Teoría de Tipos intensional de Martin-Löf

Juan Pablo García Garland

18 de noviembre, 2016

Table of contents

Tipos

"A type is explained by saying what an object of the type is and what it means for two objects of the types to be identical" [NPS00].

Tipos

Se definen con:

- Reglas de Introduccción.
- Reglas de Igualdad.

Juicios

- A Type
- $a \in A$
- $a = b \in A$

Juicios

- A Type
- $a \in A$
- $a = b \in A$
- $A = B$

Significado

A **Type**	$a \in A$
A es un conjunto (Tipo)	a es un elemento del conjunto (Tipo) A

Significado

A **Type**	$a \in A$
A es un conjunto	a es un elemento del conjunto A
A es una proposición	a es una prueba de A

Significado

A **Type**	$a \in A$
A es un conjunto	a es un elemento del conjunto A
A es una proposición	a es una prueba de A
A es una especificación	a es un programa que satisface A

Significado

A **Type**	$a \in A$
A es un conjunto	a es un elemento del conjunto A
A es una proposición	a es una prueba de A
A es una especificación	a es un programa que satisface A
A es un problema	a es una solución para A

Reglas Generales

Sobre objetos:

Reglas Generales

Sobre objetos:

- Reflexividad:

$$\frac{a \in A}{a = a \in A}$$

Reglas Generales

Sobre objetos:

- Reflexividad:

$$\frac{a \in A}{a = a \in A}$$

- Simetría:

$$\frac{a = b \in A}{b = a \in A}$$

Reglas Generales

Sobre objetos:

- Reflexividad:

$$\frac{a \in A}{a = a \in A}$$

- Simetría:

$$\frac{a = b \in A}{b = a \in A}$$

- Transitividad:

$$\frac{a = b \in A \qquad b = c \in A}{a = c \in A}$$

Reglas Generales

Sobre objetos:

- Reflexividad:

$$\frac{a \in A}{a = a \in A}$$

- Simetría:

$$\frac{a = b \in A}{b = a \in A}$$

- Transitividad:

$$\frac{a = b \in A \qquad b = c \in A}{a = c \in A}$$

Sobre Tipos:

- Reflexividad:

$$\frac{}{A = A}$$

- Simetría:

$$\frac{A = B}{B = A}$$

- Transitividad:

$$\frac{A = B \qquad B = C}{A = C}$$

Más Reglas Generales

Reglas de igualdad de tipos:

$$\frac{a \in A \qquad A = B}{a \in B}$$

$$\frac{a = b \in A \qquad A = B}{a = b \in B}$$

Juicios Hipotéticos

En su forma más general los cuatro juicios pueden depender de algunas hipótesis.

Juicios Hipotéticos

En su forma más general los cuatro juicios pueden depender de algunas hipótesis. Se introduce la noción de contexto.

$$\Gamma := x_1 \in A_1, ..., x_n \in A_n$$

Juicios Hipotéticos

En su forma más general los cuatro juicios pueden depender de algunas hipótesis. Se introduce la noción de contexto.

$$\Gamma := x_1 \in A_1, ..., x_n \in A_n$$

Escribimos:
A type $[\Gamma]$
$a \in A$ $[\Gamma]$
$A = A'$ $[\Gamma]$
$a = b \in A$ $[\Gamma]$

Juicios Hipotéticos

Tipos Funcionales (Function Types)

Una de las formas primitivas de constuir nuevos tipos a partir de otros, será introduciendo tipos funcionales

Tipos Funcionales (Function Types)

Una de las formas primitivas de constuir nuevos tipos a partir de otros, será introduciendo tipos funcionales (dependientes).

Tipos Funcionales (Function Types)

Una de las formas primitivas de constuir nuevos tipos a partir de otros, será introduciendo tipos funcionales (dependientes). Dado un tipo A, y una familia de tipos B indizados por A, construimos el conjunto funcional (dependiente) $(x \in A)B$ de funciones de A a B.

Tipos Funcionales (Function Types)

Una de las formas primitivas de constuir nuevos tipos a partir de otros, será introduciendo tipos funcionales (dependientes).
Dado un tipo A, y una familia de tipos B indizados por A, construimos el conjunto funcional (dependiente) $(x \in A)B$ de funciones de A a B.
Qué necesitamos?

Tipos Funcionales (Function Types)

Una de las formas primitivas de constuir nuevos tipos a partir de otros, será introduciendo tipos funcionales (dependientes).
Dado un tipo A, y una familia de tipos B indizados por A, construimos el conjunto funcional (dependiente) $(x \in A)B$ de funciones de A a B.
Qué necesitamos?

