Formalización de Sistema I con tipo Top

Agustin Francisco Settimo

Directora: Cecilia Manzino
Co-Director: Cristian Sottile

Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Departamento de Ciencias de la Computación

Outline

Definición

- Gramática de términos

Definición

- Gramática de términos

Definición

- Gramática de términos

$$t := x \mid \lambda x.t \mid t\ t$$

Definición

- Gramática de términos

$$t := x \mid \lambda x.t \mid t\ t \mid \langle t, t\rangle \mid \pi_1 t \mid \pi_2 t$$

Definición

- Gramática de términos

$$t := x \mid \lambda x.t \mid t\ t \mid \langle t, t \rangle \mid \pi_1 t \mid \pi_2 t \mid \star$$

- Computación (reglas de reescritura)

Definición

- Gramática de términos

$$t := x \mid \lambda x.t \mid t\ t \mid \langle t, t\rangle \mid \pi_1 t \mid \pi_2 t \mid \star$$

- Computación (reglas de reescritura)
  - Aplicación (β-reducción)

$$(\lambda x.t)s \hookrightarrow_\beta t[s/x]$$

Definición

- Gramática de términos

$$t := x \mid \lambda x.t \mid t\ t \mid \langle t, t \rangle \mid \pi_1 t \mid \pi_2 t \mid \star$$

- Computación (reglas de reescritura)
  - Aplicación (β-reducción)

$$(\lambda x.t)s \hookrightarrow_\beta t[s/x]$$

$t[s/x]$ reemplaza todas las variables x que aparecen en t por s.

Definición

- Gramática de términos

$$t := x \mid \lambda x.t \mid t\ t \mid \langle t, t\rangle \mid \pi_1 t \mid \pi_2 t \mid \star$$

- Computación (reglas de reescritura)
  - Aplicación (β-reducción)

$$(\lambda x.t)s \hookrightarrow_\beta t[s/x]$$

  $t[s/x]$ reemplaza todas las variables x que aparecen en t por s.
  - Proyección

$$\pi_1\langle r, s\rangle \hookrightarrow_\pi r \quad \pi_2\langle r, s\rangle \hookrightarrow_\pi s$$

Ejemplos

Swap

$$\lambda p.\langle \pi_2(p), \pi_1(p) \rangle$$

Ejemplos

Swap

$$\lambda p.\langle \pi_2(p), \pi_1(p) \rangle$$

$$(\lambda p.\langle \pi_2(p), \pi_1(p) \rangle)\ \langle a, b \rangle$$

Ejemplos

Swap

$$\lambda p.\langle \pi_2(p), \pi_1(p)\rangle$$

$$(\lambda p.\langle \pi_2(p), \pi_1(p)\rangle)\ \langle a, b\rangle$$
$$\hookrightarrow_\beta\ \langle \pi_2\langle a, b\rangle, \pi_1\langle a, b\rangle\rangle$$

Ejemplos

Swap

$$\lambda p.\langle \pi_2(p), \pi_1(p)\rangle$$

$$(\lambda p.\langle \pi_2(p), \pi_1(p)\rangle)\ \langle a, b\rangle$$
$$\hookrightarrow_\beta\ \langle \pi_2\langle a, b\rangle, \pi_1\langle a, b\rangle\rangle$$
$$\hookrightarrow_\pi\ \langle b, \pi_1\langle a, b\rangle\rangle$$

Ejemplos

Swap

$$\lambda p.\langle \pi_2(p), \pi_1(p)\rangle$$

$$\begin{aligned}
&(\lambda p.\langle \pi_2(p), \pi_1(p)\rangle)\ \langle a, b\rangle \\
&\hookrightarrow_\beta \langle \pi_2\langle a, b\rangle, \pi_1\langle a, b\rangle\rangle \\
&\hookrightarrow_\pi \langle b, \pi_1\langle a, b\rangle\rangle \\
&\hookrightarrow_\pi \langle b, a\rangle
\end{aligned}$$

Problemas

Algunos términos no reducen (están atascados):

$$\langle a, b \rangle \; c$$

$$\pi_1(\lambda x.x)$$

Problemas

Algunos términos no reducen (están atascados):

$$\langle a, b \rangle \; c$$

$$\pi_1(\lambda x.x)$$

Otros nunca terminan de reducir:

$$(\lambda x.xx)$$

Problemas

Algunos términos no reducen (están atascados):

$$\langle a, b \rangle \; c$$

$$\pi_1(\lambda x.x)$$

Otros nunca terminan de reducir:

$$(\lambda x.xx)(\lambda x.xx)$$

Problemas

Algunos términos no reducen (están atascados):

$$\langle a, b \rangle \; c$$

$$\pi_1(\lambda x.x)$$

Otros nunca terminan de reducir:

$$(\lambda x.xx)(\lambda x.xx) \hookrightarrow (\lambda x.xx)(\lambda x.xx)$$

Problemas

Algunos términos no reducen (están atascados):

$$\langle a, b \rangle\ c$$

$$\pi_1(\lambda x.x)$$

Otros nunca terminan de reducir:

$$(\lambda x.xx)(\lambda x.xx) \hookrightarrow (\lambda x.xx)(\lambda x.xx) \hookrightarrow (\lambda x.xx)(\lambda x.xx) \ldots$$

Sistema de tipos

- Tipos

$$A := A \to A \mid A \times A \mid \top$$

Sistema de tipos

- Tipos
$$A := A \to A \mid A \times A \mid \top$$
- Contextos de tipado
$$\Gamma := \emptyset \mid \Gamma, x : A$$

Reglas de tipado

$$\frac{\Gamma \ni x : {\color{red}A}}{\Gamma \vdash x : {\color{red}A}}\ (ax)$$

Sistema de tipos

Reglas de tipado

$$\frac{\Gamma \ni x : \color{red}{A}}{\Gamma \vdash x : \color{red}{A}}\ (ax) \qquad \frac{}{\Gamma \vdash \star : \top}\ (\top_i)$$

Reglas de tipado

$$\frac{\Gamma \ni x : A}{\Gamma \vdash x : A}\ (ax) \qquad \frac{}{\Gamma \vdash \star : \top}\ (\top_i)$$

$$\frac{\Gamma, x : A \vdash r : B}{\Gamma \vdash \lambda x.r : A \to B}\ (\Rightarrow_i) \qquad \frac{\Gamma \vdash r : A \to B \qquad \Gamma \vdash s : A}{\Gamma \vdash r\ s : B}\ (\Rightarrow_e)$$

Reglas de tipado

$$\frac{\Gamma \ni x : A}{\Gamma \vdash x : A}\ (ax) \qquad \frac{}{\Gamma \vdash \star : \top}\ (\top_i)$$

$$\frac{\Gamma, x : A \vdash r : B}{\Gamma \vdash \lambda x.r : A \to B}\ (\Rightarrow_i) \qquad \frac{\Gamma \vdash r : A \to B \qquad \Gamma \vdash s : A}{\Gamma \vdash r\ s : B}\ (\Rightarrow_e)$$

$$\frac{\Gamma \vdash r : A \qquad \Gamma \vdash s : B}{\Gamma \vdash \langle r, s\rangle : A \times B}\ (\times_i) \quad \frac{\Gamma \vdash r : A \times B}{\Gamma \vdash \pi_1(r) : A}\ (\times_{e1}) \quad \frac{\Gamma \vdash r : A \times B}{\Gamma \vdash \pi_2(r) : B}\ (\times_{e2})$$

Propiedades

Definición (Forma normal)

t es normal si no puede reducirse. $t \not\to$

Propiedades

Definición (Forma normal)

t es normal si no puede reducirse. $t \not\hookrightarrow$

Progreso

Si $\Gamma \vdash t : A$ entonces t es normal o $t \hookrightarrow t'$.

Propiedades

Definición (Forma normal)

t es normal si no puede reducirse. $t \not\hookrightarrow$

Progreso

Si $\Gamma \vdash t : A$ entonces t es normal o $t \hookrightarrow t'$.

Normalización Fuerte

Si $\Gamma \vdash t : A$ entonces no existe ninguna secuencia de reducción infinita.
$t \hookrightarrow t_1 \hookrightarrow t_2 \hookrightarrow t_3 \hookrightarrow \ldots$

Outline

Motivación

Los tipos indican la **forma** y la **clase de problema** de un programa:

Motivación

Los tipos indican la **forma** y la **clase de problema** de un programa:

$$suma : (int \times int) \to int$$

Motivación

Los tipos indican la **forma** y la **clase de problema** de un programa:

$$suma : (int \times int) \rightarrow int$$

Existen programas con tipos distintos para la misma clase de problema:

$$suma' : int \rightarrow int \rightarrow int$$

Motivación

Los tipos indican la **forma** y la **clase de problema** de un programa:

$$suma : (int \times int) \to int$$

Existen programas con tipos distintos para la misma clase de problema:

$$suma' : int \to int \to int$$

Los programas se combinan según su forma:

$$suma\ \langle 2,\ 3\rangle \quad suma'\ 2\ 3$$

Motivación

Los tipos indican la **forma** y la **clase de problema** de un programa:

$$suma : (int \times int) \to int$$

Existen programas con tipos distintos para la misma clase de problema:

$$suma' : int \to int \to int$$

Los programas se combinan según su forma:

$$suma\ \langle 2,\ 3\rangle \quad suma'\ 2\ 3$$

Sería interesante poder combinarlos según su significado:

$$suma\ 2\ 3 \quad suma'\ \langle 2,\ 3\rangle$$

Isomorfismos entre tipos

Dos tipos A y B son *isomorfos* si solo si existen:

$$f : A \to B \quad g : B \to A$$

Isomorfismos entre tipos

Dos tipos A y B son *isomorfos* si solo si existen:

$$f : A \to B \quad g : B \to A$$

Tales que:

$$f \circ g = id_A \quad g \circ f = id_B$$

Isomorfismos entre tipos

Dos tipos A y B son *isomorfos* si solo si existen:

$$f : A \to B \quad g : B \to A$$

Tales que:

$$f \circ g = id_A \quad g \circ f = id_B$$

Y lo notamos $A \equiv B$.

