd \bigtriangledown tl); (1' \otimes MAX); MAXNUM \end{aligned}$$

donde hd retorna la cabeza de la lista argumento, tl la cola, y L representa su longitud, de modo que, por ejemplo $1'_{L=0}$ representa la identidad parcial de la lista vacía. A su vez, $MINNUM$ y $MAXNUM$ retornan el mínimo y el máximo entre dos elementos, respectivamente. Estas especificaciones son las definiciones recursivas comunes de MIN y MAX. La especificación de MIN, por ejemplo, consiste en retornar la cabeza de la

lista si ésta tiene longitud igual a uno, y retornar el mínimo entre la cabeza de la lista y el valor obtenido de aplicar MIN a la cola de la lista, si la longitud es mayor.

La solución más directa del problema dado sería $MINMAX = MIN \nabla MAX$. Una implementación natural de esta definición de $MINMAX$ produce un programa que recorre la lista argumento dos veces: una para calcular el mínimo y otra para calcular el máximo. Con la intención de obtener una definición de $MINMAX$ que recorra la lista sólo una vez, comenzaremos a transformar esta especificación utilizando la estrategia *tupling* propuesta. En este caso no es necesario definir una nueva relación para la aplicación de la estrategia, pues ésta coincidiría con $MINMAX$.

Introducimos entonces directamente los casos $L = 1$ y $L > 1$:

$$MINMAX = 1'_{L=1}; \begin{matrix} MIN \\ \triangledown \\ MAX \end{matrix} + 1'_{L>1}; \begin{matrix} MIN \\ \triangledown \\ MAX \end{matrix}$$

Dado que $1'_{L=1} \cdot 1'_{L>1} = 0$, y haciendo *unfold* con la definición de $MINMAX$ obtenemos

$$MINMAX = 1'_{L=1}; \begin{matrix} hd \\ \triangledown \\ hd \end{matrix} + 1'_{L>1}; \begin{matrix} (hd \triangledown tl); (1' \otimes MIN); MINNUM \\ \triangledown \\ (hd \triangledown tl); (1' \otimes MAX); MAXNUM \end{matrix}$$

y en virtud de la funcionalidad de $(hd \triangledown tl)$, y Teorema 2 (2):

$$MINMAX = 1'_{L=1}; \begin{matrix} hd \\ \triangledown \\ hd \end{matrix} + 1'_{L>1}; (hd \triangledown tl); \begin{matrix} (1' \otimes MIN) \\ \triangledown \\ (1' \otimes MAX) \end{matrix}; \begin{matrix} MINNUM \\ \otimes \\ MAXNUM \end{matrix}$$

Necesitamos poder llegar a $MIN \triangledown MAX$ para hacer *folding* usando $MINMAX$ y obtener así a una definición recursiva. Como se puede apreciar, la aplicación de *tupling* en el contexto relacional no es tan directa como en la Programación en Lógica y necesitaremos introducir relaciones adicionales para hacer la derivación más sencilla.

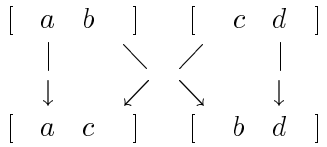
Obsérvese que las definiciones de MIN y MAX tienen la forma general

$$D_0; R_0 + D_1; (R_1 \nabla F_1); (A_1 \otimes A_2); J_1$$

que corresponde de una manera bastante directa a las construcciones categóricas conocidas como catamorfismos [MA 86][BdM97]. Un resultado bien conocido en las aplicaciones de la Teoría de Categorías a la programación es que cualquier par de catamorfismos basados en el mismo functor puede expresarse en términos de un único catamorfismo. Este resultado se conoce con el nombre de "banana split". La Proposición 1 (ver Sección 3.1.1) expresa un resultado en cierto modo equivalente en el contexto de las Fork Álgebras. Por el momento, a modo de evocación, introducimos el siguiente operador:

$$Banana = \begin{matrix} (\pi \otimes \pi) \\ \triangledown \\ (\rho \otimes \rho) \end{matrix}$$

cuyo funcionamiento intuitivo es el siguiente:



Esta relación será de gran utilidad en la derivación de soluciones utilizando la estrategia *tupling*. Utilizaremos la siguiente propiedad de esta relación, que demostraremos en la siguiente subsección:

$$\begin{array}{ccc} A \otimes B & & A \bigtriangledown C \\ \bigtriangledown & = & \otimes \\ C \otimes D & & B \bigtriangledown D \end{array} ; Banana$$

Utilizando esta propiedad, obtenemos:

$$MINMAX = 1'_{L=1}; \begin{array}{c} hd \\ \bigtriangledown \\ hd \end{array} + 1'_{L>1}; \begin{array}{c} hd \\ \bigtriangledown \\ tl \end{array} ; \begin{array}{c} (1' \bigtriangledown 1') \\ \otimes \\ (MIN \bigtriangledown MAX) \end{array} ; Banana; \begin{array}{c} MINNUM \\ \otimes \\ MAXNUM \end{array}$$

Ahora realizando *folding* de $MINMAX$ se obtiene:

$$MINMAX = 1'_{L=1}; \begin{array}{c} hd \\ \bigtriangledown \\ hd \end{array} + 1'_{L>1}; \begin{array}{c} hd \\ \bigtriangledown \\ tl \end{array} ; \begin{array}{c} (1' \bigtriangledown 1') \\ \otimes \\ MINMAX \end{array} ; Banana; \begin{array}{c} MINNUM \\ \otimes \\ MAXNUM \end{array}$$

Esta definición de $MINMAX$ nos da un programa que recorre sólo una vez su lista argumento.

