Decibilidad de Isomorfismos de Tipos

Carlos Martínez Méndez

Escuela de Ingeniería en Computación
Facultad de Ciencias Físicas y Matemáticas
Universidad de Central de Chile

Junio 04, 2008

Implementación Fold

Consideremos la función `fold` sobre listas en el contexto de *lenguajes de programación funcional*, vemos que en los siguientes lenguajes considerados, esta función ha sido nombrada y *tipeada* de manera distinta:

Implementación Fold

Consideremos la función `fold` sobre listas en el contexto de *lenguajes de programación funcional*, vemos que en los siguientes lenguajes considerados, esta función ha sido nombrada y *tipeada* de manera distinta:

Lenguaje	Nombre	Tipos
SML	foldr	$(A \times B \to B) \to B \to LIST[A] \to B[$
Ocaml	List.fold_left	$(B \to A \to B) \to B \to LIST[A] \to B$
CAML	list_it	$(A \to B \to B) \to LIST[A] \to B \to B$
Haskell	foldl	$(B \to A \to B) \to B \to LIST[A] \to B$

La función `foldr` en SML

```
fun foldr   (f: ('a * 'b) -> 'b) (base : 'b) (l: 'a list): 'b =
case l of
            [] => b
|           x::xs => f (x,foldr f base xs)
```

¿ Qué hace esta la función `fold` de listas?

```
val foldr          :  ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
```

¿ Qué hace esta la función `fold` de listas?

```
val foldr          :  ('a * 'b -> 'b) -> 'b -> 'a list -> 'b

fun largo xs       =  foldr (fn (_, n) => n+1) 0 xs
```

¿ Qué hace esta la función `fold` de listas?

```
val foldr          :  ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
```

```
fun largo xs       =  foldr (fn (_, n) => n+1) 0 xs

fun sumatoria xs   =  foldr (fn (x: int, y: int) => x+y) 0 xs
```

¿ Qué hace esta la función `fold` de listas?

```
val foldr          :  ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
```

```
fun largo xs       =  foldr (fn (_, n) => n+1) 0 xs

fun sumatoria xs   =  foldr (fn (x: int, y: int) => x+y) 0 xs

fun pitatoria xs   =  foldr (fn (x: int, y: int) => x*y) 1 xs
```

¿ Qué hace esta la función `fold` de listas?

```
val foldr          :  ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
```

```
fun largo xs       =  foldr (fn (_, n) => n+1) 0 xs

fun sumatoria xs   =  foldr (fn (x: int, y: int) => x+y) 0 xs

fun pitatoria xs   =  foldr (fn (x: int, y: int) => x*y) 1 xs

fun reversa xs     =  foldr (fn (x, y) => y  [x] ) [] xs
```

El cuestionamiento

Las preguntas pertinentes

i) ¿ Comó buscamos esta función en una libreria en SML?,

El cuestionamiento

Las preguntas pertinentes

i) ¿ Comó buscamos esta función en una libreria en SML?, Sabiendo el nombre exacto, podemos usar este como la llave de búsqueda, sino...

El cuestionamiento

Las preguntas pertinentes

i) ¿ Comó buscamos esta función en una libreria en SML?, Sabiendo el nombre exacto, podemos usar este como la llave de búsqueda, sino...

ii) ¿ Qué hacemos en ese caso de no saber el nombre exacto?

El cuestionamiento

Las preguntas pertinentes

i) ¿ Comó buscamos esta función en una libreria en SML?, Sabiendo el nombre exacto, podemos usar este como la llave de búsqueda, sino...

ii) ¿ Qué hacemos en ese caso de no saber el nombre exacto? Buscamos por su *tipo*

El cuestionamiento

Las preguntas pertinentes

i) ¿ Comó buscamos esta función en una libreria en SML?, Sabiendo el nombre exacto, podemos usar este como la llave de búsqueda, sino...

ii) ¿ Qué hacemos en ese caso de no saber el nombre exacto? Buscamos por su *tipo*

iii) ¿ Es esto suficiente?