Ejemplo

Isomorfismo de Curry

$$(A \times B) \to C \equiv A \to (B \to C)$$

Ejemplo

Isomorfismo de Curry

$$(A \times B) \to C \equiv A \to (B \to C)$$

$$curry = \lambda f^{(A \times B) \to C}.\lambda a^{A}.\lambda b^{B}.f\langle a, b\rangle$$

Ejemplo

Isomorfismo de Curry

$$(A \times B) \to C \equiv A \to (B \to C)$$

$$curry = \lambda f^{(A \times B) \to C}.\lambda a^{A}.\lambda b^{B}.f\langle a, b\rangle$$

$$uncurry = \lambda f^{A \to B \to C}.\lambda y^{A \times B}.f(\pi_1 y)(\pi_2 y)$$

Ejemplo

Isomorfismo de Curry

$$(A \times B) \to C \equiv A \to (B \to C)$$

$$curry = \lambda f^{(A \times B) \to C}.\lambda a^{A}.\lambda b^{B}.f\langle a, b\rangle$$

$$uncurry = \lambda f^{A \to B \to C}.\lambda y^{A \times B}.f(\pi_1 y)(\pi_2 y)$$

$$curry \circ uncurry = id_{(A \times B) \to C} \quad uncurry \circ curry = id_{A \to B \to C}$$

$$A \times B \equiv B \times A \qquad (\textsc{comm})$$

Conjunto axiomático

$$A \times B \equiv B \times A \tag{COMM}$$

$$A \times (B \times C) \equiv (A \times B) \times C \tag{ASSO}$$

Conjunto axiomático

$$A \times B \equiv B \times A \qquad (\textsc{comm})$$

$$A \times (B \times C) \equiv (A \times B) \times C \qquad (\textsc{asso})$$

$$(A \times B) \to C \equiv A \to (B \to C) \qquad (\textsc{curry})$$

Conjunto axiomático

$$A \times B \equiv B \times A \tag{COMM}$$

$$A \times (B \times C) \equiv (A \times B) \times C \tag{ASSO}$$

$$(A \times B) \to C \equiv A \to (B \to C) \tag{CURRY}$$

$$A \to (B \times C) \equiv (A \to B) \times (A \to C) \tag{DIST}$$

Conjunto axiomático

$$A \times B \equiv B \times A \tag{COMM}$$

$$A \times (B \times C) \equiv (A \times B) \times C \tag{ASSO}$$

$$(A \times B) \to C \equiv A \to (B \to C) \tag{CURRY}$$

$$A \to (B \times C) \equiv (A \to B) \times (A \to C) \tag{DIST}$$

$$A \times \top \equiv A \tag{$\mathrm{ID}_\times$}$$

Conjunto axiomático

$$A \times B \equiv B \times A \tag{COMM}$$

$$A \times (B \times C) \equiv (A \times B) \times C \tag{ASSO}$$

$$(A \times B) \to C \equiv A \to (B \to C) \tag{CURRY}$$

$$A \to (B \times C) \equiv (A \to B) \times (A \to C) \tag{DIST}$$

$$A \times \top \equiv A \tag{$\mathrm{ID}_\times$}$$

$$A \to \top \equiv \top \tag{ABS}$$

$$A \times B \equiv B \times A \quad (\textsc{comm})$$
$$A \times (B \times C) \equiv (A \times B) \times C \quad (\textsc{asso})$$
$$(A \times B) \to C \equiv A \to (B \to C) \quad (\textsc{curry})$$
$$A \to (B \times C) \equiv (A \to B) \times (A \to C) \quad (\textsc{dist})$$
$$A \times \top \equiv A \quad (\textsc{id}_\times)$$
$$A \to \top \equiv \top \quad (\textsc{ABS})$$
$$\top \to A \equiv A \quad (\textsc{id}_\Rightarrow)$$

Se añade la regla $\equiv$ (equiv)

$$\frac{A \equiv B \qquad \Gamma \vdash r : A}{\Gamma \vdash r : B}\ (\equiv)$$

Internalización de isomorfismos

Se añade la regla $\equiv$ (equiv)

$$\frac{A \equiv B \qquad \Gamma \vdash r : A}{\Gamma \vdash r : B}\ (\equiv)$$

Esto permite usar $r : A$ en lugar de $r : B$, siempre y cuando $A \equiv B$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash r : A}{\Gamma \vdash \lambda x.r : C \to A}\ (\Rightarrow_i) \qquad \dfrac{\Gamma \vdash s : B}{\Gamma \vdash \lambda x.s : C \to B}\ (\Rightarrow_i)}{\Gamma \vdash \langle \lambda x.r, \lambda x.s\rangle : (C \to A) \times (C \to B)}\ (\times_i)}{\Gamma \vdash \langle \lambda x.r, \lambda x.s\rangle : C \to (A \times B)}\ (\equiv) \qquad \Gamma \vdash t : C}{\Gamma \vdash \langle \lambda x.r, \lambda x.s\rangle\, t : A \times B}\ (\Rightarrow_e)$$

Relación de equivalencia entre términos

$$\langle r, s \rangle \rightleftarrows \langle s, r \rangle \qquad (\textsc{comm})$$

Relación de equivalencia entre términos

$$\langle r, s \rangle \rightleftarrows \langle s, r \rangle \tag{COMM}$$

$$\langle r, \langle s, t \rangle \rangle \rightleftarrows \langle \langle r, s \rangle, t \rangle \tag{ASSO}$$

Relación de equivalencia entre términos

$$\langle r, s \rangle \rightleftarrows \langle s, r \rangle \qquad (\textsc{comm})$$

$$\langle r, \langle s, t \rangle \rangle \rightleftarrows \langle \langle r, s \rangle, t \rangle \qquad (\textsc{asso})$$

$$\lambda x^A.\langle r, s \rangle \rightleftarrows \langle \lambda x^A.r, \lambda x^A.s \rangle \qquad (\textsc{dist}_\lambda)$$

Relación de equivalencia entre términos

$$\langle r, s\rangle \rightleftarrows \langle s, r\rangle \tag{COMM}$$

$$\langle r, \langle s, t\rangle\rangle \rightleftarrows \langle\langle r, s\rangle, t\rangle \tag{ASSO}$$

$$\lambda x^A.\langle r, s\rangle \rightleftarrows \langle \lambda x^A.r, \lambda x^A.s\rangle \tag{DIST$_\lambda$}$$

$$\langle r, s\rangle\, t \rightleftarrows \langle r\, t, s\, t\rangle \tag{DIST$_{app}$}$$

Relación de equivalencia entre términos

$$\langle r, s \rangle \rightleftarrows \langle s, r \rangle \qquad (\textsc{comm})$$

$$\langle r, \langle s, t \rangle \rangle \rightleftarrows \langle \langle r, s \rangle, t \rangle \qquad (\textsc{asso})$$

$$\lambda x^A.\langle r, s \rangle \rightleftarrows \langle \lambda x^A.r, \lambda x^A.s \rangle \qquad (\textsc{dist}_\lambda)$$

$$\langle r, s \rangle\, t \rightleftarrows \langle r\, t, s\, t \rangle \qquad (\textsc{dist}_{app})$$

$$r\, \langle s, t \rangle \rightleftarrows r\, s\, t \qquad (\textsc{curry})$$

Relación de equivalencia entre términos

$$\langle r, s \rangle \rightleftarrows \langle s, r \rangle \tag{COMM}$$

$$\langle r, \langle s, t \rangle \rangle \rightleftarrows \langle \langle r, s \rangle, t \rangle \tag{ASSO}$$

$$\lambda x^A . \langle r, s \rangle \rightleftarrows \langle \lambda x^A . r, \lambda x^A . s \rangle \tag{DIST$_\lambda$}$$

$$\langle r, s \rangle\, t \rightleftarrows \langle r\, t, s\, t \rangle \tag{DIST$_{app}$}$$

$$r\, \langle s, t \rangle \rightleftarrows r\, s\, t \tag{CURRY}$$

$$\langle \lambda x . r, \lambda x . s \rangle\ t$$

Relación de equivalencia entre términos

$$\langle r, s\rangle \rightleftarrows \langle s, r\rangle \qquad (\text{COMM})$$

$$\langle r, \langle s, t\rangle\rangle \rightleftarrows \langle\langle r, s\rangle, t\rangle \qquad (\text{ASSO})$$

$$\lambda x^A.\langle r, s\rangle \rightleftarrows \langle \lambda x^A.r, \lambda x^A.s\rangle \qquad (\text{DIST}_\lambda)$$

$$\langle r, s\rangle\, t \rightleftarrows \langle r\,t, s\,t\rangle \qquad (\text{DIST}_{app})$$

$$r\,\langle s, t\rangle \rightleftarrows r\,s\,t \qquad (\text{CURRY})$$

$$\langle \lambda x.r, \lambda x.s\rangle\; t$$

$$\rightleftarrows_{\text{DIST}_\lambda}$$

$$(\lambda x.\langle r, s\rangle)\; t$$

Relación de equivalencia entre términos

$$\langle r, s\rangle \rightleftarrows \langle s, r\rangle \qquad (\textsc{comm})$$

$$\langle r, \langle s, t\rangle\rangle \rightleftarrows \langle\langle r, s\rangle, t\rangle \qquad (\textsc{asso})$$

$$\lambda x^A.\langle r, s\rangle \rightleftarrows \langle \lambda x^A.r, \lambda x^A.s\rangle \qquad (\textsc{dist}_\lambda)$$

$$\langle r, s\rangle\, t \rightleftarrows \langle r\,t, s\,t\rangle \qquad (\textsc{dist}_{app})$$

$$r\,\langle s, t\rangle \rightleftarrows r\,s\,t \qquad (\textsc{curry})$$

$$\begin{aligned}
&\langle \lambda x.r, \lambda x.s\rangle\ t \\
\rightleftarrows_{\textsc{dist}_\lambda}\ & \\
&(\lambda x.\langle r, s\rangle)\ t \\
\hookrightarrow_\beta\ & \\
&\langle r[t/x], s[t/x]\rangle
\end{aligned}$$

Problema

COMM permite cambiar el elemento de una proyección:

Problema

$\textsc{comm}$ permite cambiar el elemento de una proyección:

$$\pi_1 \langle r, s \rangle \hookrightarrow_{\pi_1} r$$

$$\pi_1 \langle r, s \rangle \rightleftarrows_{\textsc{comm}} \pi_1 \langle s, r \rangle \hookrightarrow_{\pi_1} s$$

Problema

$\textsc{comm}$ permite cambiar el elemento de una proyección:

$$\pi_1\langle r, s\rangle \hookrightarrow_{\pi_1} r$$
$$\pi_1\langle r, s\rangle \rightleftarrows_{\textsc{comm}} \pi_1\langle s, r\rangle \hookrightarrow_{\pi_1} s$$

Si $r : A$, $s : B$ y $A \not\equiv B$ esto puede ser un problema.

Problema

$\textsc{comm}$ permite cambiar el elemento de una proyección:

$$\pi_1\langle r, s\rangle \hookrightarrow_{\pi_1} r$$
$$\pi_1\langle r, s\rangle \rightleftarrows_{\textsc{comm}} \pi_1\langle s, r\rangle \hookrightarrow_{\pi_1} s$$

Si $r : A$, $s : B$ y $A \not\equiv B$ esto puede ser un problema.
La solución es acceder al elemento usando los tipos:

$$\text{si } r : A \quad \pi_A\langle r, s\rangle \hookrightarrow_{\pi} r$$

Relación de reducción

Se define la relación de reducción módulo isomorfismos:

$$\rightsquigarrow \; := \; \rightleftarrows^{*} \circ \hookrightarrow \circ \rightleftarrows^{*}$$

Relación de reducción

Se define la relación de reducción módulo isomorfismos:

$$\rightsquigarrow := \rightleftarrows^* \circ \hookrightarrow \circ \rightleftarrows^*$$

Se aplican todas las transformaciones necesarias para poder reducir el término correctamente.

