Universidad Nacional de Córdoba

Facultad de Matemática, Astronomía y Física

Trabajo Especial

Presentado ante la Facultad de Matemática, Astronomía y Física como parte de los requerimientos para la obtención del grado de Licenciado en Ciencias de la Computación.

Estudio de semántica categórica para lenguajes Algol-like

Autor: Alejandro Emilio Gadea

Director: Miguel M. Pagano

Córdoba, Argentina, Septiembre de 2013

Resumen

Este trabajo consiste en la definición y estudio de tres lenguajes de programación. Los dos primeros serán lenguajes funcionales, uno con un sistema de tipos simple y otro con un sistema de tipos que soporta subtipado. El tercero es un lenguaje funcional con aspectos imperativos, perteneciente a la clase de lenguajes Algol-Like.

Para la definición semántica se utiliza teoría de categorías, en particular en la definición de los modelos semánticos. En particular, siguiendo propuestas de Reynolds y Oles, utilizamos categorías funtoriales para el lenguaje Algol-like.

Además se presentan las pruebas de ciertas propiedades deseables de las modelos semánticos dados: para el primer lenguaje nos enforcamos en la continuidad de las ecuaciones semánticas y en la corrección de la reducción;en el segundo lenguaje, desarrollamos la prueba de coherencia para diferentes derivaciones del mismo juicio; y para el tercero, probamos la naturalidad de las ecuaciones semánticas.

El trabajo teórico estuvo acompañado de la implementación de evaluadores en Idris, un lenguaje con tipos dependientes. Este desarrollo nos permitió descubrir algunas dificultades en ciertas ecuaciones semánticas.

Clasificación:

F.3.2 - Semantics of Programming Languages - Denotational semantics.
F.4.1 - Mathematical Logic - Lambda calculus and related systems.

Palabras clave:

Semántica denotacional, Categorías, Categoría funtorial, Stack discipline.

Índice general

1 **Introducción** 1

2 **Calculo lambda simplemente tipado** 3
- 2.1. Sintaxis de $\lambda^{\to}$ 3
- 2.2. Reglas de inferencia para $\lambda^{\to}$ 4
- 2.3. Semántica para $\lambda^{\to}$ 7
- 2.4. Continuidad de las ecuaciones semánticas de $\lambda^{\to}$ 10
- 2.5. Corrección de la regla-β para $\lambda^{\to}$ 16
- 2.6. Implementación en Idris 19

3 **Cálculo lambda tipado con subtipos** 21
- 3.1. Sintaxis para el subtipado 21
- 3.2. Semántica para $\lambda^{\leqslant}$ 22
- 3.3. Continuidad y coherencia 24
- 3.4. Implementación en Idris 32

4 **Lenguaje Algol-like** 35
- 4.1. Sintaxis de λ^{like} 36
- 4.2. Reglas de inferencia para λ^{like} 37
- 4.3. Semántica para λ^{like} 41
- 4.4. Naturalidad de las ecuaciones semánticas 56
- 4.5. Implementación en Idris 67

5 **Conclusión** 75

Bibliografía 77

Agradecimientos

A Miguel por ser un fantástico director *y jefe*, porque realmente me ayudo muchísimo en tantas cosas, ¡Muchas, muchas gracias!
Muchas gracias también a Daniel Fridlender y Leo Rodriguez, por estar en el jurado, pero sobre todo por las opiniones, recomendaciones y motivación durante el desarrollo del trabajo.
Al grupo entero Theona por la oportunidad de trabajar con ellos. Realmente me divertí mucho *trabajando*. ¡Gracias!
A mis viejos por todo el apoyo incondicional, gracias a ellos tuve la oportunidad de estudiar sin preocuparme por absolutamente nada mas que estudiar, mas allá de si me haya salido o no, la tranquilidad en el fondo estuvo. ¡Muchísimas gracias!
A mi hermana por ser genial y todo un ejemplo de persona, ¡gracias!
A mis primas, Euge y Flor, por tantos almuerzos, tardes de mates y cenas. A toda mi familia; abuelos, tíos, primos y *primos*, siempre preocupándose por mi, ¡gracias a todos!
A Eze por motivarme a estudiar en Córdoba y ser un excepcional amigo. ¡Gracias loco!
A todos mis amigos de la facu por la paciencia, *ganas de estudiar* y los buenos momentos. Para el particular grupo nocturno de *estudio*; Agus, Ale, Eze, Juan y Nacho, muchas gracias por el aguante, sobre todo por las risas y por ser geniales amigos. A Majo que estuvo muy cerca mio empujándome para delante en seguramente el momento mas difícil de mi estadía en Córdoba, ¡muchas gracias Majo, lo logre! y mucho mas.
A Dana, porque aguantarme yo se que no fue fácil, por todos los hermosos momentos juntos que no pienso olvidar. Gracias por ayudarme a madurar en tantos aspectos de la vida, me siento capaz de afrontar tantas cosas mas y es gracias a vos. ¡Gracias por aquel día de merienda en la piedra! y tantos otros ¡Gracias!
A mis amigos de La Pampa, que nunca tuvieron la menor idea de lo que hacia en Córdoba ja!; Ale K, Chavo, Higo, Mati, Manu, Marcos y Moya, gracias por los veranos de verdaderas vacaciones que me recargaban las pilas como no se imaginan. ¡Gracias!
A Mari, por toda la enorme ayuda con la facu. Por los momentos de risas interminables, las pasadas de largo programando y por lo mas importante, la amistad, gracias. Porque desde que paso lo que paso veo la vida de otra manera. ¡Muchas gracias!