3.1.1. Banana Split

En esta Subsección damos algunas propiedades de la relación $Banana$, cuyas demostraciones pueden encontrarse en el Apéndice A.

Lema 1. $Banana = Banana^T$. ■

Corolario 1. $Banana; Banana^T \preceq 1'$. ■

Lema 2. $2 \otimes (1' \bigtriangledown \rho) ; Banana = (1' \otimes 1') \bigtriangledown (1' \otimes \rho)$. ■

Teorema 3. $\begin{array}{ccc} A \otimes B & & A \bigtriangledown C \\ \bigtriangledown & = & \otimes \\ C \otimes D & & B \bigtriangledown D \end{array} ; Banana.$ ■

Proposición 1. *Sean*

$$P = D_0; R_0 + D_1; (R_1 \bigtriangledown F_1); (A_1 \otimes A_2); J_1 \ y$$
$$Q = D_0; S_0 + D_1; (R_1 \bigtriangledown F_1); (B_1 \otimes B_2); J_2,$$

donde $D_0, D_1 \preceq 1'$, $D_0 \cdot D_1 = 0$, R_1 *y* F_1 *son relaciones funcionales. Entonces*

$$P \bigtriangledown Q = D_0; \begin{array}{c} R_0 \\ \bigtriangledown \\ S_0 \end{array} + D_1; \begin{array}{c} R_1 \\ \bigtriangledown \\ F_1 \end{array} ; \begin{array}{c} A_1 \bigtriangledown B_1 \\ \otimes \\ A_2 \bigtriangledown B_2 \end{array} ; Banana; \begin{array}{c} J_1 \\ \otimes \\ J_2 \end{array} .$$ ■

Obsérvese que el ejemplo de $MINMAX$ se encuentra contemplado en esta proposición.

Lema 3. *Sean* F_1, F_2 *relaciones funcionales, donde* $Dom(F_1) \preceq Dom(F_2)$. *Entonces*

$$F_1; R = \begin{array}{c} F_1 \\ \bigtriangledown \\ F_2 \end{array} ; \begin{array}{c} R \\ \otimes \\ 1' \end{array} ; \pi = \begin{array}{c} F_2 \\ \bigtriangledown \\ F_1 \end{array} ; \begin{array}{c} 1' \\ \otimes \\ R \end{array} ; \rho$$

para cualquier relación R. ■

Corolario 2. *Sean*

$$\begin{aligned} P &= D_0; R_0 + D_1; F_1; A_1; J_1 \ y \\ Q &= D_0; S_0 + D_1; (R_1 \bigtriangledown F_1); (B_1 \otimes B_2); J_2, \end{aligned}$$

donde $D_0, D_1 \preceq 1'$, $D_0 \cdot D_1 = 0$, R_1 *y* F_1 *son relaciones funcionales tales que*

$$Dom(F_1) \preceq Dom(R_1).$$

Entonces

$$P \bigtriangledown Q = D_0; \begin{matrix} R_0 \\ \bigtriangledown \\ S_0 \end{matrix} + D_1; \begin{matrix} R_1 \\ \bigtriangledown \\ F_1 \end{matrix}; \begin{matrix} 1' \bigtriangledown B_1 \\ \otimes \\ A_1 \bigtriangledown B_2 \end{matrix}; Banana; \begin{matrix} \rho; J_1 \\ \otimes \\ J_2 \end{matrix}$$

y

$$Q \bigtriangledown P = D_0; \begin{matrix} S_0 \\ \bigtriangledown \\ R_0 \end{matrix} + D_1; \begin{matrix} R_1 \\ \bigtriangledown \\ F_1 \end{matrix}; \begin{matrix} B_1 \bigtriangledown 1' \\ \otimes \\ B_2 \bigtriangledown A_1 \end{matrix}; Banana; \begin{matrix} J_2 \\ \otimes \\ \rho; J_1 \end{matrix}. \quad \blacksquare$$

3.1.2. Ejemplo de la Aplicación de *Banana Split* en la Derivación utilizando *tupling*

Consideremos el problema de calcular el promedio de los elementos de una lista de números naturales. Una solución para este problema es la siguiente:

$$AVERAGE = 1'_{L=0}; C_0 + 1'_{L>0}; \begin{matrix} SUM \\ \bigtriangledown \\ LENGTH \end{matrix}; DIV$$

donde C_0 representa la constante cero y SUM suma todos los elementos de una lista de naturales:

$$SUM = 1'_{L=0}; C_0 + 1'_{L>0}; (hd \bigtriangledown tl); (1' \otimes SUM); add,$$

$LENGTH$ calcula la longitud de una lista:

$$LENGTH = 1'_{L=0}; C_0 + 1'_{L>0}; tl; LENGTH; succ$$

y DIV divide dos enteros.