Outline

Indices de De Bruijn

$$\lambda z.(\lambda y.y(\lambda x.x))(\lambda x.zx)$$

Indices de De Bruijn

$$\lambda z.(\lambda y.y(\lambda x.x))(\lambda x.zx)$$

$$\lambda(\lambda\ 0\ (\lambda\ 0))(\lambda\ 1\ 0)$$

Indices de De Bruijn

$$\lambda z.(\lambda y.y(\lambda x.x))(\lambda x.zx)$$

$$\lambda(\lambda\ 0\ (\lambda\ 0))(\lambda\ 1\ 0)$$

$$(\lambda x.\lambda y.zx(\lambda z.zx))\ (\lambda x.wx)$$

Indices de De Bruijn

$$\lambda z.(\lambda y.y(\lambda x.x))(\lambda x.zx)$$

$$\lambda(\lambda\ 0\ (\lambda\ 0))(\lambda\ 1\ 0)$$

$$(\lambda x.\lambda y.zx(\lambda z.zx))\ (\lambda x.wx)$$

$$(\lambda\ \lambda\ 2\ 1\ (\lambda\ 0\ 2))\ (\lambda\ 2\ 0)$$

Indices de De Bruijn

$$\lambda z.(\lambda y.y(\lambda x.x))(\lambda x.zx)$$

$$\lambda(\lambda\ 0\ (\lambda\ 0))(\lambda\ 1\ 0)$$

$$(\lambda x.\lambda y.zx(\lambda z.zx))\ (\lambda x.wx)$$

$$(\lambda\ \lambda\ 2\ 1\ (\lambda\ 0\ 2))\ (\lambda\ 2\ 0) \hookrightarrow_\beta \lambda\ 1\ \square\ (\lambda\ 0\ \square)$$

Indices de De Bruijn

$$\lambda z.(\lambda y.y(\lambda x.x))(\lambda x.zx)$$

$$\lambda(\lambda\ 0\ (\lambda\ 0))(\lambda\ 1\ 0)$$

$$(\lambda x.\lambda y.zx(\lambda z.zx))\ (\lambda x.wx)$$

$$(\lambda\ \lambda\ 2\ 1\ (\lambda\ 0\ 2))\ (\lambda\ 2\ 0) \hookrightarrow_\beta \lambda\ 1\ (\lambda\ 3\ 0)\ (\lambda\ 0\ (\lambda\ 4\ 0))$$

Usando índices, las substituciones son secuencias de términos:

$$\{0 \mapsto a, 1 \mapsto b, 2 \mapsto 0, 3 \mapsto 1, \dots\} = \{a, b, 0, 1, \dots\}$$

Substituciones explícitas

Usando índices, las substituciones son secuencias de términos:

$$\{0 \mapsto a, 1 \mapsto b, 2 \mapsto 0, 3 \mapsto 1, \dots\} = \{a, b, 0, 1, \dots\}$$

- id (substitución identidad) $\{i \mapsto i\} = \{0, 1, 2, \dots\}$.

Substituciones explícitas

Usando índices, las substituciones son secuencias de términos:

$$\{0 \mapsto a, 1 \mapsto b, 2 \mapsto 0, 3 \mapsto 1, \ldots\} = \{a, b, 0, 1, \ldots\}$$

- id (substitución identidad) $\{i \mapsto i\} = \{0, 1, 2, \ldots\}$.
- $\uparrow$ (shift) $\{i \mapsto i + 1\} = \{1, 2, 3, \ldots\}$.

Usando índices, las substituciones son secuencias de términos:

$$\{0 \mapsto a, 1 \mapsto b, 2 \mapsto 0, 3 \mapsto 1, \dots\} = \{a, b, 0, 1, \dots\}$$

- id (substitución identidad) $\{i \mapsto i\} = \{0, 1, 2, \dots\}$.
- $\uparrow$ (shift) $\{i \mapsto i + 1\} = \{1, 2, 3, \dots\}$.
- $a \bullet s$ (cons) $\{0 \mapsto a, i + 1 \mapsto s(i)\}$.

Substituciones explícitas

Usando índices, las substituciones son secuencias de términos:

$$\{0 \mapsto a, 1 \mapsto b, 2 \mapsto 0, 3 \mapsto 1, \ldots\} = \{a, b, 0, 1, \ldots\}$$

- id (substitución identidad) $\{i \mapsto i\} = \{0, 1, 2, \ldots\}$.
- $\uparrow$ (shift) $\{i \mapsto i + 1\} = \{1, 2, 3, \ldots\}$.
- $a \bullet s$ (cons) $\{0 \mapsto a, i + 1 \mapsto s(i)\}$.

Substituciones explícitas

Usando índices, las substituciones son secuencias de términos:

$$\{0 \mapsto a, 1 \mapsto b, 2 \mapsto 0, 3 \mapsto 1, \dots\} = \{a, b, 0, 1, \dots\}$$

- id (substitución identidad) $\{i \mapsto i\} = \{0, 1, 2, \dots\}$.
- $\uparrow$ (shift) $\{i \mapsto i + 1\} = \{1, 2, 3, \dots\}$.
- $a \bullet s$ (cons) $\{0 \mapsto a, i + 1 \mapsto s(i)\}$.
  Por ejemplo $a \bullet id = \{0 \mapsto a, i + 1 \mapsto i\} = \{a, 0, 1, 2, \dots\}$

Substituciones explícitas

Usamos $\langle\!\langle s \rangle\!\rangle t$ para denotar la aplicación de la subst s sobre t.

Substituciones explícitas

Usamos $\langle\!\langle s \rangle\!\rangle t$ para denotar la aplicación de la subst s sobre t.

$$\langle\!\langle s \rangle\!\rangle x = s(x)$$

Substituciones explícitas

Usamos $\langle\!\langle s \rangle\!\rangle t$ para denotar la aplicación de la subst s sobre t.

$$\langle\!\langle s \rangle\!\rangle x = s(x)$$
$$\langle\!\langle s \rangle\!\rangle (a\ b) = \langle\!\langle s \rangle\!\rangle a\ \langle\!\langle s \rangle\!\rangle b$$

Substituciones explícitas

Usamos $\langle\!\langle s \rangle\!\rangle t$ para denotar la aplicación de la subst s sobre t.

$$\begin{aligned}
\langle\!\langle s \rangle\!\rangle x &= s(x) \\
\langle\!\langle s \rangle\!\rangle (a\ b) &= \langle\!\langle s \rangle\!\rangle a\ \langle\!\langle s \rangle\!\rangle b \\
\langle\!\langle s \rangle\!\rangle (\lambda a) &= \lambda \langle\!\langle 0 \bullet (\uparrow \circ s) \rangle\!\rangle a
\end{aligned}$$

Substituciones explícitas

Usamos $\langle\langle s\rangle\rangle t$ para denotar la aplicación de la subst s sobre t.

$$\begin{aligned}\langle\langle s\rangle\rangle x &= s(x)\\ \langle\langle s\rangle\rangle(a\ b) &= \langle\langle s\rangle\rangle a\ \langle\langle s\rangle\rangle b\\ \langle\langle s\rangle\rangle(\lambda a) &= \lambda\langle\langle 0 \bullet (\uparrow \circ s)\rangle\rangle a\end{aligned}$$

Implementación de la β-reducción usando esta notación:

$$(\lambda t)\ a \hookrightarrow_\beta t[a] = \langle\langle a \bullet id\rangle\rangle t$$

Tipos

$$A := \top \mid A \to A \mid A \times A$$

```
data Type : Set where
  ⊤   : Type
  _⇒_ : Type → Type → Type
  _×_ : Type → Type → Type
```

Indices intrínsecamente tipados

$$\Gamma := \emptyset \mid \Gamma, x : A$$

```
data Context : Set where
  ∅ : Context
  _,_ : Context → Type → Context
```

$$\Gamma := \emptyset \mid \Gamma, x : A$$

```
data Context : Set where
  ∅ : Context
  _,_ : Context → Type → Context



data _∋_ : Context → Type → Set where
  Z : ∀ {Γ A}
      ----------
    → Γ , A ∋ A

  S_ : ∀ {Γ A B}
    → Γ ∋ B
      ----------
    → Γ , A ∋ B
```

Ejemplo

$\_ : \emptyset \; , \top \Rightarrow \top \; , \top \ni \top$
$\_ = \mathrm{Z}$

$\_ : \emptyset \; , \top \Rightarrow \top \; , \top \ni \top \Rightarrow \top$
$\_ = \mathrm{S} \; \mathrm{Z}$

```
data _⊢_ : Context → Type → Set where

  `_ : ∀ {Γ A}          -- (ax)
    → Γ ∋ A
      -----
    → Γ ⊢ A

  ★ : ∀ {Γ} → Γ ⊢ ⊤ -- (⊤ᵢ)

  ⋮
```

$$\frac{}{\Gamma, x : A \vdash x : A}\ (ax)$$

$$\frac{}{\Gamma \vdash \star : \top}\ (\top_i)$$

```
⋮
λ_  : ∀ {Γ A B} -- (⇒ᵢ)
  → Γ , A ⊢ B
    ---------
  → Γ ⊢ A ⇒ B

_·_ : ∀ {Γ A B} -- (⇒ₑ)
  → Γ ⊢ A ⇒ B
  → Γ ⊢ A
    ---------
  → Γ ⊢ B
⋮
```

$$\frac{\Gamma, x : A \vdash r : B}{\Gamma \vdash \lambda x.r : A \to B}\ (\Rightarrow_i)$$

$$\frac{\Gamma \vdash r : A \to B \qquad \Gamma \vdash s : A}{\Gamma \vdash rs : B}\ (\Rightarrow_e)$$

Términos III

```
⋮
⟨_,_⟩ : ∀ {Γ A B} -- (×ᵢ)
  → Γ ⊢ A
  → Γ ⊢ B
    -----------
  → Γ ⊢ A × B

π : ∀ {Γ A B}      -- (×ₑ)
  → (C : Type)
  → {proof : (C ≅ A) ⊎ (C ≅ B)}
  → Γ ⊢ A × B
    -----------
  → Γ ⊢ C
⋮
```

$$\frac{\Gamma \vdash r : A \qquad \Gamma \vdash s : B}{\Gamma \vdash \langle r, s \rangle : A \times B}\ (\times_i)$$

$$\frac{\Gamma \vdash r : A \times B}{\Gamma \vdash \pi_A(r) : A}\ (\times_e)$$

Términos IV

```
⋮
[_]≡_ : ∀ {Γ A B} -- (≡)
  → A ≡ B
  → Γ ⊢ A
    ---------
  → Γ ⊢ B
```

$$\frac{A \equiv B \qquad \Gamma \vdash r : A}{\Gamma \vdash r : B}\ (\equiv)$$

Ejemplo

```
_ : ∅ , ⊤ ⇒ ⊤ ⊢ ⊤ ⇒ ⊤
_ = ƛ (‘ S Z · (‘ S Z · ‘ Z))
```

$$\lambda x^{\top}.y\ (y\ x) : \top \to \top$$

Ejemplo

```
_ : ∅ , ⊤ ⇒ ⊤ ⊢ ⊤ ⇒ ⊤
_ = ƛ (` S Z · (` S Z · ` Z))
```

$$\lambda x^{\top}.y\ (y\ x) : \top \to \top$$

```
_ : ∅ ⊢ (⊤ ⇒ ⊤) ⇒ ⊤ ⇒ ⊤
_ = ƛ ƛ (` S Z · (` S Z · ` Z))
```

$$\lambda y^{\top\to\top}.\lambda x^{\top}.y\ (y\ x) : (\top \to \top) \to \top \to \top$$

Ejemplo

```
_ : ∅ , ⊤ ⇒ ⊤ ⊢ ⊤ ⇒ ⊤
_ = ƛ (‘ S Z · (‘ S Z · ‘ Z))
```

$$\lambda x^{\top}.y\ (y\ x) : \top \to \top$$

```
_ : ∅ ⊢ (⊤ ⇒ ⊤) ⇒ ⊤ ⇒ ⊤
_ = ƛ ƛ (‘ S Z · (‘ S Z · ‘ Z))
```

$$\lambda y^{\top \to \top}.\lambda x^{\top}.y\ (y\ x) : (\top \to \top) \to \top \to \top$$

```
swap× : ∅ ⊢ A × B ⇒ B × A
swap× = ƛ ⟨ π B (‘ Z) , π A (‘ Z) ⟩
```

Substituciones explícitas

```
Subst : Context → Context → Set
Subst Γ Δ = ∀{A} → Γ ∋ A → Δ ⊢ A
```