El cuestionamiento

Las preguntas pertinentes

i) ¿ Comó buscamos esta función en una libreria en SML?, Sabiendo el nombre exacto, podemos usar este como la llave de búsqueda, sino...

ii) ¿ Qué hacemos en ese caso de no saber el nombre exacto? Buscamos por su *tipo*

iii) ¿ Es esto suficiente? Tampoco es suficiente, sin embargo, de las declaraciones de tipos asociados a `fold` podemos apreciar ciertas *similaridades.*

fold y tipos

Lenguaje	Nombre	Tipos
SML	foldr	$(A \times B \to B) \to B \to LIST[A] \to B$
Ocaml	List.fold_left	$(B \to A \to B) \to B \to LIST[A] \to B$
CAML	list_it	$(A \to B \to B) \to LIST[A] \to B \to B$
Haskell	foldl	$(B \to A \to B) \to B \to LIST[A] \to B$

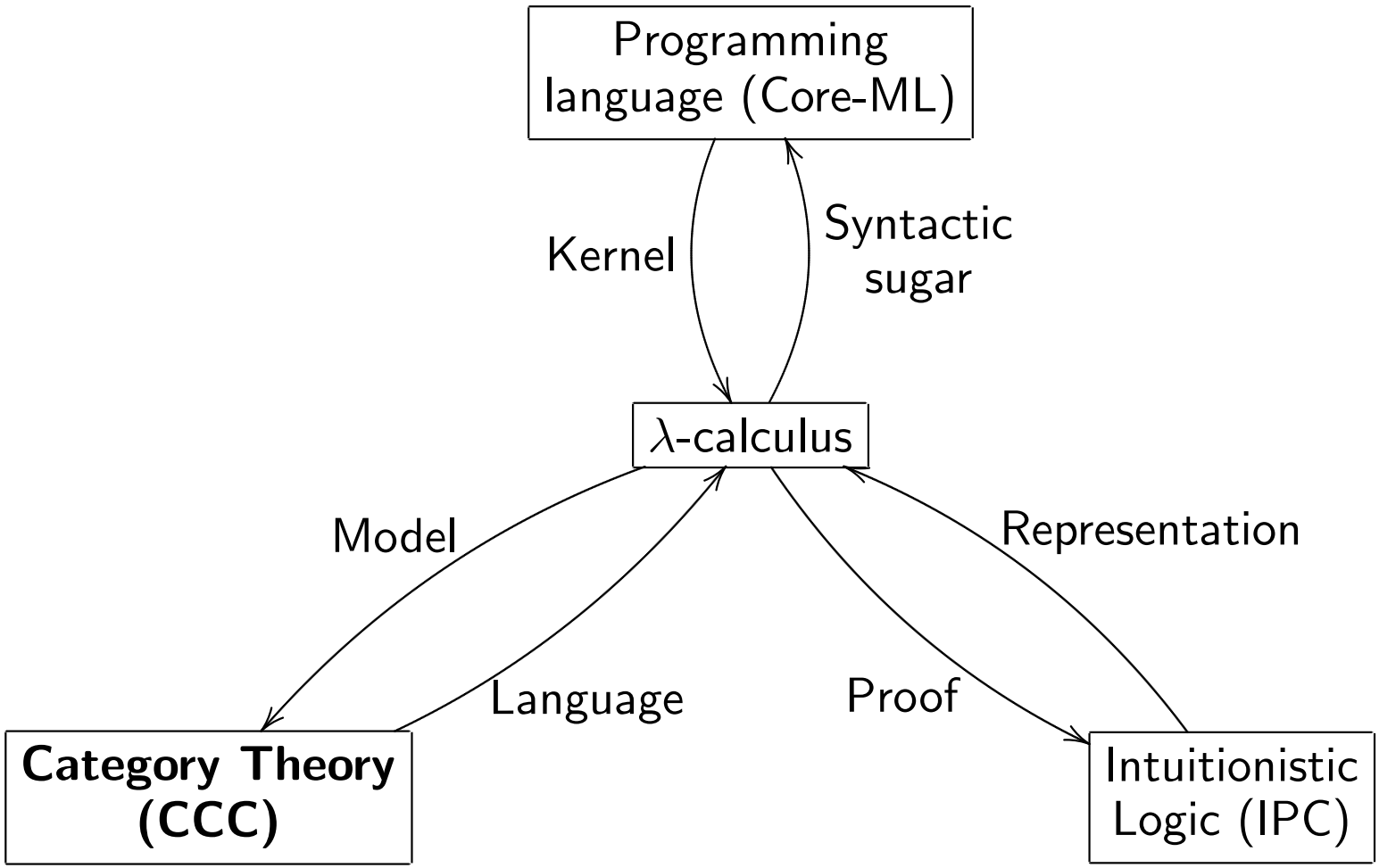
¿ Qué observamos?