Nuestra definición actual de $AVERAGE$ recorre dos veces la lista argumento, una para calcular la suma de sus valores y otra para calcular su longitud. Definimos $SL = SUM \bigtriangledown LENGTH$. El corolario anterior es aplicable tomando: $Q = SUM$, $P = LENGTH$, $D_0 = 1'_{L=0}$, $D_1 = 1'_{L>0}$, $S_0 = R_0 = C_0$, $R_1 = hd$, $F_1 = tl$, $A_1 = LENGTH$, $B_1 = 1'$, $B_2 = SUM$, $J_1 = succ$, $J_2 = add$, y obtenemos

$$SL = 1'_{L=0}; \begin{matrix} C_0 \\ \bigtriangledown \\ C_0 \end{matrix} + 1'_{L>0}; \begin{matrix} hd \\ \bigtriangledown \\ tl \end{matrix}; \begin{matrix} 1' \bigtriangledown 1' \\ \otimes \\ SUM \bigtriangledown LENGTH \end{matrix}; Banana; \begin{matrix} add \\ \otimes \\ \rho; succ \end{matrix}$$

Haciendo *folding* con la definición de SL obtenemos

$$SL = 1'_{L=0}; \begin{matrix} C_0 \\ \bigtriangledown \\ C_0 \end{matrix} + 1'_{L>0}; \begin{matrix} hd \\ \bigtriangledown \\ tl \end{matrix}; \begin{matrix} 1' \bigtriangledown 1' \\ \otimes \\ SL \end{matrix}; Banana; \begin{matrix} add \\ \otimes \\ \rho; succ \end{matrix}$$

que es una versión más eficiente de SL, pues recorre la lista sólo una vez.

Adviértase que en nuestra especificación actual de SL se realiza una copia del primer elemento de la lista innecesariamente. Aplicamos el Teorema 2 (2) y el Lema 2 para obtener:

$$SL = 1'_{L=0}; \begin{matrix} C_0 \\ \bigtriangledown \\ C_0 \end{matrix} + 1'_{L>0}; \begin{matrix} hd \\ \bigtriangledown \\ tl; SL \end{matrix} ; \begin{matrix} 1' \otimes \pi \\ \bigtriangledown \\ 1' \otimes \rho \end{matrix} ; \begin{matrix} add \\ \otimes \\ \rho; succ \end{matrix} =$$

$$= 1'_{L=0}; \begin{matrix} C_0 \\ \bigtriangledown \\ C_0 \end{matrix} + 1'_{L>0}; \begin{matrix} hd \\ \bigtriangledown \\ tl; SL \end{matrix} ; \begin{matrix} (1' \otimes \pi); add \\ \bigtriangledown \\ (1' \otimes \rho); \rho; succ \end{matrix}$$

que equivale a

$$1'_{L=0}; \begin{matrix} C_0 \\ \bigtriangledown \\ C_0 \end{matrix} + 1'_{L>0}; \begin{matrix} hd \\ \bigtriangledown \\ tl; SL \end{matrix} ; \begin{matrix} (1' \otimes \pi); add \\ \bigtriangledown \\ \rho; \rho; succ \end{matrix}$$

Hacemos *folding* de $SUM \bigtriangledown LENGTH$ en la definición de $AVERAGE$, luego *unfolding* de SL y obtenemos

$$AVERAGE = 1'_{L=0}; C_0 + \\ + 1'_{L>0}; \left(1'_{L=0}; \begin{matrix} C_0 \\ \bigtriangledown \\ C_0 \end{matrix} + 1'_{L>0}; \begin{matrix} hd \\ \bigtriangledown \\ tl; SL \end{matrix} ; \begin{matrix} (1' \otimes \pi); add \\ \bigtriangledown \\ \rho; \rho; succ \end{matrix} \right); DIV$$

y aplicando Ax.2, $1'_{L=0} \cdot 1'_{L>0} = 0$, conseguimos la siguiente versión final para $AVERAGE$:

$$AVERAGE = 1'_{L=0}; C_0 + 1'_{L>0}; \begin{matrix} hd \\ \bigtriangledown \\ tl; SL \end{matrix} ; \begin{matrix} (1' \otimes \pi); add \\ \bigtriangledown \\ \rho; \rho; succ \end{matrix} ; DIV.$$

3.2. Generalización

Cuando no puede llegarse en forma directa a una solución recursiva para un problema, muchas veces conviene introducir un problema más general, que abarque al anterior como caso particular. Este nuevo problema debe ser tal que permita derivar a partir de él una solución recursiva más directamente. Una vez obtenida una solución para el nuevo problema más general, definimos el problema original en función de esta generalización.

Esta estrategia es conocida como **generalización** o **embedding** [Par 90]. Consideraremos aquí esta estrategia, al igual que *tupling*, para obtener soluciones más eficientes a partir de definiciones operacionales.