Substituciones explícitas

```
Subst : Context → Context → Set
Subst Γ Δ = ∀{A} → Γ ∋ A → Δ ⊢ A
```

```
ids : ∀{Γ} → Subst Γ Γ
ids x = ` x
```

Substituciones explícitas

```
Subst : Context → Context → Set
Subst Γ Δ = ∀{A} → Γ ∋ A → Δ ⊢ A
```

```
ids : ∀{Γ} → Subst Γ Γ
ids x = ` x
```

```
_•_ : ∀{Γ Δ A} → (Δ ⊢ A) → Subst Γ Δ → Subst (Γ , A) Δ
(M • σ) Z     = M
(M • σ) (S x) = σ x
```

Substituciones explícitas

```
_[_] : ∀ {Γ A B}
  → Γ , B ⊢ A
  → Γ ⊢ B
    ---------
  → Γ ⊢ A
N [ M ] = ⟪ M • ids ⟫ N
```

Substituciones explícitas

```
Rename : Context → Context → Set
Rename Γ Δ = ∀{A} → Γ ∋ A → Δ ∋ A

rename : ∀ {Γ Δ} → Rename Γ Δ → Γ ⊢ A → Δ ⊢ A
rename ρ N = ...
```

Substituciones explícitas

```
↑ : ∀ {Γ A B} → Γ ⊢ A → Γ , B ⊢ A
↑ = λ N → rename S_ N
```

Substituciones explícitas

```
↑ : ∀ {Γ A B} → Γ ⊢ A → Γ , B ⊢ A
↑ = λ N → rename S_ N


exts : ∀ {Γ Δ A}
  → Subst Γ Δ
    -----------------------
  → Subst (Γ , A) (Δ , A)
exts σ = ‘ Z • ↑ ∘ σ
```

Substituciones explícitas

```
↑ : ∀ {Γ A B} → Γ ⊢ A → Γ , B ⊢ A
↑ = λ N → rename S_ N


exts : ∀ {Γ Δ A}
  → Subst Γ Δ
    ------------------------
  → Subst (Γ , A) (Δ , A)
exts σ = ` Z • ↑ ∘ σ


⟪_⟫_ : ∀{Γ Δ A} → Subst Γ Δ → Γ ⊢ A → Δ ⊢ A
⟪ σ ⟫ (` k)          = σ k
⟪ σ ⟫ (ƛ N)          = ƛ (⟪ exts σ ⟫ N)
⟪ σ ⟫ (L · M)        = (⟪ σ ⟫ L) · (⟪ σ ⟫ M)
⟪ σ ⟫ 〈 M , N 〉      = 〈 ⟪ σ ⟫ M , ⟪ σ ⟫ N 〉
⟪ σ ⟫ (π C {p} L)    = π C {p} (⟪ σ ⟫ L)
⟪ σ ⟫ ⋆              = ⋆
```

Substituciones explícitas

```
↑ : ∀ {Γ A B} → Γ ⊢ A → Γ , B ⊢ A
↑ = λ N → rename S_ N


exts : ∀ {Γ Δ A}
  → Subst Γ Δ
    ------------------------
  → Subst (Γ , A) (Δ , A)
exts σ = ` Z ● ↑ ∘ σ


⟪_⟫_ : ∀{Γ Δ A} → Subst Γ Δ → Γ ⊢ A → Δ ⊢ A
⟪ σ ⟫ (` k)         = σ k
⟪ σ ⟫ (ƛ N)         = ƛ (⟪ exts σ ⟫ N)
⟪ σ ⟫ (L · M)       = (⟪ σ ⟫ L) · (⟪ σ ⟫ M)
⟪ σ ⟫ ⟨ M , N ⟩     = ⟨ ⟪ σ ⟫ M , ⟪ σ ⟫ N ⟩
⟪ σ ⟫ (π C {p} L) = π C {p} (⟪ σ ⟫ L)
⟪ σ ⟫ ⋆             = ⋆
```

Reducción I

```
data _↪_ : (Γ ⊢ A) → (Γ ⊢ A) → Set where
  ξ-·₁ : ∀ {t t' : Γ ⊢ A ⇒ B} {s : Γ ⊢ A}
    → t ↪ t'
      ---------------
    → t · s ↪ t' · s

  ξ-·₂ : ∀ {t : Γ ⊢ A ⇒ B} {s s' : Γ ⊢ A}
    → s ↪ s'
      ---------------
    → t · s ↪ t · s'

  ζ : ∀ {t t' : Γ , B ⊢ A}
    → t ↪ t'
      --------------------
    → ƛ t ↪ ƛ t'
```

$\vdots$

$$\frac{t \hookrightarrow t'}{t\ s \hookrightarrow t'\ s}$$

$$\frac{s \hookrightarrow s'}{t\ s \hookrightarrow t\ s'}$$

$$\frac{t \hookrightarrow t'}{\lambda t \hookrightarrow \lambda t'}$$

Reducción II

$\vdots$

$\beta\text{-}\lambda : \forall \{t : \Gamma , A \vdash B\} \{s : \Gamma \vdash A\}$

$\text{-----------------------}$

$\to (\lambda\ t) \cdot s \hookrightarrow t\ [\ s\ ]$

$\vdots$

$$(\lambda t)\ s \hookrightarrow t[s]$$

Reducción III

```
⋮
ξ-⟨,⟩₁ : ∀ {r r' : Γ ⊢ A} {s : Γ ⊢ B}
  → r ↪ r'
    -------------------------
  → ⟨ r , s ⟩ ↪ ⟨ r' , s ⟩

ξ-⟨,⟩₂ : ∀ {r : Γ ⊢ A} {s s' : Γ ⊢ B}
  → s ↪ s'
    -------------------------
  → ⟨ r , s ⟩ ↪ ⟨ r , s' ⟩

ξ-π : ∀ {t t' : Γ ⊢ A × B}
  → t ↪ t'
    ---------------------
  → π C {p} t ↪ π C {p} t'
⋮
```

$$\frac{r \hookrightarrow r'}{\langle r, s\rangle \hookrightarrow \langle r', s\rangle}$$

$$\frac{s \hookrightarrow s'}{\langle r, s\rangle \hookrightarrow \langle r, s'\rangle}$$

$$\frac{t \hookrightarrow t'}{\pi_C t \hookrightarrow \pi_C t'}$$

Reducción IV

$\vdots$

$\beta\text{-}\pi_1 : \forall\ \{r : \Gamma \vdash A\}\ \{s : \Gamma \vdash B\}$

$\rightarrow \pi\ A\ \{\mathtt{inj}_1\ \mathtt{refl}\}\ \langle\ r\ ,\ s\ \rangle \hookrightarrow r$

$\beta\text{-}\pi_2 : \forall\ \{r : \Gamma \vdash A\}\ \{s : \Gamma \vdash B\}$

$\rightarrow \pi\ B\ \{\mathtt{inj}_2\ \mathtt{refl}\}\ \langle\ r\ ,\ s\ \rangle \hookrightarrow s$

$\vdots$

Si $r : A \quad \pi_A\langle r, s\rangle \hookrightarrow r$

Reducción V

```
⋮
ξ-≡ : ∀ {t t' : Γ ⊢ A} {iso : A ≡ B}
  → t ↪ t'
    ----------------------
  → [ iso ]≡ t ↪ [ iso ]≡ t'
```

$$\frac{t \hookrightarrow t'}{[iso]_{\equiv}\, t \hookrightarrow [iso]_{\equiv}\, t'}$$

Ejemplo

$$(\pi_{A \to A} \langle \lambda x.x, \star \rangle)\ y \hookrightarrow_{\pi} (\lambda x.x)\ y$$

Ejemplo

$$(\pi_{A\to A}\langle\lambda x.x,\star\rangle)\ y \hookrightarrow_{\pi} (\lambda x.x)\ y$$

```
T₃ : ∅ , ⊤ ⊢ ⊤
T₃ = (π (⊤ ⇒ ⊤) ⟨ ƛ ‘ Z , ⋆ ⟩) · ‘ Z

_ : T₃ ↪ (ƛ ‘ Z) · ‘ Z
_ = ξ-·₁ β-π₁
```

Isomorfismos de tipos

```
data _≡_ : Type → Type → Set where
  comm  : ∀ {A B} → A × B ≡ B × A
  asso  : ∀ {A B C} → A × (B × C) ≡ (A × B) × C
  dist  : ∀ {A B C} → (A ⇒ B) × (A ⇒ C) ≡ A ⇒ B × C
  curry : ∀ {A B C} → A ⇒ B ⇒ C ≡ (A × B) ⇒ C
  id-×  : ∀ {A} → A × ⊤ ≡ A
  id-⇒ : ∀ {A} → ⊤ ⇒ A ≡ A
  abs   : ∀ {A} → A ⇒ ⊤ ≡ ⊤

  sym    : ∀ {A B} → A ≡ B → B ≡ A
  cong⇒₁ : ∀ {A B C} → A ≡ B → A ⇒ C ≡ B ⇒ C
  cong⇒₂ : ∀ {A B C} → A ≡ B → C ⇒ A ≡ C ⇒ B
  cong×₁  : ∀ {A B C} → A ≡ B → A × C ≡ B × C
  cong×₂  : ∀ {A B C} → A ≡ B → C × A ≡ C × B
```

Ejemplo

```
_ : ∀ {A B} → A × B ⇒ ⊤ ≡ B × A ⇒ ⊤
_ = cong⇒₁ comm

T₄ : ∀ {A B} → ∅ ⊢ (A ⇒ B ⇒ A) × (A ⇒ B ⇒ B)
T₄ = [ sym dist ]≡ (ƛ [ sym dist ]≡ (ƛ ⟨ ‘ (S Z) , ‘ Z ⟩))
```

Equivalencias entre términos

$$\langle r, s\rangle \rightleftarrows \langle s, r\rangle$$

```
data _⇄_ : (Γ ⊢ A) → (Γ ⊢ A) → Set where
  comm : ∀ {r : Γ ⊢ A} → {s : Γ ⊢ B}
    → [ comm ]≡ ⟨ r , s ⟩ ⇄ ⟨ s , r ⟩
  sym-comm : ∀ {r : Γ ⊢ A} → {s : Γ ⊢ B}
    → [ sym comm ]≡ ⟨ r , s ⟩ ⇄ ⟨ s , r ⟩
  ⋮
```

Equivalencias entre términos

$$\langle r, \langle s, t\rangle\rangle \rightleftarrows \langle\langle r, s\rangle, t\rangle$$

```
⋮
asso : ∀ {r : Γ ⊢ A} → {s : Γ ⊢ B} → {t : Γ ⊢ C}
  → [ asso ]≡ ⟨ r , ⟨ s , t ⟩ ⟩ ⇄ ⟨ ⟨ r , s ⟩ , t ⟩
sym-asso : ∀ {r : Γ ⊢ A} → {s : Γ ⊢ B} → {t : Γ ⊢ C}
  → [ sym asso ]≡ ⟨ ⟨ r , s ⟩ , t ⟩ ⇄ ⟨ r , ⟨ s , t ⟩ ⟩
⋮
```

Equivalencias entre términos

$$\langle r, s \rangle \rightleftarrows_{\text{SPLIT}}$$

Equivalencias entre términos

$$\langle r, s \rangle \rightleftarrows_{\text{SPLIT}} \langle r, \langle \pi_B(s), \pi_C(s) \rangle \rangle$$

Equivalencias entre términos

$$\langle r, s \rangle \rightleftarrows_{\text{SPLIT}} \langle r, \langle \pi_B(s), \pi_C(s) \rangle \rangle \rightleftarrows_{\text{ASSO}} \langle \langle r, \pi_B(s) \rangle, \pi_C(s) \rangle$$