En resumen, ¡gracias a todos mis amigos! Me gusta pensar que esto que logre es de todos nosotros.

Introducción

Como lo sugiere el titulo de la tesis, esta tratará sobre el estudio de semánticas categóricas para lenguajes Algol-like. Para esto vamos a estudiar tres lenguajes, $\lambda^{\rightarrow}$, $\lambda^{\leqslant}$ y λ^{like}, este ultimo perteneciente a la clase de lenguajes Algol-like y para el cual vamos a dar dos definiciones semánticas. La base de los lenguajes que vamos a estudiar se puede encontrar en los capítulos 15, 16 y 19 de [5] y [2], la idea de estudiar categóricamente esta en realidad motivado por la doble definición semántica que pretendemos dar de λ^{like}. Vamos a querer mostrar que podemos cambiar el significado semántico de un lenguaje sin cambiar sus ecuaciones, simplemente eligiendo ciertas categorías.

El trabajo empezará definiendo el lenguaje $\lambda^{\rightarrow}$, que es el cálculo lambda simplemente tipado con constantes y algunos operadores. Este lenguaje nos servirá para introducir conceptos generales sobre los lenguajes que vamos a estudiar, tales como los contextos, la semántica intrínseca para los tipos o la definición de semántica para un juicio de tipado como manera de representar el significado de las frases de nuestro lenguaje, entre otras cuestiones relacionas. Otro aporte de esta tesis es la implementación de cada uno de los lenguajes a estudiar utilizando un lenguaje de programación con tipos dependientes [1]. En este sentido empezar con $\lambda^{\rightarrow}$ nos sirvió para familiarizarnos con un lenguaje con tipos dependientes y esperamos también facilite la comprensión del lector.

El siguiente paso va a ser tomar a $\lambda^{\rightarrow}$ y extender su sistema de tipos para que incluya subtipado, a este nuevo lenguaje lo llamaremos $\lambda^{\leqslant}$. Algo interesante de este nuevo del lenguaje $\lambda^{\leqslant}$ es que vamos a reutilizar completamente todas las definiciones que hicimos en $\lambda^{\rightarrow}$. A su vez, esta forma de ir enriqueciendo cada lenguaje dará lugar a λ^{like}. Una ventaja de realizar las pruebas de una manera general, teniendo en cuenta esta jerarquía de lenguajes, es que las mismas serán validas, o servirán para completar las pruebas, para los lenguajes mas expresivos.

Para finalizar vamos a extender el lenguaje $\lambda^{\leqslant}$ con características imperativas, dando lugar al lenguaje λ^{like}. Este será un lenguaje Algol-like el cual combina aspectos imperativos y funcionales. Una particularidad importante de estos lenguajes es que los mismo cuentan con una semántica que respeta la stack discipline. Gracias a dar una semántica categórica para este lenguaje, en esta tesis mostramos como variando algunas elecciones se puede tener una semántica para λ^{like} sin stack discipline.

El segundo capítulo, puntualmente va a incluir la definición del lenguaje $\lambda^{\rightarrow}$, esto será presentar la sintáxis y las ecuaciones semánticas, además vamos a introducir todos los conceptos relacionados al sistemas de tipo simple, para finalizar vamos a dar pruebas de continuidad de las ecuaciones semánticas

y vamos a probar la corrección del lenguaje con respecto a la regla-β. Para finalizar presentamos una pequeña implementación en Idris.