3.2.1. Generalización de Rango

Consideremos el siguiente programa, que retorna verdadero si cada elemento de una lista es mayor que la suma de los elementos que le siguen:

$$STEEP = 1'_{L=0}; C_{true} + 1'_{L>0}; \begin{matrix} hd \\ \nabla \\ tl \end{matrix} ; \begin{matrix} 1' \\ \otimes \\ STEEP \nabla SUM \end{matrix} ; \begin{matrix} (1' \otimes \rho); > \\ \nabla \\ \rho; \pi \end{matrix} ; and$$

donde C_{true} representa la constante booleana verdadero, and es la conjunción lógica y $>$ representa la relación *"es mayor que"* para números naturales. Una implementación directa de $STEEP$ a partir de esta especificación toma tiempo cuadrático.

Para obtener una versión más eficiente de $STEEP$ aplicando la estrategia de *tupling* intentaríamos obtener una versión de $St = STEEP \nabla SUM$ que recorra sólo una vez la lista argumento; pero advertimos que St es una **generalización** de $STEEP$, por lo cual,

si logramos la definición buscada para St, la aprovecharemos para redefinir $STEEP$ como $St;\pi$.

Haciendo *unfold* de St y aplicando la Proposición 1:

$$St = 1'_{L=0}; (C_{true} \nabla C_0) + 1'_{L>0}; \begin{array}{c} hd \\ \nabla \\ tl \end{array} ; \begin{array}{c} 2 \\ \otimes \\ STEEP \nabla SUM \\ \bigtriangledown \\ SUM \end{array} ; Banana; \begin{array}{c} J \\ \otimes \\ add \end{array}$$

donde $J = (((1' \otimes \rho)\,; >) \bigtriangledown (\rho; \pi))\,; and$.

Hacemos ahora *folding* usando St y reescribimos SUM como $St;\rho$, y llegamos a

$$St = 1'_{L=0}; (C_{true} \nabla C_0) + 1'_{L>0}; \begin{array}{c} hd \\ \nabla \\ tl; St \end{array} ; \left(\begin{array}{c} 2 \\ \otimes \\ 1' \\ \bigtriangledown \\ \rho \end{array} \right) ; Banana; \begin{array}{c} J \\ \otimes \\ add \end{array}$$

y, aplicando el Lema 2 obtenemos la siguiente versión final de St:

$$St = 1'_{L=0}; (C_{true} \nabla C_0) + 1'_{L>0}; \begin{array}{c} hd \\ \nabla \\ tl; St \end{array} ; \begin{array}{c} J \\ \bigtriangledown \\ (1' \otimes \rho)\,; add \end{array} .$$

Si tomáramos directamente $St;\pi$ como definición de $STEEP$, tendríamos ya una versión más eficiente (St es lineal); pero podemos trabajar aún más sobre esta definición:

$$\begin{aligned} STEEP &= St;\pi = \left[1'_{L=0}; (C_{true} \nabla C_0) + 1'_{L>0}; \begin{array}{c} hd \\ \nabla \\ tl; St \end{array} ; \begin{array}{c} J \\ \bigtriangledown \\ (1' \otimes \rho)\,; add \end{array} \right] ; \pi = \\ &= 1'_{L=0}; (C_{true} \nabla C_0)\,; \pi + 1'_{L>0}; \begin{array}{c} hd \\ \nabla \\ tl; St \end{array} ; \begin{array}{c} J \\ \bigtriangledown \\ (1' \otimes \rho)\,; add \end{array} ; \pi = \\ &= 1'_{L=0}; C_{true} + 1'_{L>0}; \begin{array}{c} hd \\ \nabla \\ tl; St \end{array} ; \mathrm{Dom}\left((1' \otimes \rho)\,; add\right); J = \\ &= 1'_{L=0}; C_{true} + 1'_{L>0}; \begin{array}{c} hd \\ \nabla \\ tl; St \end{array} ; \left(\begin{array}{c} 1'_N \\ \otimes \\ (1' \otimes 1'_N) \end{array} \right) ; \begin{array}{c} ((1' \otimes \rho)\,; >) \\ \bigtriangledown \\ (\rho; \pi) \end{array} ; and \end{aligned}$$

lo que nos permite llegar a la siguiente versión final de $STEEP$:

$$STEEP = 1'_{L=0}; true + 1'_{L>0}; \begin{array}{c} hd \\ \nabla \\ tl; St \end{array} ; \begin{array}{c} ((1' \otimes \rho)\,; >) \\ \bigtriangledown \\ (\rho; \pi) \end{array} ; and.$$

3.2.2. Generalización de Dominio

Consideremos la especificación de MIN dada anteriormente:

$$MIN = 1'_{L=1}; hd + 1'_{L>1}; (hd \bigtriangledown tl); (1' \otimes MIN); MINNUM$$

Buscando mejorar la eficiencia de MIN, intentaremos derivar a partir de esta especificación una definición recursiva a la cola. Para esto, definimos $GENMIN = (1' \otimes MIN); MINNUM$, y buscamos una definición recursiva a la cola de $GENMIN$.