fold y tipos

Lenguaje	Nombre	Tipos
SML	foldr	$(A \times B \to B) \to B \to LIST[A] \to B$
Ocaml	List.fold_left	$(B \to A \to B) \to B \to LIST[A] \to B$
CAML	list_it	$(A \to B \to B) \to LIST[A] \to B \to B$
Haskell	foldl	$(B \to A \to B) \to B \to LIST[A] \to B$

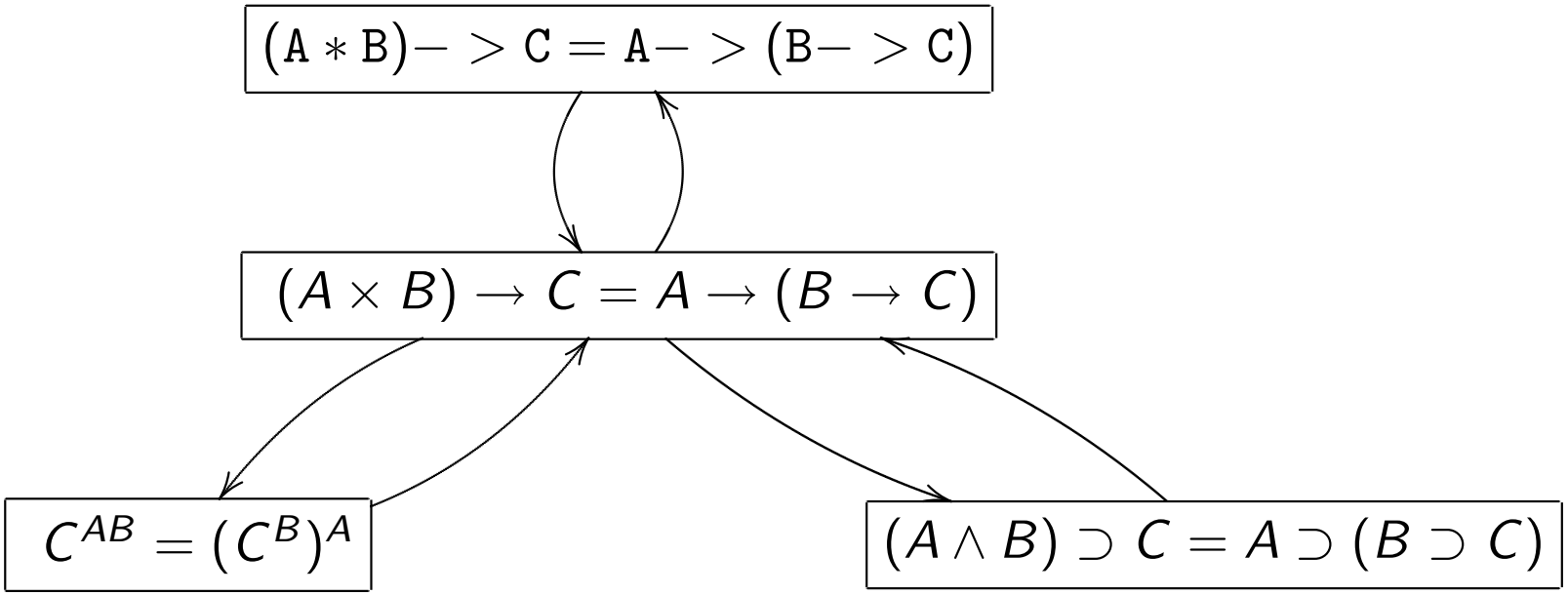
¿ Qué observamos? Un *álgebra de tipos.*