En el tercer capítulo vamos a presentar todos los conceptos sobre un sistema de tipos con subtipado y adecuar entonces los tipos de $\lambda^{\rightarrow}$ para definir el sistema de tipos de $\lambda^{\leqslant}$, vamos a probar continuidad y como propiedad interesante teniendo subtipado, vamos a probar coherencia.

El cuarto capítulo tratará sobre el lenguaje λ^{like}, como en el segundo capítulo, vamos a presentar la sintáxis y las ecuaciones semánticas. Algo interesante será que ahora nuestra categoría principal será una categoría funtorial. Para este lenguaje, como ya mencionamos, vamos a dar dos semánticas distintas, además vamos a dar una prueba de la naturalidad de las ecuaciones semánticas.

La tesis requerirá conocimientos previos sobre teoría de dominios (predominios, dominios y funciones continuas), teoría de categorías (categorías concretas y funtoriales, producto, objeto exponencial y funtores) y semántica denotacional. Un conocimiento extra sobre tipos dependientes resultara practico exclusivamente para la parte de implementación.

Calculo lambda simplemente tipado

2

En este capítulo presentaremos un lenguaje funcional con tipado explicito simple. La idea de comenzar con este lenguaje, él cual ya tiene semántica bien conocida, es introducir los primeros conceptos relevantes para esta tesis sobre los sistemas de tipado simple, así como de la semántica denotácional del lenguaje, ademas es lo suficientemente simple como para no eclipsar el aprendizaje de conceptos sobre la implementación de su evaluador utilizando tipos dependientes.

2.1. Sintaxis de $\lambda^{\rightarrow}$

En esta sección vamos a introducir la sintáxis de los términos de $\lambda^{\rightarrow}$ y de los tipos. Empezamos introduciendo la sintaxis abstracta de los tipos simples.

$$\begin{aligned}\langle \mathrm{Type} \rangle ::= &\ \mathbf{bool} \mid \mathbf{int} \mid \mathbf{real} \\ &\mid \langle \mathrm{Type} \rangle \rightarrow \langle \mathrm{Type} \rangle\end{aligned}$$

Ahora bien **bool**, **int** y **real** representaran los conjuntos de booleanos, enteros y reales respectivamente, y si θ y θ' son tipos, $\theta \rightarrow \theta'$ sera la representación del conjunto de funciones que acepta valores de tipo θ y retorna valores de tipo θ', en ocasiones nos referiremos a este tipo como *tipo flecha*. Hay que mencionar que $\rightarrow$ asocia a derecha.

Con el lenguaje de los tipos definido podemos dar la sintáxis abstracta del lenguaje, notar que por el tipado explicito de la abstracción lambda nos hace falta tener definida la sintaxis de los tipos.

$$\begin{aligned}\langle \mathrm{Phrase} \rangle ::= &\ \langle \mathrm{PBool} \rangle \mid \langle \mathrm{PInt} \rangle \mid \langle \mathrm{PReal} \rangle \\ &\mid \odot\ \langle \mathrm{Phrase} \rangle \mid \langle \mathrm{Phrase} \rangle \circledcirc \langle \mathrm{Phrase} \rangle \\ &\mid \mathbf{if}\ \langle \mathrm{Phrase} \rangle\ \mathbf{then}\ \langle \mathrm{Phrase} \rangle\ \mathbf{else}\ \langle \mathrm{Phrase} \rangle \\ &\mid \langle \mathrm{Id} \rangle \\ &\mid \langle \mathrm{Phrase} \rangle \langle \mathrm{Phrase} \rangle \\ &\mid \lambda\ \langle \mathrm{Id} \rangle_{\theta}\ .\ \langle \mathrm{Phrase} \rangle \\ &\mid \mathbf{rec} \langle \mathrm{Phrase} \rangle\end{aligned}$$

$$\langle \mathrm{PBool} \rangle ::= \mathsf{True} \mid \mathsf{False}$$

$$\langle \mathrm{PNat} \rangle ::= 0 \mid 1 \mid 2 \mid \ldots$$

$$\langle \mathrm{PInt} \rangle ::= \ldots \mid -2 \mid -1 \mid \langle \mathrm{PNat} \rangle$$

$\langle \text{PReal} \rangle ::= \langle \text{PNat} \rangle.\{\langle \text{PNat} \rangle\}+$
$\quad\quad | -\langle \text{PNat} \rangle.\{\langle \text{PNat} \rangle\}+$

donde

$\langle \mathit{Id} \rangle$ es un conjunto numerable.
$\theta \in \langle \text{Type} \rangle$
$\odot \in \{-, \neg\}$ y
$\circledcirc \in \{+, -, *, /, \div, \textbf{rem}, \wedge, \vee, \Rightarrow, \Leftrightarrow, =, \neq, <, >, \leq, \geq\}$

Antes de continuar con la semántica de $\lambda^{\rightarrow}$ vamos a introducir 1. contextos, que van a permitir anotar el tipo de los identificadores de las frases 2. juicios de tipado, que será una relación entre un contexto, una frase y un tipo 3. las reglas de inferencia de tipos, que nos van a permitir decidir que juicios son validos y cuales no mediante una derivación 4. la semántica de los tipos.