Hacemos *unfolding* de MIN:

$$GENMIN = \left[\begin{array}{c} 1' \\ \otimes \\ 1'_{L=1}; hd + 1'_{L>1}; (hd \bigtriangledown tl); (1' \otimes MIN); MINNUM \end{array}\right]; MINNUM$$

No es difícil probar que esta definición es equivalente a:

$$\begin{aligned} GENMIN &= \left[\begin{array}{c} 1' \\ \otimes \\ 1'_{L=1}; hd \end{array}\right]; MINNUM + \\ &\quad + \left[\begin{array}{ccc} & & 1' \\ & & \otimes \\ & hd & 1' \\ 1'_{L>1}; & \bigtriangledown \; ; & \otimes \quad ; MINNUM \\ & tl & MIN \end{array}\right] \end{aligned}$$

que en virtud de la asociatividad de $MINNUM$ queda

$$\begin{aligned} GENMIN &= \left[\begin{array}{c} 1' \\ \otimes \\ 1'_{L=1}; hd \end{array}\right]; MINNUM + \\ &\quad + \left[\begin{array}{ccc} & 1' & \\ & \otimes & \\ & hd & \\ 1'_{L>1}; & \bigtriangledown \; ; & \end{array}\right]; lassoc; \left[\begin{array}{cc} 1' & \\ \otimes & ; MINNUM \\ 1' & \\ & \otimes \\ & 1' \end{array}\right]; \left[\begin{array}{c} 1' \\ \otimes \\ MIN \end{array}\right]; MINNUM \end{aligned}$$

donde $lassoc = ((\pi \bigtriangledown \rho; \pi) \bigtriangledown (\rho; \rho))$. Hacemos *folding* a la cola usando $GENMIN$ y obtenemos:

$$\begin{aligned} GENMIN &= \left[\begin{array}{c} 1' \\ \otimes \\ 1'_{L=1}; hd \end{array}\right]; MINNUM + \\ &\quad + \left[\begin{array}{ccc} & 1' & \\ & \otimes & \\ & hd & \\ 1'_{L>1}; & \bigtriangledown \; ; & \end{array}\right]; lassoc; \left[\begin{array}{c} MINNUM \\ \otimes \\ 1' \end{array}\right]; GENMIN \end{aligned}$$

Realizamos algunas transformaciones más para eliminar $lassoc$, y obtenemos finalmente

$$\begin{aligned} GENMIN &= \left[\begin{array}{c} 1' \\ \otimes \\ 1'_{L=1}; hd \end{array}\right]; MINNUM + \\ &\quad + \left[\begin{array}{c} 1' \\ \otimes \\ 1'_{L>1} \end{array}\right]; \left[\begin{array}{c} (1' \bigtriangledown \rho; hd) \\ \bigtriangledown \\ \rho; tl \end{array}\right]; \left[\begin{array}{c} MINNUM \\ \otimes \\ 1' \end{array}\right]; GENMIN \end{aligned}$$

La definición de MIN queda:

$$MIN = 1'_{L=1}; hd + 1'_{L>1}; (hd \bigtriangledown tl); GENMIN$$

Esta definición corresponde al programa imperativo:

```
function MIN (natsequ s) : nat
begin
  if length(s)=1 then return(hd(s))
                 else
                   X:=hd(s);
                   Y:=tl(s);
                   while length(Y)>1 do
                       X:=MINNUM(X, hd(Y));
                       Y:=tl(Y)
                   endwhile;
                   return(MINNUM(X, hd(Y)))
end;
```

Esta derivación es una aplicación del siguiente resultado general, cuya demostración es similar a la derivación anterior:

Teorema 4. *Sea*

$$P = D_0; R_0 + D_1; (R_1 \nabla F_1; P)\,; R$$

donde $D_0, D_1 \preceq 1'$, $D_0 \cdot D_1 = 0$, *y* R *verifica:*

$$(1' \otimes R); R = lassoc; (R \otimes 1')\,; R \qquad \textit{(Levo-Asociatividad de } R\textit{)}$$

Entonces,

$$P = D_0; R_0 + D_1; (R_1 \nabla F_1)\,; GENP$$

donde

$$GENP = (1' \otimes D_0; R_0)\,; R + (1' \otimes D_1)\,; \begin{bmatrix} (1' \bigtriangledown \rho; R_1)\,; R \\ \bigtriangledown \\ \rho; F_1 \end{bmatrix}; GENP. \blacksquare$$

Ejemplo: Este último Teorema es aplicable a MAX para obtener una versión recursiva a la cola, tomando $D_0 = 1'_{L=1}$, $D_1 = 1'_{L>1}$, $R_0 = hd$, $R_1 = hd$, $F_1 = tl$, y $R = MAXNUM$. Obtenemos como nueva versión para MAX:

$$\begin{aligned} MAX &= 1'_{L=1}; hd + 1'_{L>1}; (hd \bigtriangledown tl); GENMAX \\ GENMAX &= \begin{bmatrix} 1' \\ \otimes \\ 1'_{L=1}; hd \end{bmatrix}; MAXNUM + \\ &\quad + \begin{bmatrix} 1' \\ \otimes \\ 1'_{L>1} \end{bmatrix}; \begin{bmatrix} (1' \bigtriangledown \rho; hd) \\ \bigtriangledown \\ \rho; tl \end{bmatrix}; \begin{bmatrix} MAXNUM \\ \otimes \\ 1' \end{bmatrix}; GENMAX \end{aligned}$$