Equivalencias entre términos

$$\langle r, s\rangle \rightleftarrows_{\text{SPLIT}} \langle r, \langle \pi_B(s), \pi_C(s)\rangle\rangle \rightleftarrows_{\text{ASSO}} \langle\langle r, \pi_B(s)\rangle, \pi_C(s)\rangle$$

```
⋮
asso-split : ∀ {r : Γ ⊢ A} → {s : Γ ⊢ B × C}
  → [ asso ]≡ ⟨ r , s ⟩ ⇄ ⟨ ⟨ r , π B s ⟩ , π C s ⟩
sym-asso-split : ∀ {r : Γ ⊢ A × B} → {s : Γ ⊢ C}
  → [ sym asso ]≡ ⟨ r , s ⟩ ⇄ ⟨ π A r , ⟨ π B r , s ⟩ ⟩
⋮
```

Equivalencias entre términos

$$\lambda\langle r, s\rangle \rightleftarrows \langle \lambda r, \lambda s\rangle$$

⋮

```
dist-λ : ∀ {r : Γ , C ⊢ A} → {s : Γ , C ⊢ B}
  → [ dist ]≡ ⟨ λ r , λ s ⟩ ⇄ λ ⟨ r , s ⟩
```

```
sym-dist-λ : ∀ {r : Γ , C ⊢ A} → {s : Γ , C ⊢ B}
  → [ sym dist ]≡ (λ ⟨ r , s ⟩) ⇄ ⟨ λ r , λ s ⟩
```

⋮

Equivalencias entre términos

$$\lambda x^{A}.\lambda y^{B}.r \rightleftarrows \lambda z^{A\times B}.r[\pi_B(z)/y, \pi_A(z)/x]$$

Equivalencias entre términos

$$\lambda x^{A}.\lambda y^{B}.r \rightleftarrows \lambda z^{A\times B}.r[\pi_B(z)/y, \pi_A(z)/x]$$

$$\lambda r[\pi_B(0), \pi_A(0)]$$

Equivalencias entre términos

$$\lambda x^{A}.\lambda y^{B}.r \rightleftarrows \lambda z^{A\times B}.r[\pi_B(z)/y, \pi_A(z)/x]$$

$$\lambda r[\pi_B(0), \pi_A(0)]$$

```
σ-curry : ∀ {Γ A B} → Subst (Γ , A , B) (Γ , A × B)
σ-curry = π B (‘ Z) • π A (‘ Z) • ids ∘ S_
```

Equivalencias entre términos

$$\lambda x^A.\lambda y^B.r \rightleftarrows \lambda z^{A\times B}.r[\pi_B(z)/y, \pi_A(z)/x]$$

$$\lambda r[\pi_B(0), \pi_A(0)]$$

```
σ-curry : ∀ {Γ A B} → Subst (Γ , A , B) (Γ , A × B)
σ-curry = π B (‘ Z) • π A (‘ Z) • ids ∘ S_
```

$\vdots$

```
curry : ∀ {r : Γ , A , B ⊢ C}
  → [ curry ]≡ (ƛ ƛ r) ⇄ ƛ ⟪ σ-curry ⟫ r
```

$\vdots$

Equivalencias entre términos

$$\lambda x^{A\times B}.r \rightleftarrows \lambda y^{A}.\lambda z^{B}.r[\langle y, z\rangle/x]$$

Equivalencias entre términos

$$\lambda x^{A \times B}.r \rightleftarrows \lambda y^{A}.\lambda z^{B}.r[\langle y, z\rangle / x]$$

$$\lambda \lambda r[\langle 1, 0\rangle]$$

Equivalencias entre términos

$$\lambda x^{A \times B}.r \rightleftarrows \lambda y^{A}.\lambda z^{B}.r[\langle y, z\rangle / x]$$

$$\lambda\lambda r[\langle 1, 0\rangle]$$

```
σ-uncurry : ∀ {Γ A B} → Subst (Γ , A × B) (Γ , A , B)
σ-uncurry = ⟨ ‘ S Z , ‘ Z ⟩ • ids ∘ S_ ∘ S_
```

Equivalencias entre términos

$$\lambda x^{A \times B}.r \rightleftarrows \lambda y^{A}.\lambda z^{B}.r[\langle y, z\rangle/x]$$

$$\lambda\lambda r[\langle 1, 0\rangle]$$

```
σ-uncurry : ∀ {Γ A B} → Subst (Γ , A × B) (Γ , A , B)
σ-uncurry = ⟨ ‘ S Z , ‘ Z ⟩ • ids ∘ S_ ∘ S_
```

⋮

```
uncurry : ∀ {r : Γ , A × B ⊢ C}
  → [ sym curry ]≡ (ƛ r) ⇄ ƛ ƛ ⟪ σ-uncurry ⟫ r
```

⋮

Equivalencias entre términos

$\vdots$

$$
\begin{aligned}
&\xi\text{-}\langle,\rangle_1 : \forall \{r\ r' : \Gamma \vdash A\}\ \{s : \Gamma \vdash B\} \\
&\quad \to r \rightleftarrows r' \\
&\quad \to \langle\ r\ ,\ s\ \rangle \rightleftarrows \langle\ r'\ ,\ s\ \rangle
\end{aligned}
$$

$$
\begin{aligned}
&\xi\text{-}\langle,\rangle_2 : \forall \{r : \Gamma \vdash A\}\ \{s\ s' : \Gamma \vdash B\} \\
&\quad \to s \rightleftarrows s' \\
&\quad \to \langle\ r\ ,\ s\ \rangle \rightleftarrows \langle\ r\ ,\ s'\ \rangle
\end{aligned}
$$

$$
\begin{aligned}
&\xi\text{-}\pi : \forall \{t\ t' : \Gamma \vdash A \times B\} \\
&\quad \to t \rightleftarrows t' \\
&\quad \to \pi\ C\ \{p\}\ t \rightleftarrows \pi\ C\ \{p\}\ t'
\end{aligned}
$$

$$
\begin{aligned}
&\xi\text{-}\equiv : \forall \{t\ t' : \Gamma \vdash A\}\ \{iso : A \equiv B\} \\
&\quad \to t \rightleftarrows t' \\
&\quad \to ([\ iso\ ]\!\equiv t) \rightleftarrows ([\ iso\ ]\!\equiv t')
\end{aligned}
$$

Ejemplo

```
T₄ : ∀ {A B} → ∅ ⊢ (A ⇒ B ⇒ A) × (A ⇒ B ⇒ B)
T₄ = [ sym dist ]≡ (ƛ [ sym dist ]≡ (ƛ ⟨ ‘ (S Z) , ‘ Z ⟩))

_ : T₄ ⇄ [ sym dist ]≡ (ƛ ⟨ ƛ ‘ (S Z) , ƛ ‘ Z ⟩)
_ = ξ-≡ (ζ sym-dist-ƛ)
```

$$\lambda x.\lambda y.\langle x, y\rangle \rightleftarrows \lambda x.\langle \lambda y.x, \lambda y.y\rangle$$

Outline

Definición (SN)

Un término s es fuertemente normalizante si todos sus reductos también lo son:

$$\frac{\forall t'.\, t \hookrightarrow t' \implies t' \in SN}{t \in SN}$$

Normalización Fuerte

Definición (SN)

Un término s es fuertemente normalizante si todos sus reductos también lo son:

$$\frac{\forall t'.\, t \hookrightarrow t' \implies t' \in SN}{t \in SN}$$

```
data SN {Γ A} (t : Γ ⊢ A) : Set where
  sn : (∀ {t'} → t ↪ t' → SN t') → SN t
```

Ejemplos

```
SN⋆ : SN ⋆
SN⋆ = sn (λ ())
```

Ejemplos

```
SN⋆ : SN ⋆
SN⋆ = sn (λ ())
```

```
SN⟨,⟩ : SN ⟨ ⋆ , ‘ Z ⟩
SN⟨,⟩ = sn λ { (ξ-⟨,⟩₁ ())
             ; (ξ-⟨,⟩₂ ())}
```

Ejemplos

```
SN⋆ : SN ⋆
SN⋆ = sn (λ ())
```

```
SN⟨,⟩ : SN ⟨ ⋆ , ‘ Z ⟩
SN⟨,⟩ = sn λ { (ξ-⟨,⟩₁ ())
             ; (ξ-⟨,⟩₂ ())}
```

```
SNπ : SN (π ⊤ {inj₁ refl} ⟨ ⋆ , ‘ Z ⟩)
SNπ = sn λ { (ξ-π (ξ-⟨,⟩₁ ()))
           ; (ξ-π (ξ-⟨,⟩₂ ()))
           ; β-π₁ → SN⋆}
```

Approach 1

Realizamos inducción directa sobre los términos:

```
strong-norm : ∀ {Γ A} (t : Γ ⊢ A) → SN t
```

Realizamos inducción directa sobre los términos:

```
strong-norm : ∀ {Γ A} (t : Γ ⊢ A) → SN t
strong-norm ⋆          = sn (λ ())
strong-norm (‘ v)      = sn (λ ())
strong-norm (ƛ t)      = lemma-ƛ (strong-norm t)
strong-norm (a · b)    = lemma-· (strong-norm a) (strong-norm b)
strong-norm ⟨ a , b ⟩  = lemma-⟨,⟩ (strong-norm a) (strong-norm b)
strong-norm (π _ t)    = lemma-π (strong-norm t)
```

- Por H.I. sabemos que SN vale para los subterminos.

Approach 1

Realizamos inducción directa sobre los términos:

```
strong-norm : ∀ {Γ A} (t : Γ ⊢ A) → SN t
strong-norm ⋆         = sn (λ ())
strong-norm (‘ v)     = sn (λ ())
strong-norm (ƛ t)     = lemma-ƛ (strong-norm t)
strong-norm (a · b)   = lemma-· (strong-norm a) (strong-norm b)
strong-norm ⟨ a , b ⟩ = lemma-⟨,⟩ (strong-norm a) (strong-norm b)
strong-norm (π _ t)   = lemma-π (strong-norm t)
```

- Por H.I. sabemos que SN vale para los subterminos.
- Definimos algunos lemas para construir SN para todo el término.