'The Frame'


Programming language (Core-ML)
Kernel
Syntactic sugar
λ-calculus
Model
Representation
Language
Proof
Category Theory (CCC)
Intuitionistic Logic (IPC)

Dicho de otro modo

Ejemplo: Curry



El Álgebra de los Tipos

Definición

La siguiente es un conjunto de ecuaciones (identidades) $\mathcal{T}$, que carácterizan lo que denominamos un sistema de Isomorfismos de Tipos:

$$
\begin{array}{ll}
(A0) & A \to (B \to C) = B \to (A \to C) \ \textit{(Swap)} \\
(A1) & A \times B = B \times A \\
(A2) & A \times (B \times C) = (A \times B) \times C \\
(A3) & (A \times B) \to C = A \to (B \to C) \\
(A4) & C \to (A \times B) = (C \to A) \times (C \to B) \\
(A5) & A \times \mathrm{T} = A \\
(A6) & A \to \mathrm{T} = \mathrm{T} \\
(A7) & \mathrm{T} \to A = A
\end{array}
$$

Intuiciones y otras preguntas

i) ¿ Qué determinad estas igualdades?

Intuiciones y otras preguntas

i) ¿ Qué determinad estas igualdades? Debemos pensar en que cada tipo es un conjunto de programas que satisfacen dicha especificación, con esto en mente estamos hablando de algún *'sentido de igualdad'* de conjuntos de programas.

Intuiciones y otras preguntas

i) ¿ Qué determinad estas igualdades? Debemos pensar en que cada tipo es un conjunto de programas que satisfacen dicha especificación, con esto en mente estamos hablando de algún *'sentido de igualdad'* de conjuntos de programas.

ii) Por ejemplo ¿Estamos hablando de igualdad conjuntista?

Intuiciones y otras preguntas

i) ¿ Qué determinad estas igualdades? Debemos pensar en que cada tipo es un conjunto de programas que satisfacen dicha especificación, con esto en mente estamos hablando de algún *'sentido de igualdad'* de conjuntos de programas.

ii) Por ejemplo ¿Estamos hablando de igualdad conjuntista? No, sin embargo, estamos hablando de que cada programa en un tipo, tiene un *programa copia* en el otro tipo.

Intuiciones y otras preguntas

i) ¿ Qué determinad estas igualdades? Debemos pensar en que cada tipo es un conjunto de programas que satisfacen dicha especificación, con esto en mente estamos hablando de algún *'sentido de igualdad'* de conjuntos de programas.

ii) Por ejemplo ¿Estamos hablando de igualdad conjuntista? No, sin embargo, estamos hablando de que cada programa en un tipo, tiene un *programa copia* en el otro tipo.

iii) ¿ Qué relación hay entre estos programas?

Intuiciones y otras preguntas

i) ¿ Qué determinad estas igualdades? Debemos pensar en que cada tipo es un conjunto de programas que satisfacen dicha especificación, con esto en mente estamos hablando de algún *'sentido de igualdad'* de conjuntos de programas.

ii) Por ejemplo ¿Estamos hablando de igualdad conjuntista? No, sin embargo, estamos hablando de que cada programa en un tipo, tiene un *programa copia* en el otro tipo.

iii) ¿ Qué relación hay entre estos programas? Tema para otra charla!