Como $GENMAX = (1' \otimes MAX)\,; MAXNUM$ y el operador $MAXNUM$ posee elemento neutro (C_0), podemos escribir MAX como

$$MAX = (C_0 \bigtriangledown 1')\,; GENMAX$$

que corresponde al siguiente programa imperativo:

```
function MAX (natsequ s) : nat
begin
  X:=0;
  Y:=s;
```

```
    while length(Y)>1 do
      X:=MAXNUM(X, hd(Y));
      Y:=tl(Y)
    endwhile;
    return(MAXNUM(X, hd(Y)))
  end;
```

4. Conclusiones

En este trabajo hemos presentado algunas estrategias de transformación de programas que permiten mejorar la eficiencia de especificaciones relacionales recursivas.

En el desarrollo formal de programas se comienza a partir de una especificación descriptiva sencilla para luego pasar a una especificación operacional, aunque no eficiente. Los siguientes pasos del desarrollo consisten en transformarla en una especificación operacional eficiente. Las estrategias propuestas en este trabajo corresponden a estas transformaciones finales (de operacional ineficiente a eficiente) y se complementan perfectamente con la metodología propuesta en [FBH 97]. En ese trabajo el objetivo es obtener soluciones recursivas (Divide & Conquer) para especificaciones declarativas. La combinación de ambas estrategias sienta las bases para establecer una metodología que permita abarcar gran parte del proceso de desarrollo de programas a partir de especificaciones formales.

Referencias

[B+ 96] Baum, G.A., Frías, M.F., Haeberer, A.M. and Martínez López, P.E., *From Specifications to Programs: A Fork-algebraic Approach to Bridge the Gap*, in Proceedings of MFCS '96, LNCS 1113, Springer-Verlag, 180-191, 1996.

[BD 77] Burstall, R.M. and Darlington, J., *A Transformation System for Developing Recursive Programs*, Journal of the ACM, Vol. 24, No. 1, 44-67, 1997.

[BdM 93] Bird, R. and de Moor, O., *List Partitions*, Formal Aspects of Computing, Vol. 5, No. 1, 67-78, 1993.

[BdM97] Bird, R. and de Moor, O., *Algebra of Programming*, Prentice Hall, 1997.

[BH 93] Backhouse, R.C. and Hoogendijk, P., *Elements of a Relational Theory of Datatypes*, Formal Program Development, IFIP TC2/WG 2.1 State-of-the-Art Report, LNCS 755, Springer-Verlag, 7-42, 1993.

[BK 97] Berghammer, R. and von Karger, *Algorithms from Relational Specifications*, Chapter 9 of [BKS 97].

[BKS 97] Brink, C., Kahl, W. and Schmidt, G. (Eds.), *Relational Methods in Computer Science*, Springer Wien-New York, 1997.

[DGB 97] Doornbos, H., van Gasteren, N. and Backhouse, R.C., *Programs and Datatypes*, Chapter 10 of [BKS 97].

[FBH 96] Frías, M.F., Baum, G.A. and Haeberer, A.M., *Adding Design Strategies to Fork Algebras*, in Perspectives of System Informatics, LNCS 1181, Springer-Verlag, 214-226, 1996.

[FBH 97] Frías, M.F., Baum, G.A. and Haeberer, A.M., *A Calculus for Program Construction Based on Fork Algebras and Generic Algorithms.*

[FBH 98b] Frías, M.F., Baum, G.A. and Haeberer, A.M., *Representability and Program Construction within Fork Algebras*, in Journal of IGPL, 1998.

[HV 91] Haeberer, A.M. and Veloso, P.A.S., *Partial Relations for Program Derivation: Adequacy, Inevitability and Expressiveness*, in Proceedings of the IFIP TC2 Working Conference on Constructing Programs from Specifications, North Holland, 319-3

[Jeu 94] Jeuring, J.T., *The Derivation of On-Line Algorithms with an Application to Finding Palindromes*, Algorithmica, Vol. 11, No. 2, 146-184, 1994.

[MA 86] Manes, E.G. and Arbib, M.A., *Algebraic Approaches to Program Semantics*, Texts and Monographs in Computer Science, Springer-Verlag, 1986.

[Mee] Meertens, L., *Algorithmics - Toward Programming as a Mathematical Activity*, in De Bakker, J.W., Hazewinkel, M. and Lenstra, J.K., (Eds.), *Mathematics and Computer Science*, Vol. 1 of CWI Monographs, 3-42, North Holland.

[Par 90] Partsch, H.A., *Specification and Transformation of Programs. A Formal Approach to Software Development*, Texts and Monographs in Computer Science, Springer-Verlag, 1990.

[PrP96] Pettorossi, A. and Proietti, M., *Rules and Strategies for Transforming Functional and Logic Programs*, ACM Computing Surveys, Vol.28, No.2, June 1996.71, 1991.