- Reglas de constucción
- Reglas de igualdad

Tipos Funcionales (Function Types)

Tipo funcional:

$$\frac{A \text{ type} \qquad B \text{ type } [x \in A]}{(x \in A)B \text{ type}}$$

Tipos Funcionales (Function Types)

Tipo funcional:

$$\frac{A \text{ type} \qquad B \text{ type } [x \in A]}{(x \in A)B \text{ type}}$$

Igualdad de tipos funcionales:

$$\frac{A = A' \qquad B = B'[x \in A]}{(x \in A)B = (x \in A')B'}$$

Notación

Notación

- Escribimos $(A)B$ para denotar $(x \in A)B$ cuando el producto no es dependiente (tipo flecha usual).

Notación

- Escribimos $(A)B$ para denotar $(x \in A)B$ cuando el producto no es dependiente (tipo flecha usual).
- Escribimos $(x \in A, y \in B)C$ para denotar $(x \in A)((y \in B)C)$

Notación

- Escribimos $(A)B$ para denotar $(x \in A)B$ cuando el producto no es dependiente (tipo flecha usual).
- Escribimos $(x \in A, y \in B)C$ para denotar $(x \in A)((y \in B)C)$
- Escribimos $(x, y \in A)C$ para denotar $(x \in A)((y \in A)C)$

Abstracción

Abstracción

Abstracción:

$$\frac{b \in B[x \in A]}{([x \in A]b) \in (x \in A)B}$$

Más Reglas

Reglas de Aplicación:

Más Reglas

Reglas de Aplicación:

$$\frac{c \in (x \in A)B \qquad a \in A}{c(a) \in B[a/x]}$$

$$\frac{c \in (x \in A)B \qquad a = b \in A}{c(a) = c(b) \in B[a/x]}$$

Reglas sobre igualdad de funciones

Reglas sobre igualdad de funciones

Aplicación:

$$\frac{c = d \in (x \in A)B \qquad a \in A}{c(a) = d(a) \in B[a/x]}$$

Reglas sobre igualdad de funciones

Aplicación:

$$\frac{c = d \in (x \in A)B \qquad a \in A}{c(a) = d(a) \in B[a/x]}$$

Extensionalidad:

$$\frac{c \in (x \in A)B \qquad d \in (x \in A)B \qquad c(x) = d(x) \in B[x \in A]}{c = d \in (x \in A)B}$$

Notación

Notación

- Escribimos $c(a_1, a_2, ...a_n)$ para denotar $c(a_1)(a_2)..(a_n)$

Notación

- Escribimos $c(a_1, a_2, ...a_n)$ para denotar $c(a_1)(a_2)..(a_n)$
- Escribimos $[x_1 \in A_1, x_2 \in A_2, ..x_n \in A_n]b$ para denotar $[x_1 \in A_1][x_2 \in A_2]..[x_n \in A_n]b$

Computando

Computando

- β-conversión:

$$\frac{a \in A \qquad b \in B[x \in A]}{([x \in A]b)(a) = b[a/x] \in B[a/x]}$$

Computando

- β-conversión:

$$\frac{a \in A \qquad b \in B[x \in A]}{([x \in A]b)(a) = b[a/x] \in B[a/x]}$$

Las siguientes reglas se pueden probar a partr de las anteriores:

Computando

- β-conversión:

$$\frac{a \in A \qquad b \in B[x \in A]}{([x \in A]b)(a) = b[a/x] \in B[a/x]}$$

Las siguientes reglas se pueden probar a partr de las anteriores:

- η-conversión:

$$\frac{c \in (x \in A)B}{([x \in a]c(x)) = c \in (x \in A)B}$$

Computando

- β-conversión:

$$\frac{a \in A \qquad b \in B[x \in A]}{([x \in A]b)(a) = b[a/x] \in B[a/x]}$$

Las siguientes reglas se pueden probar a partr de las anteriores:

- η-conversión:

$$\frac{c \in (x \in A)B}{([x \in a]c(x)) = c \in (x \in A)B}$$

- ξ-rule:

$$\frac{b = d \in B[x \in A]}{([x \in A]b = [x \in A]d \in (x \in A)B}$$

El tipo *Set*

La teoría de tipos intensional introduce un tipo particular, *Set*, cuyos objetos son los conjuntos definidos inductivamente.