Desafíos

i) Queremos buscar programas en una librería, con la llave de búsqueda siendo el tipo del programa deseado.

Desafíos

i) Queremos buscar programas en una librería, con la llave de búsqueda siendo el tipo del programa deseado.

ii) Debemos entonces, construir mecanismos que nos permitán, **decidir** cuando dos tipos son isomorfos.

Desafíos

i) Queremos buscar programas en una librería, con la llave de búsqueda siendo el tipo del programa deseado.

ii) Debemos entonces, construir mecanismos que nos permitán, **decidir** cuando dos tipos son isomorfos.

iii) Escalar los resultados parcialmente logrados.

Decisión

Definición

(Problemas de Decisión) *Sea* **Q** *una clase de preguntas que tienen por respuesta* **Sí** *, o bien* **No**. *Decimos que la clase* **Q** *es decidible si y sólo si existe* **Alg** *un algoritmo, tal que para toda pregunta* $q \in \mathbf{Q}$, $\mathbf{Alg}(q) \in \{\mathbf{Sí}, \mathbf{No}\}$.

El problema de decisión para Isomorfismos de Tipos

Definición

(PD Isomorfismos de Tipos) *Sean T y S dos tipos sobre $\mathcal{T}$, queremos decidir efectivamente si $T =_{\mathcal{T}} S$*

Reescritura

La aproximación escogida se basa en las técnicas desarrolladas en lo que se ha llamado **Sistemas de Reescritura de Términos**.

Reescritura

La aproximación escogida se basa en las técnicas desarrolladas en lo que se ha llamado **Sistemas de Reescritura de Términos**. Estableceremos un sistema de reescritura $\mathcal{R}$ *completo* [**confluente**, **fuertemente normalizable**] y *equivalente* con la especificación ecuacional $\mathcal{T}$,

El sistema de Rescritura $\mathcal{R}$ para $\mathcal{T}$

Definición

(R2) $A \times (B \times C) > (A \times B) \times C$

(R3) $(A \times B) \to C > A \to (B \to C)$

(R4) $C \to (A \times B) > (C \to A) \times (C \to B)$

(R5) $A \times \mathrm{T} > A$

(R5)' $\mathrm{T} \times A > A$

(R6) $A \to \mathrm{T} > \mathrm{T}$

(R7) $\mathrm{T} \to A > A$

Una representación más intuitiva:

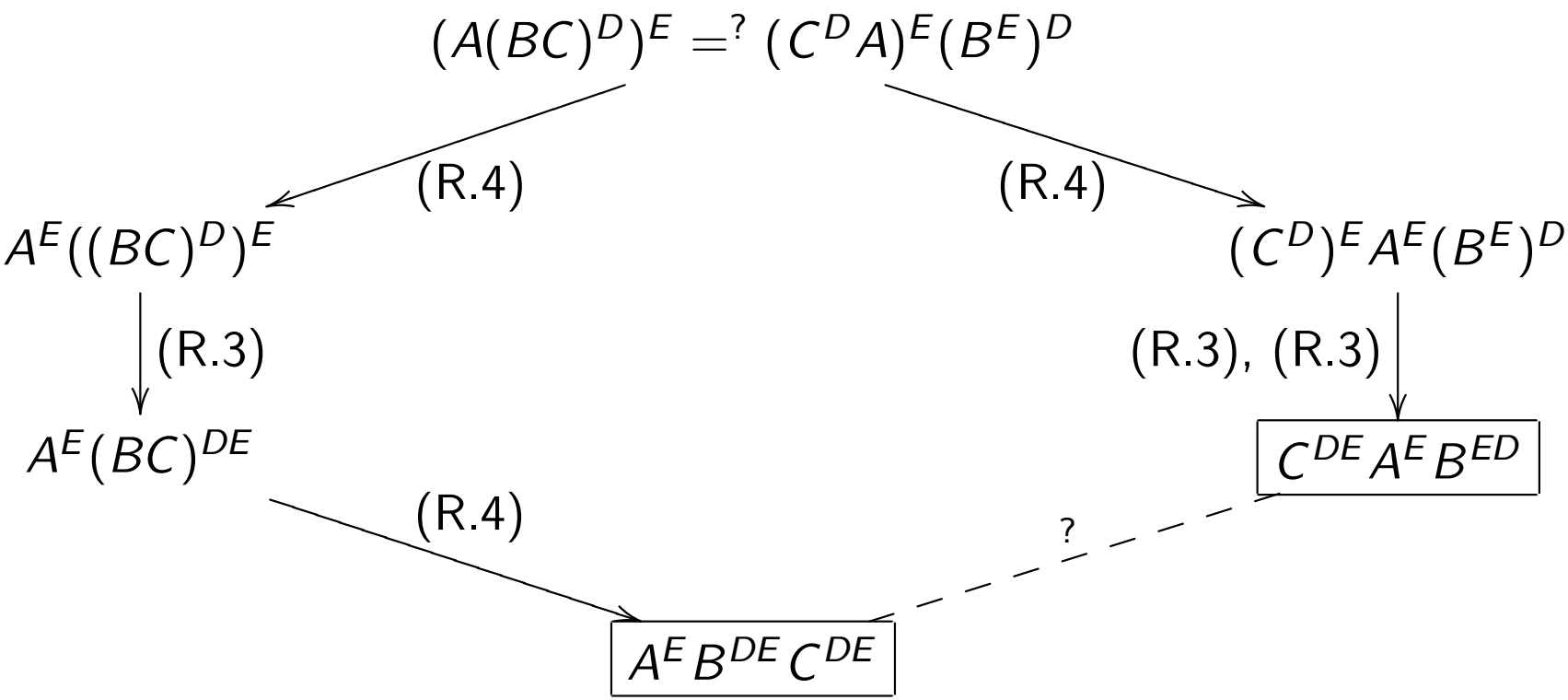
Como expresiones aritméticas:

Una representación más intuitiva:

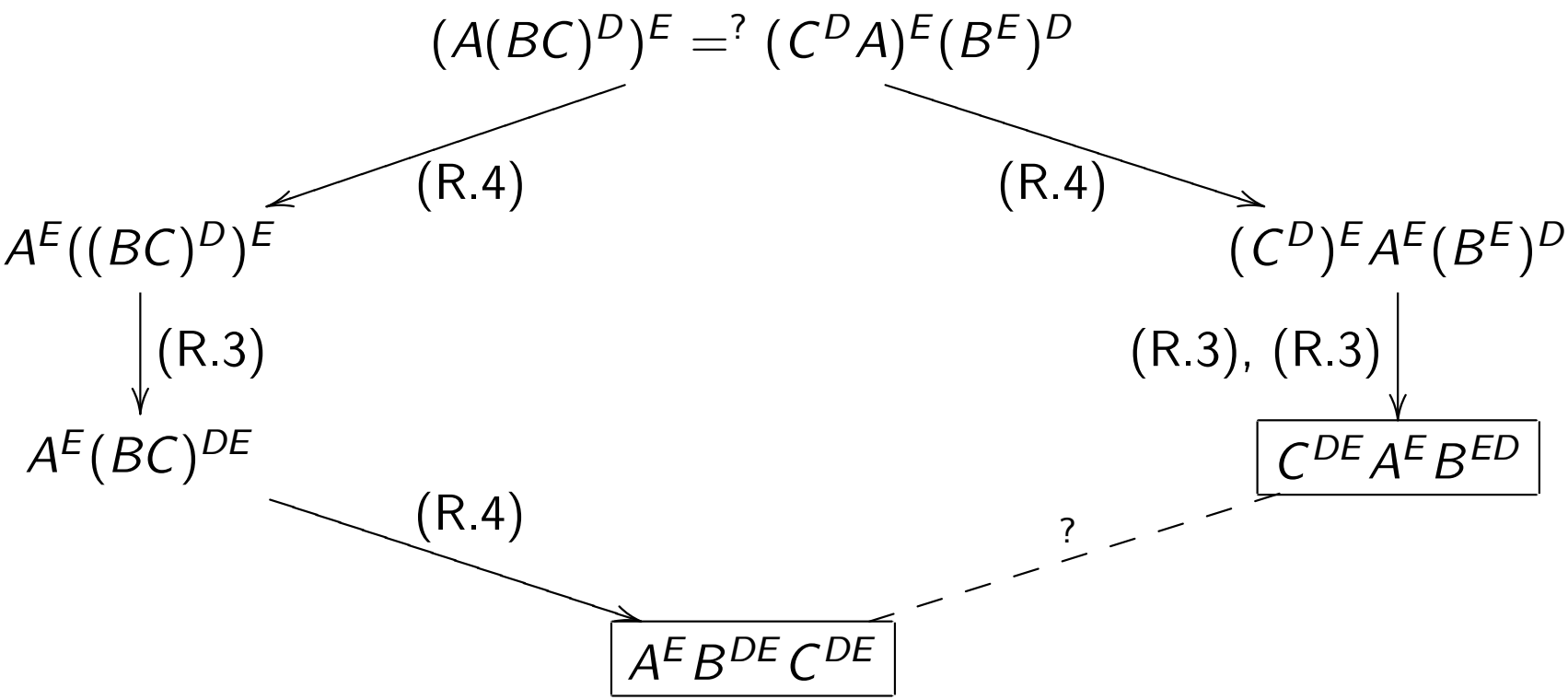
Como expresiones aritméticas:

$$
\begin{array}{ll}
\text{(R2)} & A(BC) > (AB)C \\
\text{(R3)} & (C^B)^A > C^{AB} \\
\text{(R4)} & (AB)^C > (A^C)(B^C) \\
\text{(R5)} & A1 > A \\
\text{(R5)'} & 1A > A \\
\text{(R6)} & A^1 > A \\
\text{(R7)} & 1^A > 1
\end{array}
$$

Ejemplo:


$(A(BC)^D)^E =^? (C^D A)^E (B^E)^D$
(R.4)
(R.4)
$A^E((BC)^D)^E$
$(C^D)^E A^E (B^E)^D$
(R.3)
(R.3), (R.3)
$A^E(BC)^{DE}$
$C^{DE} A^E B^{ED}$
(R.4)
?
$A^E B^{DE} C^{DE}$

Ejemplo:



Finalmente reordenamos productos: $C^{DE} A^E B^{ED} \rightarrow A^E B^{DE} C^{DE}$

Referencias

Bruce K., Di Cosmo R., Longo G.
Provable Isomorphism of Types, *Mathematical Structures in Computer Science*, **2**, 231 - 247, 1991.

Dezani-Ciancaglini M.
Characterization of Normal Forms Possessing Inverse in the $\lambda_{\beta\eta}$-Calculus, *Theoretical Computer Science*, **2**, 323 - 337, 1976.

Di Cosmo R.
Isomorphism of Types: from λ-calculus to information retrieval and language design, *Birkhäuser*, 1995.

Referencias

Rittri M.
Retrieving Library Functions by Unifying Types Modulo Linear Type Isomorphism, *Theoretical Informatics and Applications*, **27**, 71 -89, 1993.

Soloviev S.
The Category of Finite Sets and Cartesian Closed Categories, *Journal of Soviet Mathematics*, **22(3)**, 1387 - 1400, 1983.

Zibin Y., Gil J., Considine J.
Efficient Algorithm for Isomorphism of Simple Types, *Proceedings POPL'03 in ACM SIGPLAN*, **38**, 160 - 171, 2003.