5. Apéndice A

En este apéndice se incluyen las demostraciones de los resultados que consideramos más relevantes del artículo.

Lema 1: $Banana = Banana^T$.

D) Por Def. Banana y Ax.8:

$$Banana^T = ((\pi \otimes \pi) \bigtriangledown (\rho \otimes \rho))^T = ((\pi \otimes \pi); \pi^T \cdot (\rho \otimes \rho); \rho^T)^T$$

Por Teorema 1 (6), Ax.6 y Teorema 2 (10) respectivamente, obtenemos

$$((\pi \otimes \pi); \pi^T \cdot (\rho \otimes \rho); \rho^T)^T = \pi; (\pi \otimes \pi)^T \cdot \rho; (\rho \otimes \rho)^T = \pi; (\pi^T \otimes \pi^T) \cdot \rho; (\rho^T \otimes \rho^T)$$

que por Def.$\otimes$, es igual a

$$\pi; (\pi; \pi^T \bigtriangledown \rho; \pi^T) \cdot \rho; (\pi; \rho^T \bigtriangledown \rho; \rho^T) = \pi; (\pi; \pi^T; \pi^T \cdot \rho; \pi^T; \rho^T) \cdot \rho; (\pi; \rho^T; \pi^T \cdot \rho; \rho^T; \rho^T)$$

Aplicando Teorema 1 (4) se obtiene

$$(\pi; \pi; \pi^T; \pi^T \cdot \pi; \rho; \pi^T; \rho^T) \cdot (\rho; \pi; \rho^T; \pi^T \cdot \rho; \rho; \rho^T; \rho^T)$$

Asociamos ahora convenientemente, aplicamos Teorema 1 (8),

$$\begin{array}{l} (\pi; \pi; \pi^T \cdot \rho; \pi; \rho^T); \pi^T \cdot (\pi; \rho; \pi^T \cdot \rho; \rho; \rho^T); \rho^T = \\ \quad (\pi; \pi; \pi^T \cdot \rho; \pi; \rho^T) \qquad (\pi; \pi \bigtriangledown \rho; \pi) \\ = \qquad \bigtriangledown \qquad = \qquad \bigtriangledown \\ \quad (\pi; \rho; \pi^T \cdot \rho; \rho; \rho^T) \qquad (\pi; \rho \bigtriangledown \rho; \rho) \end{array}$$

Por Def.$\otimes$, esto último equivale a $Banana$.■

Teorema 3: $\begin{matrix} A\otimes B \\ \bigtriangledown \\ C\otimes D \end{matrix} = \begin{matrix} A\bigtriangledown C \\ \otimes \\ B\bigtriangledown D \end{matrix} ; Banana.$

D)

$$\begin{matrix} A\bigtriangledown C \\ \otimes \\ B\bigtriangledown D \end{matrix} ; Banana = \left(\begin{matrix} A\bigtriangledown C \\ \otimes \\ B\bigtriangledown D \end{matrix} ; Banana \right)^{T^T} = \left(Banana; \begin{matrix} (A\bigtriangledown C)^T \\ \otimes \\ (B\bigtriangledown D)^T \end{matrix} \right)^T =$$

$$= \left(\begin{matrix} (\pi\otimes\pi) \\ \bigtriangledown \\ (\rho\otimes\rho) \end{matrix} ; \begin{matrix} (A\bigtriangledown C)^T \\ \otimes \\ (B\bigtriangledown D)^T \end{matrix} \right)^T = \left(\begin{matrix} (\pi\otimes\pi);(A\bigtriangledown C)^T \\ \bigtriangledown \\ (\rho\otimes\rho);(B\bigtriangledown D)^T \end{matrix} \right)^T =$$

$$= \left[(\pi\otimes\pi);(A\bigtriangledown C)^T ;\pi^T\cdot(\rho\otimes\rho);(B\bigtriangledown D)^T ;\rho^T \right]^T =$$

$$= \left((\pi\otimes\pi);(A\bigtriangledown C)^T ;\pi^T \right)^T \cdot \left((\rho\otimes\rho);(B\bigtriangledown D)^T ;\rho^T \right)^T =$$

$$= \pi;(A\bigtriangledown C);(\pi\otimes\pi)^T\cdot\rho;(B\bigtriangledown D);(\rho\otimes\rho)^T =$$

$$= \pi;\left(A;\pi^T\bigtriangledown C;\pi^T\right)\cdot\rho;\left(B;\rho^T\bigtriangledown D;\rho^T\right) =$$

$$= \left(\pi;A;\pi^T\bigtriangledown\pi;C;\pi^T\right)\cdot\left(\rho;B;\rho^T\bigtriangledown\rho;D;\rho^T\right) =$$

$$= \left(\pi;A;\pi^T;\pi^T\cdot\pi;C;\pi^T;\rho^T\right)\cdot\left(\rho;B;\rho^T;\pi^T\cdot\rho;D;\rho^T;\rho^T\right) =$$

$$= \begin{matrix} \pi;A;\pi^T\cdot\rho;B;\rho^T \\ \bigtriangledown \\ \pi;C;\pi^T\cdot\rho;D;\rho^T \end{matrix} = \begin{matrix} \pi;A\bigtriangledown\rho;B \\ \bigtriangledown \\ \pi;C\bigtriangledown\rho;D \end{matrix} = \begin{matrix} A\otimes B \\ \bigtriangledown \\ C\otimes D \end{matrix} .■$$