Approach 1

Definimos una función auxiliar para realizar *pattern matching* sobre el paso de reducción:

```
lemma-ƛ : ∀ {Γ A B} → {t : Γ , B ⊢ A} → SN t → SN (ƛ t)
lemma-ƛ SNt = sn (λ step → aux SNt step)
  where aux : ∀ {t ƛt'} → SN t → (ƛ t) ↪ ƛt' → SN ƛt'
        aux (sn f) (ζ step) = lemma-ƛ (f step)
```

Definimos una función auxiliar para realizar *pattern matching* sobre el paso de reducción:

```
lemma-λ̄ : ∀ {Γ A B} → {t : Γ , B ⊢ A} → SN t → SN (λ̄ t)
lemma-λ̄ SNt = sn (λ step → aux SNt step)
  where aux : ∀ {t λ̄t'} → SN t → (λ̄ t) ↪ λ̄t' → SN λ̄t'
        aux (sn f) (ζ step) = lemma-λ̄ (f step)
```

$step : t \hookrightarrow t'$

Definimos una función auxiliar para realizar *pattern matching* sobre el paso de reducción:

```
lemma-λ : ∀ {Γ A B} → {t : Γ , B ⊢ A} → SN t → SN (λ t)
lemma-λ SNt = sn (λ step → aux SNt step)
  where aux : ∀ {t λt'} → SN t → (λ t) ↪ λt' → SN λt'
        aux (sn f) (ζ step) = lemma-λ (f step)
```

$step : t \hookrightarrow t'$
$f\ step : SN\ t'$

Approach 1

Definimos una función auxiliar para realizar *pattern matching* sobre el paso de reducción:

```
lemma-ƛ : ∀ {Γ A B} → {t : Γ , B ⊢ A} → SN t → SN (ƛ t)
lemma-ƛ SNt = sn (λ step → aux SNt step)
  where aux : ∀ {t ƛt'} → SN t → (ƛ t) ↪ ƛt' → SN ƛt'
        aux (sn f) (ζ step) = lemma-ƛ (f step)
```

$step : t \hookrightarrow t'$

$f\ step : SN\ t'$

lemma-ƛ $(f\ step) : SN\ (ƛ\ t')$

Approach 1

```
lemma-· : ∀ {a : Γ ⊢ A ⇒ B} {b : Γ ⊢ A} → SN a → SN b → SN (a · b)
lemma-· SNa SNb = sn (λ step → aux SNa SNb step)
  where aux : ∀ {a b t'} → SN a → SN b → a · b ↪ t' → SN t'
        aux (sn f) SNb (ξ-·₁ step) = lemma-· (f step) SNb
        aux SNa (sn f) (ξ-·₂ step) = lemma-· SNa (f step)
        aux {a = ƛ t} SNa SNb β-ƛ  = {! !} -- SN (t [ b ])
```

```
lemma-· : ∀ {a : Γ ⊢ A ⇒ B} {b : Γ ⊢ A} → SN a → SN b → SN (a · b)
lemma-· SNa SNb = sn (λ step → aux SNa SNb step)
  where aux : ∀ {a b t'} → SN a → SN b → a · b ↪ t' → SN t'
        aux (sn f) SNb (ξ-·₁ step) = lemma-· (f step) SNb
        aux SNa (sn f) (ξ-·₂ step) = lemma-· SNa (f step)
        aux {a = ƛ t} SNa SNb β-ƛ  = {! !} -- SN (t [ b ])
```

No podemos concluir nada sobre $t[b]$, de hecho, la substitución podría crear nuevos redexes dentro de t.

El lema de introducción del par es simple...

```
lemma-⟨,⟩ : ∀ {Γ A B} → {a : Γ ⊢ A} {b : Γ ⊢ B} → SN a → SN b → SN ⟨ a , b ⟩
lemma-⟨,⟩ SNa SNb = sn (λ step → aux SNa SNb step)
  where aux : ∀ {a b t'} → SN a → SN b → ⟨ a , b ⟩ ↪ t' → SN t'
        aux (sn f) SNb (ξ-⟨,⟩₁ step) = lemma-⟨,⟩ (f step) SNb
        aux SNa (sn f) (ξ-⟨,⟩₂ step) = lemma-⟨,⟩ SNa (f step)
```

Pero la eliminación también da problemas:

```
lemma-π : ∀ {Γ A B C p} → {t : Γ ⊢ A × B} → SN t → SN (π C {p} t)
lemma-π SNt = sn (λ step → aux SNt step)
  where aux : ∀ {C p t t'} → SN t → (π C {p} t) ↪ t' → SN t'
        aux (sn f) (ξ-π step)              = lemma-π (f step)
        aux {t = ⟨ a , b ⟩} (sn f) β-π₁ = {! !} -- SN a
        aux {t = ⟨ a , b ⟩} (sn f) β-π₂ = {! !} -- SN b
```

Approach 2

Definimos un SN más general que añade un predicado sobre el término:

```
data SN* {Γ A} (P : Γ ⊢ A → Set) (t : Γ ⊢ A) : Set where
  sn* : P t → (∀ {t'} → t ↪ t' → SN* P t') → SN* P t
```

Approach 2

Definimos un SN más general que añade un predicado sobre el término:

```
data SN* {Γ A} (P : Γ ⊢ A → Set) (t : Γ ⊢ A) : Set where
  sn* : P t → (∀ {t'} → t ↪ t' → SN* P t') → SN* P t
```

Es fácil ver que SN^* implica SN:

```
SN*-SN : ∀ {Γ A P} {t : Γ ⊢ A} → SN* P t → SN t
SN*-SN (sn* P SNt) = sn (λ step → SN*-SN (SNt step))
```

Definimos un SN más general que añade un predicado sobre el término:

```
data SN* {Γ A} (P : Γ ⊢ A → Set) (t : Γ ⊢ A) : Set where
  sn* : P t → (∀ {t'} → t ↪ t' → SN* P t') → SN* P t
```

Es fácil ver que SN^* implica SN:

```
SN*-SN : ∀ {Γ A P} {t : Γ ⊢ A} → SN* P t → SN t
SN*-SN (sn* P SNt) = sn (λ step → SN*-SN (SNt step))
```

Luego definimos la **interpretación del término**:

```
⟦_⟧ : ∀ {Γ A} → Γ ⊢ A → Set
⟦ (ƛ t) ⟧      = ∀ {u} → SN* ⟦_⟧ u → SN* ⟦_⟧ (t [ u ])
⟦ ⟨ a , b ⟩ ⟧ = SN* ⟦_⟧ a ⊗ SN* ⟦_⟧ b
⟦ t ⟧          = Top
```

Approach 2

La prueba queda divida en dos pasos:

- Primero se prueba el teorema fundamental:

```
adequacy : ∀ {Γ A} → (t : Γ ⊢ A) → SN* ⟦_⟧ t
```

Approach 2

La prueba queda divida en dos pasos:

- Primero se prueba el teorema fundamental:

```
adequacy : ∀ {Γ A} → (t : Γ ⊢ A) → SN* ⟦_⟧ t
```

- Luego es fácil probar SN:

```
strong-norm : ∀ {Γ A} (t : Γ ⊢ A) → SN t
strong-norm t = SN*-SN (adequacy t)
```

Approach 2

- La interpretación de la abstracción permite probar el caso de la aplicación.

```
lemma-· : ∀ {a : Γ ⊢ A ⇒ B} {b : Γ ⊢ A} → SN* ⟦_⟧ a → SN* ⟦_⟧ b → SN* ⟦_⟧ (a ·
lemma-· SNa SNb = sn* tt (λ step → aux SNa SNb step)
  where aux : ∀ {a b t'} → SN* ⟦_⟧ a → SN* ⟦_⟧ b → a · b ↪ t' → SN* ⟦_⟧ t'
        aux (sn* _ f) SNb (ξ-·₁ step) = lemma-· (f step) SNb
        aux SNa (sn* _ f) (ξ-·₂ step) = lemma-· SNa (f step)
        aux (sn* La _) SNb β-ƛ         = La SNb
```

Approach 2

- La interpretación del par resuelve el caso de la proyección.

```
lemma-π : ∀ {Γ A B C p} → {t : Γ ⊢ A × B} → SN* ⟦_⟧ t → SN* ⟦_⟧ (π C {p} t)
lemma-π SNt = sn* tt (λ step → aux SNt step)
  where aux : ∀ {C p t t'} → SN* ⟦_⟧ t → (π C {p} t) ↪ t' → SN* ⟦_⟧ t'
        aux (sn* _ f) (ξ-π step)                             = lemma-π (f step)
        aux {t = ⟨ a , b ⟩} (sn* (SNa , _ ) f) β-π₁ = SNa
        aux {t = ⟨ a , b ⟩} (sn* ( _ , SNb ) f) β-π₂ = SNb
```

Las introducciones ahora deben construir las interpretaciones.

- El caso del par es simple:

```
lemma-⟨,⟩ : ∀ {Γ A B} → {a : Γ ⊢ A} {b : Γ ⊢ B} → SN* ⟦_⟧ a → SN* ⟦_⟧ b → SN*
lemma-⟨,⟩ SNa SNb = sn* ( SNa , SNb ) (...)
```

Approach 2

Las introducciones ahora deben construir las interpretaciones.

- El caso del par es simple:

```
lemma-⟨,⟩ : ∀ {Γ A B} → {a : Γ ⊢ A} {b : Γ ⊢ B} → SN* ⟦_⟧ a → SN* ⟦_⟧ b → SN*
lemma-⟨,⟩ SNa SNb = sn* ( SNa , SNb ) (...)
```

- El caso de la abstracción es complejo:

```
lemma-λ : ∀ {Γ A B} → {t : Γ , B ⊢ A} → ⟦ λ t ⟧ → SN* ⟦_⟧ t → SN* ⟦_⟧ (λ t)
lemma-λ Lt (sn* _ f) = sn* Lt (...)
```

Las introducciones ahora deben construir las interpretaciones.

- El caso del par es simple:

```
lemma-⟨,⟩ : ∀ {Γ A B} → {a : Γ ⊢ A} {b : Γ ⊢ B} → SN* ⟦_⟧ a → SN* ⟦_⟧ b → SN*
lemma-⟨,⟩ SNa SNb = sn* ( SNa , SNb ) (...)
```

- El caso de la abstracción es complejo:

```
lemma-ƛ : ∀ {Γ A B} → {t : Γ , B ⊢ A} → ⟦ ƛ t ⟧ → SN* ⟦_⟧ t → SN* ⟦_⟧ (ƛ t)
lemma-ƛ Lt (sn* _ f) = sn* Lt (...)
```

Approach 2

```
adequacy (ƛ t)     = lemma-ƛ
                       (λ SNu → {! !}) -- SN* ⟦_⟧ (t [ u ])
                       (adequacy t)
adequacy (a · b)   = lemma-· (adequacy a) (adequacy b)
adequacy ⟨ a , b ⟩ = lemma-⟨,⟩ (adequacy a) (adequacy b)
adequacy (π _ t)   = lemma-π (adequacy t)
```

Approach 3

Generalizamos el teorema fundamental, vamos a probar que vale para cualquier t con cualquier substitución aplicada:

```
adequacy : ∀ {Γ Δ A} → (t : Γ ⊢ A) → (σ : Subst Γ Δ) → SN* ⟦_⟧ (⟪ σ ⟫ t)
```

Generalizamos el teorema fundamental, vamos a probar que vale para cualquier t con cualquier substitución aplicada:

$$\mathtt{adequacy} : \forall \{\Gamma\ \Delta\ A\} \to (t : \Gamma \vdash A) \to (\sigma : \mathtt{Subst}\ \Gamma\ \Delta) \to \mathtt{SN*}\ [\![\_]\!]\ (\langle\!\langle\ \sigma\ \rangle\!\rangle\ t)$$

Generalizamos el teorema fundamental, vamos a probar que vale para cualquier t con cualquier substitución aplicada:

```
adequacy : ∀ {Γ Δ A} → (t : Γ ⊢ A) → (σ : Subst Γ Δ) → SN* ⟦_⟧ (⟪ σ ⟫ t)
```

```
strong-norm : ∀ {Γ A} (t : Γ ⊢ A) → SN t
strong-norm t = (SN*-SN (adequacy t ids))
```

Approach 3

```
adequacy (` v) σ    = {!  !} -- SN* ⟦_⟧ (σ v)
adequacy (ƛ t) σ    =
  lemma-ƛ
    (λ {u = u} SNu → (adequacy t (u • σ)))
    (adequacy t (exts σ))
adequacy (a · b) σ = lemma-· (adequacy a σ) (adequacy b σ)
```

```
adequacy (` v) σ   = {!  !} -- SN* ⟦_⟧ (σ v)
adequacy (ƛ t) σ   =
  lemma-ƛ
    (λ {u = u} SNu → (adequacy t (u • σ)))
    (adequacy t (exts σ))
adequacy (a · b) σ = lemma-· (adequacy a σ) (adequacy b σ)


(⟪ exts σ ⟫ t)[ u ] = ⟪ u • σ ⟫ t
```

Approach 3

```
adequacy (` v) σ   = {!  !} -- SN* ⟦_⟧ (σ v)
adequacy (ƛ t) σ   =
  lemma-ƛ
    (λ {u = u} SNu → (adequacy t (u • σ)))
    (adequacy t (exts σ))
adequacy (a · b) σ = lemma-· (adequacy a σ) (adequacy b σ)
```

```
(⟪ exts σ ⟫ t) [ u ] = ⟪ u • σ ⟫ t
```

Ahora el problema está en el caso de las variables, una substitución arbitraria podría "romper" el término.