El tipo *Set*

La teoría de tipos intensional introduce un tipo particular, *Set*, cuyos objetos son los conjuntos definidos inductivamente.
Set es un tipo, entonces..

El tipo *Set*

La teoría de tipos intensional introduce un tipo particular, *Set*, cuyos objetos son los conjuntos definidos inductivamente.
Set es un tipo, entonces..
Tenemos que definir qué significa ser un *Set* (reglas de construcción) y cuando dos objetos de *Set* son iguales (reglas de igualdad).

El tipo *Set*

La teoría de tipos intensional introduce un tipo particular, *Set*, cuyos objetos son los conjuntos definidos inductivamente.
Set es un tipo, entonces..
Tenemos que definir qué significa ser un *Set* (reglas de construcción) y cuando dos objetos de *Set* son iguales (reglas de igualdad).

Construyendo Sets

Se construyen mediante la introducción de constantes.

Construyendo Sets

Se construyen mediante la introducción de constantes.
Los objetos canónicos de tipo *Set* son de la forma:

$$c(a_1, ..., a_n) \in Set$$

en donde $c \in (x_1 \in A_1, ..., x_n \in A_n)Set$ es un *constructor de conjuntos*

Construyendo Sets

Ejemplos:

$$bool \in Set$$

$$nat \in Set$$

$$prod \in (Set, Set)Set$$

$$\Sigma \in (A : Set, (A)Set)Set$$

Formación de *Set*, Formación de *El*

Introducimos la siguiente reglas:

Formación de *Set*, Formación de *El*

Introducimos la siguiente reglas:
Formación de *Set*:

$$\frac{}{Set \text{ type}}$$

Formación de *Set*, Formación de *El*

Introducimos la siguiente reglas:
Formación de *Set*:

$$\frac{}{Set \text{ type}}$$

Para cada objeto A de *Set* se define un tipo $El(A)$ cuyos objetos son los elementos del *Set* A.

Formación de *Set*, Formación de *El*

Introducimos la siguiente reglas:
Formación de *Set*:

$$\frac{}{Set \text{ type}}$$

Para cada objeto A de *Set* se define un tipo $El(A)$ cuyos objetos son los elementos del *Set* A.
Formación de *El*:

$$\frac{a \in Set}{El(a) \text{ type}}$$

Set es abierto

Set es abierto

- El tipo *Set* tiene definición abierta.

Set es abierto

- El tipo *Set* tiene definición abierta. i.e. puede extenderse con nuevas definiciones.

Set es abierto

- El tipo *Set* tiene definición abierta. i.e. puede extenderse con nuevas definiciones.
- Los tipos $El(a)$ son cerrados, dado que las nuevas definiciones en *Set* son inductivas.

Notación

Notación

- Escribimos a en lugar de $El(a)$.

Notación

- Escribimos a en lugar de $El(a)$.
  El objeto $a \in Set$ y el tipo $El(a)$ siempre ocurren en contextos distintos, por lo que no hay ambigüedad.

Un conjunto (o familia) de conjuntos inductivos se define introduciendo las siguientes constantes:

Un conjunto (o familia) de conjuntos inductivos se define introduciendo las siguientes constantes:

- Un *constructor de conjunto*

$$C \in (\vec{x} \in \vec{A})Set$$

Un conjunto (o familia) de conjuntos inductivos se define introduciendo las siguientes constantes:

- Un *constructor de conjunto*

$$C \in (\vec{x} \in \vec{A})Set$$

- Constructores $c_1, c_2, ..., c_n$ que definen los objetos de tipo $El(C(\vec{a}))$

Un conjunto (o familia) de conjuntos inductivos se define introduciendo las siguientes constantes:

- Un *constructor de conjunto*

$$C \in (\vec{x} \in \vec{A})Set$$

- Constructores $c_1, c_2, ..., c_n$ que definen los objetos de tipo $El(C(\vec{a}))$
- Un destructor genérico d asociado a la familia C.

Un conjunto (o familia) de conjuntos inductivos se define introduciendo las siguientes constantes:

- Un *constructor de conjunto*

$$C \in (\vec{x} \in \vec{A})Set$$

- Constructores $c_1, c_2, ..., c_n$ que definen los objetos de tipo $El(C(\vec{a}))$
- Un destructor genérico d asociado a la familia C.
- Reglas de igualdad que describen el comportamiento de d respecto de los constructores.

Ejemplos de Sets

El conjunto de los valores booleanos

Definimos al set de los valores Booleanos.