Proposición 1: Sean

$$\begin{aligned} P &= D_0;R_0+D_1;(R_1\bigtriangledown F_1);(A_1\otimes A_2);J_1 \text{ y} \\ Q &= D_0;S_0+D_1;(R_1\bigtriangledown F_1);(B_1\otimes B_2);J_2, \end{aligned}$$

donde $D_0, D_1 \preceq 1'$, $D_0\cdot D_1 = 0$, R_1 y F_1 son relaciones funcionales. Entonces

$$P\bigtriangledown Q = D_0;\begin{matrix} R_0 \\ \bigtriangledown \\ S_0 \end{matrix} + D_1;\begin{matrix} R_1 \\ \bigtriangledown \\ F_1 \end{matrix} ;\begin{matrix} A_1\bigtriangledown B_1 \\ \otimes \\ A_2\bigtriangledown B_2 \end{matrix} ; Banana;\begin{matrix} J_1 \\ \otimes \\ J_2 \end{matrix} .$$

D) En virtud de nuestras hipótesis $D_0, D_1 \preceq 1'$, $D_0\cdot D_1 = 0$, tenemos que

$$P\bigtriangledown Q = D_0;(P\bigtriangledown Q) + D_1;(P\bigtriangledown Q)$$

Hacemos *unfolding* de P y Q, y haciendo algunas transformaciones sencillas obtenemos

$$P\bigtriangledown Q = D_0;(R_0\bigtriangledown S_0) + D_1;\begin{matrix} (R_1\bigtriangledown F_1);(A_1\otimes A_2);J_1 \\ \bigtriangledown \\ (R_1\bigtriangledown F_1);(B_1\otimes B_2);J_2 \end{matrix}$$

que, en virtud de la funcionalidad de $(R_1\bigtriangledown F_1)$, equivale a

$$D_0;(R_0\bigtriangledown S_0) + D_1;(R_1\bigtriangledown F_1);\begin{matrix} (A_1\otimes A_2);J_1 \\ \bigtriangledown \\ (B_1\otimes B_2);J_2 \end{matrix}$$

Por Teorema 2 (2) tenemos

$$P\bigtriangledown Q = D_0;(R_0\bigtriangledown S_0) + D_1;(R_1\bigtriangledown F_1);\begin{matrix} (A_1\otimes A_2) \\ \bigtriangledown \\ (B_1\otimes B_2) \end{matrix} ;\begin{matrix} J_1 \\ \otimes \\ J_2 \end{matrix}$$

que por el Teorema 3, equivale a

$$D_0;(R_0 \bigtriangledown S_0) + D_1;(R_1 \bigtriangledown F_1);\ \begin{matrix}(A_1 \bigtriangledown B_1)\\ \otimes \\ (A_2 \bigtriangledown B_2)\end{matrix};Banana;\ \begin{matrix}J_1\\ \otimes \\ J_2\end{matrix}\ \blacksquare$$

Lema 3: Sean F_1, F_2 relaciones funcionales, donde $\mathrm{Dom}(F_1) \preceq \mathrm{Dom}(F_2)$. Entonces

$$F_1;R = \begin{matrix}F_1\\ \bigtriangledown \\ F_2\end{matrix}\ ;\ \begin{matrix}R\\ \otimes \\ 1'\end{matrix};\pi = \begin{matrix}F_2\\ \bigtriangledown \\ F_1\end{matrix}\ ;\ \begin{matrix}1'\\ \otimes \\ R\end{matrix};\rho$$

para cualquier relación R.

D) Por Teorema 2 (2) tenemos que

$$\begin{matrix}F_1\\ \bigtriangledown \\ F_2\end{matrix}\ ;\ \begin{matrix}R\\ \otimes \\ 1'\end{matrix};\pi = \begin{matrix}F_1;R\\ \bigtriangledown \\ F_2;1'\end{matrix};\pi$$

que por Teorema 2 (7) equivale a

$$\mathrm{Dom}(F_2;1');F_1;R$$

que es equivalente a

$$\mathrm{Dom}(F_2);F_1;R$$

en virtud de la Totalidad de $1'$; luego, por nuestra hipótesis $\mathrm{Dom}(F_1) \preceq \mathrm{Dom}(F_2)$:

$$\begin{matrix}F_1\\ \bigtriangledown \\ F_2\end{matrix}\ ;\ \begin{matrix}R\\ \otimes \\ 1'\end{matrix};\pi = F_1;R$$

La demostración de que $F_1;R = \begin{matrix}F_2\\ \bigtriangledown \\ F_1\end{matrix}\ ;\ \begin{matrix}1'\\ \otimes \\ R\end{matrix};\rho$ es análoga.$\blacksquare$