Approach 4

Definimos las **substituciones adecuadas**:

```
_⊨_ : ∀{Δ} → (Γ : Context) → (σ : Subst Γ Δ) → Set
Γ ⊨ σ = ∀ (v : Γ ∋ A) → SN* ⟦_⟧ (σ v)
```

Approach 4

Definimos las **substituciones adecuadas**:

```
_⊨_ : ∀{Δ} → (Γ : Context) → (σ : Subst Γ Δ) → Set
Γ ⊨ σ = ∀ (v : Γ ∋ A) → SN* ⟦_⟧ (σ v)
```

En particular ids es adecuada:

```
⊨ids : ∀{Γ} → Γ ⊨ ids
⊨ids _ = sn* tt (λ ())
```

Definimos las **substituciones adecuadas**:

```
_⊨_ : ∀{Δ} → (Γ : Context) → (σ : Subst Γ Δ) → Set
Γ ⊨ σ = ∀ (v : Γ ∋ A) → SN* ⟦_⟧ (σ v)
```

En particular ids es adecuada:

```
⊨ids : ∀{Γ} → Γ ⊨ ids
⊨ids _ = sn* tt (λ ())
```

El cons entre un término SN y una subst. adecuada, también es adecuada:

```
⊨_•_ : ∀{Γ Δ A σ} {t : Γ ⊢ A} → SN* ⟦_⟧ t → Δ ⊨ σ → (Δ , A) ⊨ (t • σ)
(⊨ t • σ) Z     = t
(⊨ t • σ) (S v) = σ v
```

Definimos las **substituciones adecuadas**:

```
_⊨_ : ∀{Δ} → (Γ : Context) → (σ : Subst Γ Δ) → Set
Γ ⊨ σ = ∀ (v : Γ ∋ A) → SN* ⟦_⟧ (σ v)
```

En particular ids es adecuada:

```
⊨ids : ∀{Γ} → Γ ⊨ ids
⊨ids _ = sn* tt (λ ())
```

El cons entre un término SN y una subst. adecuada, también es adecuada:

```
⊨_•_ : ∀{Γ Δ A σ} {t : Γ ⊢ A} → SN* ⟦_⟧ t → Δ ⊨ σ → (Δ , A) ⊨ (t • σ)
(⊨ t • σ) Z     = t
(⊨ t • σ) (S v) = σ v
```

Nuevamente fortalecemos el teorema:

```
adequacy : ∀ {Γ Δ A σ} → (t : Γ ⊢ A) → Γ ⊨ σ → SN* ⟦_⟧ (⟪ σ ⟫ t)
```

Approach 4

```
adequacy (` v) ⊨σ = ⊨σ v
adequacy (ƛ t) ⊨σ =
  lemma-ƛ
    (λ {u = u} SNu → (adequacy t (⊨ SNu • ⊨σ)))
    (adequacy t (⊨exts ⊨σ))
```

Approach 4

```
adequacy (` v) ⊨σ = ⊨σ v
adequacy (ƛ t) ⊨σ =
  lemma-ƛ
    (λ {u = u} SNu → (adequacy t (⊨ SNu • ⊨σ)))
    (adequacy t (⊨exts ⊨σ))
```

```
strong-norm : ∀ {Γ A} (t : Γ ⊢ A) → SN t
strong-norm t = (SN*-SN (adequacy t ⊨ids))
```

Approach 4

```
adequacy (` v) ⊨σ = ⊨σ v
adequacy (ƛ t) ⊨σ =
  lemma-ƛ
    (λ {u = u} SNu → (adequacy t (⊨ SNu • ⊨σ)))
    (adequacy t (⊨exts ⊨σ))
```

```
strong-norm : ∀ {Γ A} (t : Γ ⊢ A) → SN t
strong-norm t = (SN*-SN (adequacy t ⊨ids))
```

Approach 4

La extensión de una subst. adecuada es también es adecuada:

```
⊨exts : ∀{Γ Δ A} {σ : Subst Γ Δ} → Γ ⊨ σ → (Γ , A) ⊨ (exts σ)
⊨exts ⊨σ Z      = sn* tt (λ ())           -- SN* ⟦_⟧ (` Z)
⊨exts ⊨σ (S v) = SN*-rename S_ (⊨σ v)  -- SN* ⟦_⟧ (rename S_ (σ v))
```

Approach 4

La extensión de una subst. adecuada es también es adecuada:

```
⊨exts : ∀{Γ Δ A} {σ : Subst Γ Δ} → Γ ⊨ σ → (Γ , A) ⊨ (exts σ)
⊨exts ⊨σ Z      = sn* tt (λ ())           -- SN* ⟦_⟧ (` Z)
⊨exts ⊨σ (S v) = SN*-rename S_ (⊨σ v)  -- SN* ⟦_⟧ (rename S_ (σ v))
```

Los renombres preservan la propiedad SN^*:

```
SN*-rename : ∀{t} → (ρ : Rename Γ Δ) → SN* ⟦_⟧ t → SN* ⟦_⟧ (rename ρ t)
SN*-rename ρ (sn* Lt f) = sn* (⟦⟧-rename ρ Lt) (...)
```

La extensión de una subst. adecuada es también es adecuada:

```
⊨exts : ∀{Γ Δ A} {σ : Subst Γ Δ} → Γ ⊨ σ → (Γ , A) ⊨ (exts σ)
⊨exts ⊨σ Z       = sn* tt (λ ())           -- SN* ⟦_⟧ (` Z)
⊨exts ⊨σ (S v) = SN*-rename S_ (⊨σ v)  -- SN* ⟦_⟧ (rename S_ (σ v))
```

Los renombres preservan la propiedad SN^*:

```
SN*-rename : ∀{t} → (ρ : Rename Γ Δ) → SN* ⟦_⟧ t → SN* ⟦_⟧ (rename ρ t)
SN*-rename ρ (sn* Lt f) = sn* (⟦⟧-rename ρ Lt) (...)
```

Los renombres preservan la interpretación:

```
⟦⟧-rename : ∀{t : Γ ⊢ A} → (ρ : Rename Γ Δ) → ⟦ t ⟧ → ⟦ rename ρ t ⟧
⟦⟧-rename {t = ƛ n} ρ Ln = λ SNu → {!  !}
⋮
                                    -- SN* ⟦_⟧ ((rename ρ n) [ u ])
```

Generalizamos la interpretación de la abstracción:

$$
\begin{array}{ll}
[\![\_]\!] : \forall \{\Gamma\ A\} \to \Gamma \vdash A \to \mathtt{Set} & \\
[\![\ (\lambda\ t\ )\ ]\!] & = \forall \{\rho\ \}\{u\} \to \mathtt{SN*}\ [\![\_]\!]\ u \to \mathtt{SN*}\ [\![\_]\!]\ (\boxed{\langle\!\langle\ u \bullet (\mathtt{ids} \circ \rho)\ \rangle\!\rangle}\ t\ ) \\
[\![\ \langle\ a\ ,\ b\ \rangle\ ]\!] & = \mathtt{SN*}\ [\![\_]\!]\ a \otimes \mathtt{SN*}\ [\![\_]\!]\ b \\
[\![\ t\ ]\!] & = \mathtt{Top}
\end{array}
$$

Approach 5

Generalizamos la interpretación de la abstracción:

$$
\begin{array}{l}
[\![\_]\!] : \forall \, \{\Gamma \; A\} \to \Gamma \vdash A \to \mathtt{Set} \\
[\![\; (\lambda \; t) \;]\!] \qquad = \forall \, \{\rho\}\{u\} \to \mathtt{SN*} \; [\![\_]\!] \; u \to \mathtt{SN*} \; [\![\_]\!] \; (\langle\!\langle \; u \bullet (\mathtt{ids} \circ \rho) \; \rangle\!\rangle \; t) \\
[\![\; \langle \; a \; , \; b \; \rangle \;]\!] = \mathtt{SN*} \; [\![\_]\!] \; a \otimes \mathtt{SN*} \; [\![\_]\!] \; b \\
[\![\; t \;]\!] \qquad\quad = \mathtt{Top}
\end{array}
$$

$$
\begin{array}{l}
[\![]\!]\mathtt{-rename} : \forall\{\Gamma \; \Delta \; A\} \; \{t : \Gamma \vdash A\} \to (\rho : \mathtt{Rename} \; \Gamma \; \Delta) \to [\![\; t \;]\!] \to [\![\; \mathtt{rename} \; \rho \; t \;]\!] \\
[\![]\!]\mathtt{-rename} \; \{t = \lambda \; n\} \; \rho \; Ln \; \{\rho_1\} = \lambda \; \{ \; SNu \to Ln \; \{\rho = \rho_1 \circ \rho\} \; SNu \; \}
\end{array}
$$

$\vdots$

Approach 5

Generalizamos la interpretación de la abstracción:

$$
\begin{aligned}
&[\![\_]\!] : \forall \{\Gamma\ A\} \to \Gamma \vdash A \to \mathsf{Set}\\
&[\![\ (\lambda\ t)\ ]\!] \quad = \forall\ \{\rho\ \}\{u\} \to \mathtt{SN*}\ [\![\_]\!]\ u \to \mathtt{SN*}\ [\![\_]\!]\ (\langle\!\langle\ u\ \bullet\ (\mathtt{ids} \circ \rho)\ \rangle\!\rangle\ t)\\
&[\![\ \langle\ a\ ,\ b\ \rangle\ ]\!] = \mathtt{SN*}\ [\![\_]\!]\ a \otimes \mathtt{SN*}\ [\![\_]\!]\ b\\
&[\![\ t\ ]\!] \qquad\quad = \mathtt{Top}
\end{aligned}
$$

$$
\begin{aligned}
&[\![]\!]\text{-}\mathtt{rename} : \forall\{\Gamma\ \Delta\ A\}\ \{t\ :\ \Gamma \vdash A\} \to (\rho\ :\ \mathtt{Rename}\ \Gamma\ \Delta) \to [\![\ t\ ]\!] \to [\![\ \mathtt{rename}\ \rho\ t\ ]\!]\\
&[\![]\!]\text{-}\mathtt{rename}\ \{t\ =\ \lambda\ n\}\ \rho\ Ln\ \{\rho_1\} = \lambda\ \{\ SNu \to Ln\ \{\rho = \rho_1 \circ \rho\}\ SNu\ \}
\end{aligned}
$$

$\vdots$

La composición entre un renombre y una subst. adecuada, también es adecuada:

$$
\begin{aligned}
&\vDash\mathtt{rename}\ : \forall\{\Gamma\ \Delta\ \Delta_1\ \sigma\} \to \Gamma \vDash \sigma \to (\rho\ :\ \mathtt{Rename}\ \Delta\ \Delta_1) \to \Gamma \vDash (\langle\!\langle\ \mathtt{ids} \circ \rho\ \rangle\!\rangle \circ \sigma)\\
&\vDash\mathtt{rename}\ \vDash\sigma\ \rho\ v\ =\ (\mathtt{SN*\text{-}rename}\ \rho\ (\vDash\sigma\ v))
\end{aligned}
$$