El conjunto de los valores booleanos

Definimos al set de los valores Booleanos.
Es un ejemplo de conjunto enumerado (es trivialmente recursivo).

El conjunto de los valores booleanos

Definimos al set de los valores Booleanos.
Es un ejemplo de conjunto enumerado (es trivialmente recursivo).

Declarando *bool*

Declárense las siguientes constantes:

Declarando *bool*

Declárense las siguientes constantes:
El constructor del conjunto:

Declarando *bool*

Declárense las siguientes constantes:
El constructor del conjunto:

$$bool \in Set$$

Declarando *bool*

Declárense las siguientes constantes:
El constructor del conjunto:

$$bool \in Set$$

Los constructores:

Declarando *bool*

Declárense las siguientes constantes:
El constructor del conjunto:

$$bool \in Set$$

Los constructores:

$$true \in bool$$
$$false \in bool$$

Declarando *bool*

Declárense las siguientes constantes:
El constructor del conjunto:

$$bool \in Set$$

Los constructores:

$$true \in El(bool)$$
$$false \in El(bool)$$

Destructor de *bool*

Destructor de *bool*

La constante para la destrucción de los Booleanos:

Destructor de *bool*

La constante para la destrucción de los Booleanos:

$$boolrec \in (P \in (bool)Set, P(true), P(false), b \in bool)P(b)$$

Destructor de *bool*

La constante para la destrucción de los Booleanos:

$$boolrec \in (P \in (bool)Set, P(true), P(false), b \in bool)P(b)$$

Las reglas de igualdad:

$$boolrec(P, P_t, P_f, true) = P_t \in P(true)$$
$$boolrec(P, P_t, P_f, false) = P_f \in P(false)$$

Destructor de *bool*

La constante para la destrucción de los Booleanos:

$$boolrec \in (P \in (El(bool))Set, El(P(true)), El(P(false)), b \in El(bool))El(P(b))$$

Las reglas de igualdad:

$$boolrec(P, P_t, P_f, true) = P_t \in P(true)$$
$$boolrec(P, P_t, P_f, false) = P_f \in P(false)$$

En particular, podemos construir a partir de este un caso mas débil, no dependiente:

$$boolelim \in (P \in Set, P, P, bool)P$$

En particular, podemos construir a partir de este un caso mas débil, no dependiente:

$$boolelim \in (P \in Set, P, P, bool)P$$

Se construye como:

$$boolelim := [P \in Set]boolrec([\_ \in bool]P)$$

En particular, podemos construir a partir de este un caso mas débil, no dependiente:

$$boolelim \in (P \in Set, P, P, bool)P$$

Se construye como:

$$boolelim := [P \in Set]boolrec([\_ \in bool]P)$$

Corresponde a la primitiva if-then-else.

Declarando *unit*

Definimos al set de un único habitante.

Declarando *unit*

Definimos al set de un único habitante.

Declarando *unit*

Declárense las siguientes constantes:

Declarando *unit*

Declárense las siguientes constantes:
El constructor del conjunto:

Declarando *unit*

Declárense las siguientes constantes:
El constructor del conjunto:

$$unit \in Set$$

Declarando *unit*

Declárense las siguientes constantes:
El constructor del conjunto:

$$unit \in Set$$

La constante para el constructor:

Declarando *unit*

Declárense las siguientes constantes:
El constructor del conjunto:

$$unit \in Set$$

La constante para el constructor:

$$<> \in unit$$

Selectores de *unit*

Selectores de *unit*

La constante para la selección en *unit* (destructor):

Selectores de *unit*

La constante para la selección en *unit* (destructor):

$$unitrec \in (P \in (unit)Set, P(<>), u \in unit)P(u)$$

Selectores de *unit*

La constante para la selección en *unit* (destructor):

$$unitrec \in (P \in (unit)Set, P(<>), u \in unit)P(u)$$

La regla de igualdad::

$$unitrec(P, p, <>) = p \in P(<>)$$

En particular, podemos construir a partir de este un caso mas débil, no dependiente:

$$unitelim \in (P \in Set, P, unit)P$$

En particular, podemos construir a partir de este un caso mas débil, no dependiente:

$$unitelim \in (P \in Set, P, unit)P$$

Se construye como:

$$unitelim := [P \in Set]unitrec([\_ \in unit]P)$$

En particular, podemos construir a partir de este un caso mas débil, no dependiente:

$$unitelim \in (P \in Set, P, unit)P$$

Se construye como:

$$unitelim := [P \in Set]unitrec([\_ \in unit]P)$$

El conjunto vacío

Declárense las siguientes constantes:

El conjunto vacío

Declárense las siguientes constantes:
El constructor del conjunto:

El conjunto vacío

Declárense las siguientes constantes:
El constructor del conjunto:

$$empty \in Set$$

El conjunto vacío

Declárense las siguientes constantes:
El constructor del conjunto:

$$\mathit{empty} \in \mathit{Set}$$

No hay constructores

El conjunto vacío

Declárense las siguientes constantes:
El constructor del conjunto:

$$empty \in Set$$

No hay constructores
La constante para la destrucción:

El conjunto vacío

Declárense las siguientes constantes:
El constructor del conjunto:

$$empty \in Set$$

No hay constructores
La constante para la destrucción:

$$emptyrec \in (P \in (empty)Set, e \in empty)P(e)$$

El conjunto vacío

Declárense las siguientes constantes:
El constructor del conjunto:

$$empty \in Set$$

No hay constructores
La constante para la destrucción:

$$emptyrec \in (P \in (empty)Set, e \in empty)P(e)$$

No hay reglas de igualdad:

En particular, podemos construir a partir de este un caso mas débil, no dependiente:

$$emptyelim \in (P \in Set, empty)P$$

En particular, podemos construir a partir de este un caso mas débil, no dependiente:

$$emptyelim \in (P \in Set, empty)P$$

Se construye como:

$$emptyelim := [P \in Set]emptyrec([\_ \in empty]P)$$

En particular, podemos construir a partir de este un caso mas débil, no dependiente:

$$emptyelim \in (P \in Set, empty)P$$

Se construye como:

$$emptyelim := [P \in Set]emptyrec([\_ \in empty]P)$$

Corresponde a al principio *ex falso quodlibet*.

En particular, podemos construir a partir de este un caso mas débil, no dependiente:

$$emptyelim \in (P \in Set, empty)P$$

Se construye como:

$$emptyelim := [P \in Set]emptyrec([\_ \in empty]P)$$

Corresponde a al principio *ex falso quodlibet*.
El conjunto *empty* constituye una representación adecuada de la proposición absurda.

En particular, podemos construir a partir de este un caso mas débil, no dependiente:

$$emptyelim \in (P \in Set, empty)P$$

Se construye como:

$$emptyelim := [P \in Set]emptyrec([\_ \in empty]P)$$

Corresponde a al principio *ex falso quodlibet*.
El conjunto *empty* constituye una representación adecuada de la proposición absurda.
Así como *unit* de la trivial.

El conjunto de los naturales

El conjunto de los naturales

Declárense las siguientes constantes:

El conjunto de los naturales

Declárense las siguientes constantes:
El constructor del conjunto:

$$nat \in Set$$

El conjunto de los naturales

Declárense las siguientes constantes:
El constructor del conjunto:

$$nat \in Set$$

Los constructores:

$$0 \in nat$$

$$S \in (nat)nat$$

El destructor genérico:

$$natrec \in (P \in (nat)Set, P(0), (n \in nat, P(n))P(S(n)), n \in nat)P(n)$$

El destructor genérico:

$$natrec \in (P \in (nat)Set, P(0), (n \in nat, P(n))P(S(n)), n \in nat)P(n)$$

Las reglas de igualdad:

$$natrec(P, p_0, p_s, 0) = p_0 \in P(0)$$

$$natrec(P, p_0, p_s, S(n)) = p_s(n, natrec(P, p_0, p_s, n)) \in P(S(n))$$

El conjunto de los naturales

Eliminación no dependiente:

$$natelim \in (P \in Set, P, (nat, P)P, nat)P$$

$$natelim := [P \in Set]natrec([\_ \in nat]P)$$

Programando con nats

Ejemplos de implementaciones:

Programando con nats

Ejemplos de implementaciones:

- $plus \in (nat, nat)nat$

Programando con nats

Ejemplos de implementaciones:

- $plus \in (nat, nat)nat$

$$plus := [x, y \in nat]natelim(nat, x, [\_, z \in nat]S(z), y)$$

Programando con nats

Ejemplos de implementaciones:

- $plus \in (nat, nat)nat$

$$plus := [x, y \in nat]natelim(nat, x, [\_, z \in nat]S(z), y)$$

- $mult \in (nat, nat)nat$

Programando con nats

Ejemplos de implementaciones:

- $plus \in (nat, nat)nat$

$$plus := [x, y \in nat]natelim(nat, x, [\_, z \in nat]S(z), y)$$

- $mult \in (nat, nat)nat$

$$mult := [x, y \in nat]natelim(nat, 0, [\_, z \in nat]plus(z, x), y)$$

Programando con nats

Ejemplos de implementaciones:

- $plus \in (nat, nat)nat$

$$plus := [x, y \in nat]natelim(nat, x, [\_, z \in nat]S(z), y)$$

- $mult \in (nat, nat)nat$

$$mult := [x, y \in nat]natelim(nat, 0, [\_, z \in nat]plus(z, x), y)$$

- $pred \in (nat)nat$

Programando con nats

Ejemplos de implementaciones:

- $plus \in (nat, nat)nat$

$$plus := [x, y \in nat]natelim(nat, x, [\_, z \in nat]S(z), y)$$

- $mult \in (nat, nat)nat$

$$mult := [x, y \in nat]natelim(nat, 0, [\_, z \in nat]plus(z, x), y)$$

- $pred \in (nat)nat$

$$pred := [x \in nat]natelim(nat, 0, [z, \_ \in nat]z, x)$$

El producto cartesiano prod

Se considera la *familia* de conjuntos definida por:

El producto cartesiano prod

Se considera la *familia* de conjuntos definida por:
El constructor de conjuntos:

$$prod \in (A, B \in Set)Set$$

El producto cartesiano prod

Se considera la *familia* de conjuntos definida por:
El constructor de conjuntos:

$$prod \in (A, B \in Set)Set$$

El constructor:

$$pair \in (A, B \in Set, A, B)prod(A, B)$$

El producto cartesiano prod

El producto cartesiano prod

El destructor:

$$prodrec \in (A, B \in Set, P \in (A \times B)Set,$$

$$(a \in A, b \in B)P(pair(A, B, a, b)), p \in A \times B)P(p)$$

El producto cartesiano prod

El destructor:

$$prodrec \in (A, B \in Set, P \in (A \times B)Set,$$

$$(a \in A, b \in B)P(pair(A, B, a, b)), p \in A \times B)P(p)$$

La regla de igualdad:

$$prodrec(A, B, P, f, pair(A, B, a, b)) = f(a, b) \in P(pair(A, B, a, b))$$

El producto cartesiano prod

El destructor:

$$prodrec \in (A, B \in Set, P \in (A \times B)Set,$$
$$(a \in A, b \in B)P(pair(A, B, a, b)), p \in A \times B)P(p)$$

La regla de igualdad:

$$prodrec(A, B, P, f, pair(A, B, a, b)) = f(a, b) \in P(pair(A, B, a, b))$$

Se pueden implementar *fst*, *snd*

La suma directa sum

Se considera la *familia* de conjuntos definida por:

La suma directa sum

Se considera la *familia* de conjuntos definida por:
El constructor de conjuntos:

$$sum \in (A, B \in Set)Set$$

La suma directa sum

Se considera la *familia* de conjuntos definida por:
El constructor de conjuntos:

$$sum \in (A, B \in Set)Set$$

Los constructores:

$$left \in (A, B \in Set, A)sum(A, B)$$

$$right \in (A, B \in Set, B)sum(A, B)$$

La suma directa sum

La suma directa sum

El destructor:

$$sumrec \in (A, B \in Set, P \in (A + B)Set,$$
$$(a \in A)P(leftA, B, a), (b \in B)P(right(A, B, b)),$$
$$p \in A + B)P(p)$$

La suma directa sum

El destructor:

$$sumrec \in (A, B \in Set, P \in (A + B)Set,$$
$$(a \in A)P(leftA, B, a), (b \in B)P(right(A, B, b)),$$
$$p \in A + B)P(p)$$

Las regla de igualdad:

$$sumrec(A, B, P, f, g, left(A, B, a)) = f(a) \in P(left(A, B, a))$$

$$sumrec(A, B, P, f, g, right(A, B, b)) = g(b) \in P(right(A, B, b))$$

La suma directa sum

El destructor:

$$sumrec \in (A, B \in Set, P \in (A + B)Set,$$
$$(a \in A)P(leftA, B, a), (b \in B)P(right(A, B, b)),$$
$$p \in A + B)P(p)$$

Las regla de igualdad:

$$sumrec(A, B, P, f, g, left(A, B, a)) = f(a) \in P(left(A, B, a))$$

$$sumrec(A, B, P, f, g, right(A, B, b)) = g(b) \in P(right(A, B, b))$$