Approach 5

```
adequacy (ƛ t) ⊨σ =
  lemma-ƛ
    (λ { {ρ} SNu → (adequacy t (⊨ SNu • (⊨rename ⊨σ ρ)))})
    (adequacy t (⊨exts ⊨σ))
```

Approach 5

```
adequacy (λ t) ⊨σ =
  lemma-λ
    (λ { {ρ} SNu → (adequacy t (⊨ SNu • (⊨rename ⊨σ ρ)))})
    (adequacy t (⊨exts ⊨σ))
```

Prueba para Sistema I

Definimos la SN para la unión entre $\hookrightarrow$ y $\rightleftarrows$:

```
_⇝_ : ∀ {Γ A} → (t t' : Γ ⊢ A) → Set
t ⇝ t' = t ↪ t' ⊎ t ⇄ t'

data SN {Γ A} (t : Γ ⊢ A) : Set where
  sn : (∀ {t'} → t ⇝ t' → SN t') → SN t
```

Prueba para Sistema I

Definimos la SN para la unión entre $\hookrightarrow$ y $\rightleftarrows$:

```
_⇝_ : ∀ {Γ A} → (t t' : Γ ⊢ A) → Set
t ⇝ t' = t ↪ t' ⊎ t ⇄ t'

data SN {Γ A} (t : Γ ⊢ A) : Set where
  sn : (∀ {t'} → t ⇝ t' → SN t') → SN t
```

La suma $\uplus$ funciona igual al **Either** de Haskell:

Prueba para Sistema I

Definimos la SN para la unión entre $\hookrightarrow$ y $\rightleftarrows$:

```
_⇝_ : ∀ {Γ A} → (t t' : Γ ⊢ A) → Set
t ⇝ t' = t ↪ t' ⊎ t ⇄ t'

data SN {Γ A} (t : Γ ⊢ A) : Set where
  sn : (∀ {t'} → t ⇝ t' → SN t') → SN t
```

La suma $\uplus$ funciona igual al **Either** de Haskell:

$$\text{data } \textbf{Either } a\ b = \textbf{Left } a \mid \textbf{Right } b$$

Prueba para Sistema I

Definimos la SN para la unión entre $\hookrightarrow$ y $\rightleftarrows$:

```
_⇝_ : ∀ {Γ A} → (t t' : Γ ⊢ A) → Set
t ⇝ t' = t ↪ t' ⊎ t ⇄ t'

data SN {Γ A} (t : Γ ⊢ A) : Set where
  sn : (∀ {t'} → t ⇝ t' → SN t') → SN t
```

La suma $\uplus$ funciona igual al **Either** de Haskell:

$$\text{data } \textbf{Either } a\ b = \textbf{Left } a \mid \textbf{Right } b$$

Donde **Left** y **Right** se llaman inj_1 y inj_2 resp.

Prueba para Sistema I

Definimos la SN para la unión entre $\hookrightarrow$ y $\rightleftarrows$:

```
_⇝_ : ∀ {Γ A} → (t t' : Γ ⊢ A) → Set
t ⇝ t' = t ↪ t' ⊎ t ⇄ t'

data SN {Γ A} (t : Γ ⊢ A) : Set where
  sn : (∀ {t'} → t ⇝ t' → SN t') → SN t
```

La suma $\uplus$ funciona igual al **Either** de Haskell:

$$\text{data } \textbf{Either } a\ b = \textbf{Left } a \mid \textbf{Right } b$$

Donde **Left** y **Right** se llaman inj_1 y inj_2 resp.

```
data SN* {Γ A} (P : Γ ⊢ A → Set) (t : Γ ⊢ A) : Set where
  sn* : P t → (∀ {t'} → t ⇝ t' → SN* P t') → SN* P t
```

Prueba para Sistema I

Se agrega el caso en adequacy:

```
⋮
adequacy ([ iso ]≡ n) ⊨σ = lemma-≡ (adequacy n ⊨σ)
⋮
```

Se agrega el caso en adequacy:

```
⋮
adequacy ([ iso ]≡ n) ⊨σ = lemma-≡ (adequacy n ⊨σ)
⋮
```

El caso de la congruencia es simple:

```
lemma-≡ : ∀ {t : Γ ⊢ A} → SN* ⟦_⟧ t → SN* {A = B} ⟦_⟧ ([ iso ]≡ t)
lemma-≡ SN*t = sn* tt (aux SN*t)
where aux : ∀ {Γ A iso t'} → {t : Γ ⊢ A} →
          SN* ⟦_⟧ t → ([ iso ]≡ t) ⇝ t' → SN* ⟦_⟧ t'
      aux (sn* _ f) (inj₁ (ξ-≡ step)) = lemma-≡ (f (inj₁ step))
      aux (sn* _ f) (inj₂ (ξ-≡ step)) = lemma-≡ (f (inj₂ step))
      ⋮
```

Para las equivalencias la idea es:

- "Desarmar" las hipótesis del lado izquierdo.
- Reconstruir $SN*$ del lado derecho usando los lemas anteriores.

```
⋮
aux (sn* ( SNr , SNs ) _) (inj₂ comm)     = lemma-⟨,⟩ SNs SNr
aux (sn* ( SNr , SNs ) _) (inj₂ sym-comm) = lemma-⟨,⟩ SNs SNr
⋮
```

Caso CURRY

$$\lambda x^A.\lambda y^B.t \rightleftarrows \lambda z^{A\times B}.t[\pi_A(z)/x, \pi_B(z)/y]$$

Caso CURRY

$$\lambda x^A.\lambda y^B.t \rightleftarrows \lambda z^{A\times B}.t[\pi_A(z)/x, \pi_B(z)/y]$$

Por hipótesis sabemos:

$$\forall u_1, u_2 \in SN* \Longrightarrow t[u_1/x, u_2/y] \in SN*$$

$$\lambda x^A.\lambda y^B.t \rightleftarrows \lambda z^{A\times B}.t[\pi_A(z)/x, \pi_B(z)/y]$$

Por hipótesis sabemos:

$$\forall u_1, u_2 \in SN* \Longrightarrow t[u_1/x, u_2/y] \in SN*$$

Debemos probar:

$$\lambda z.t[\pi_A(z)/x, \pi_B(z)/y] \in SN*$$

$$\lambda x^A.\lambda y^B.t \rightleftarrows \lambda z^{A\times B}.t[\pi_A(z)/x, \pi_B(z)/y]$$

Por hipótesis sabemos:

$$\forall u_1, u_2 \in SN* \Longrightarrow t[u_1/x, u_2/y] \in SN*$$

Debemos probar:

$$\lambda z.t[\pi_A(z)/x, \pi_B(z)/y] \in SN*$$

$$\forall u \in SN* \Longrightarrow (t[\pi_A(z)/x, \pi_B(z)/y])[u/z]$$

Caso CURRY

$$\lambda x^A.\lambda y^B.t \rightleftarrows \lambda z^{A\times B}.t[\pi_A(z)/x, \pi_B(z)/y]$$

Por hipótesis sabemos:

$$\forall u_1, u_2 \in SN* \Longrightarrow t[u_1/x, u_2/y] \in SN*$$

Debemos probar:

$$\lambda z.t[\pi_A(z)/x, \pi_B(z)/y] \in SN*$$

$$\forall u \in SN* \Longrightarrow (t[\pi_A(z)/x, \pi_B(z)/y])[u/z] = t[\pi_A(u)/x, \pi_B(u)/y] \in SN*$$

Caso CURRY

$$\lambda x^A.\lambda y^B.t \rightleftarrows \lambda z^{A\times B}.t[\pi_A(z)/x, \pi_B(z)/y]$$

Por hipótesis sabemos:

$$\forall u_1, u_2 \in SN* \implies t[u_1/x, u_2/y] \in SN*$$

Debemos probar:

$$\lambda z.t[\pi_A(z)/x, \pi_B(z)/y] \in SN*$$

$$\forall u \in SN* \implies (t[\pi_A(z)/x, \pi_B(z)/y])[u/z] = t[\pi_A(u)/x, \pi_B(u)/y] \in SN*$$

Basta con instanciar la hipótesis usando:

$$u_1 = \pi_A(u) \quad u_2 = \pi_B(u)$$

Caso CURRY

$$\lambda x^{A\times B}.t \rightleftarrows \lambda y^{A}.\lambda z^{B}.t[\langle y, z\rangle/x]$$

$$\lambda x^{A\times B}.t \rightleftarrows \lambda y^{A}.\lambda z^{B}.t[\langle y, z\rangle/x]$$

Por hipótesis sabemos:

$$\forall u \in SN* \implies t[u/x] \in SN*$$

$$\lambda x^{A\times B}.t \rightleftarrows \lambda y^{A}.\lambda z^{B}.t[\langle y, z\rangle/x]$$

Por hipótesis sabemos:

$$\forall u \in SN* \Longrightarrow t[u/x] \in SN*$$

Debemos probar:

$$\lambda \color{blue}{y}.\lambda \color{red}{z}.t[\langle \color{blue}{y}, \color{red}{z}\rangle/x] \in SN*$$

$$\lambda x^{A\times B}.t \rightleftarrows \lambda y^{A}.\lambda z^{B}.t[\langle y, z\rangle/x]$$

Por hipótesis sabemos:

$$\forall u \in SN* \Longrightarrow t[u/x] \in SN*$$

Debemos probar:

$$\lambda y.\lambda z.t[\langle y, z\rangle/x] \in SN*$$

$$\forall u_1, u_2 \in SN* \Longrightarrow ((t[\langle y, z\rangle/x])[u_1/y])[u_2/z]$$

$$\lambda x^{A\times B}.t \rightleftarrows \lambda y^{A}.\lambda z^{B}.t[\langle y, z\rangle/x]$$

Por hipótesis sabemos:

$$\forall u \in SN* \Longrightarrow t[u/x] \in SN*$$

Debemos probar:

$$\lambda y.\lambda z.t[\langle y, z\rangle/x] \in SN*$$

$$\forall u_1, u_2 \in SN* \Longrightarrow ((t[\langle y, z\rangle/x])[u_1/y])[u_2/z] = t[\langle u_1, u_2\rangle/x] \in SN*$$

$$\lambda x^{A\times B}.t \rightleftarrows \lambda y^{A}.\lambda z^{B}.t[\langle y, z\rangle/x]$$

Por hipótesis sabemos:

$$\forall u \in SN* \Longrightarrow t[u/x] \in SN*$$

Debemos probar:

$$\lambda y.\lambda z.t[\langle y, z\rangle/x] \in SN*$$

$$\forall u_1, u_2 \in SN* \Longrightarrow ((t[\langle y, z\rangle/x])[u_1/y])[u_2/z] = t[\langle u_1, u_2\rangle/x] \in SN*$$

Basta con instanciar la hipótesis usando:

$$u = \langle u_1, u_2\rangle$$

Outline

Conclusiones

- Introducción teórica.

Conclusiones

- Introducción teórica.
- Formalización de Sistema I en Agda.

- Introducción teórica.
- Formalización de Sistema I en Agda.
- Prueba de normalización fuerte.

Trabajo futuro

- Inferencia de tipos.

Trabajo futuro

- Inferencia de tipos.
- Formalizar Sistema I Polimórfico.

Trabajo futuro

- Inferencia de tipos.
- Formalizar Sistema I Polimórfico.
- Formalización con tipos extrínsecos.