Empecemos introduciendo la idea de contexto, este será una lista de pares que asocian un identificador con un tipo, con la restricción de que no pueden existir identificadores repetidos, esta restricción no es menor y mas adelante cuando tengamos definidas las reglas de inferencia vamos a ver alguna consecuencia de esta elección. En particular, para definir los contextos usamos una gramática, esto nos va a permitir tener una versión sintáctica de los contextos y ademas una semántica la cual, un poco mas adelante, vamos a necesitar para definir la semántica del lenguaje.

Definición 1. *Un contexto estará definido por la siguiente gramática,*

$\langle \text{Context} \rangle ::= \varnothing \mid \langle \text{Context} \rangle, \langle \text{Id} \rangle : \langle \text{Type} \rangle$

tal que dado cualquier contexto $\iota_0 : \theta_0, \ldots, \iota_n : \theta_n$, los identificadores $\iota_0, \ldots, \iota_n$ son todos distintos.

La razón por la que la definición de contextos tiene la restricción de no tener identificadores repetidos la dejamos para mas adelante cuando terminemos de introducir los juicios de tipado, que es lo que empezaremos a hacer a continuación.

Definición 2. *Un juicio de tipado sera una relación entre un contexto π, una frase del lenguaje e y un tipo θ que establecerá que bajo el contexto π la expresión e tiene tipo θ. Lo denotaremos como $\pi \vdash e : \theta$, en particular, cuando $\pi = \varnothing$ escribiremos $\vdash e : \theta$*

2.2. Reglas de inferencia para $\lambda^{\rightarrow}$

Antes definimos los contextos como listas de pares y dimos restricciones en el sentido de que no cualquier lista de pares era una contexto valido, ahora necesitamos algo parecido para los juicios de tipado, queremos una manera de decidir que juicios son validos y cuales no, para esto vamos a definir las reglas de inferencia, empecemos estableciendo algunas metavariables generales:

$$
\begin{array}{ll}
\theta \in \langle \text{Type} \rangle & \pi \in \langle \text{Context} \rangle \\
\delta \in \{\mathbf{bool}, \mathbf{int}, \mathbf{real}\} & \\
e \in \langle \text{Phrase} \rangle & \iota \in \langle \text{Id} \rangle \\
n, m \ \in \mathbb{N} & b \ \in \mathbb{B} \\
i \ \in \mathbb{Z} & r \ \in \mathbb{R}
\end{array}
$$

Ahora sí presentamos las reglas de inferencia:

`Ty Rule`: Constantes.

$$
\frac{}{\pi \vdash b : \mathbf{bool}} \qquad \frac{}{\pi \vdash i : \mathbf{int}} \qquad \frac{}{\pi \vdash r : \mathbf{real}}
$$

`Ty Rule`: Operadores básicos.

$$
\frac{\pi \vdash e : \mathbf{bool}}{\pi \vdash \neg e : \mathbf{bool}} \qquad \frac{\pi \vdash e : \mathbf{int}}{\pi \vdash -e : \mathbf{int}} \qquad \frac{\pi \vdash e : \mathbf{real}}{\pi \vdash -e : \mathbf{real}}
$$

$$
\frac{\pi \vdash e : \mathbf{int} \qquad \pi \vdash e' : \mathbf{int}}{\pi \vdash e \otimes e' : \mathbf{int}} \ \otimes \in \{+, -, *, /\ , \mathbf{rem}\}
$$

$$
\frac{\pi \vdash e : \mathbf{real} \qquad \pi \vdash e' : \mathbf{real}}{\pi \vdash e \otimes e' : \mathbf{real}} \ \otimes \in \{+, -, *\}
$$

$$
\frac{\pi \vdash e : \mathbf{bool} \qquad \pi \vdash e' : \mathbf{bool}}{\pi \vdash e \oslash e' : \mathbf{bool}} \ \oslash \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}
$$

$$
\frac{\pi \vdash e : \delta \qquad \pi \vdash e' : \delta}{\pi \vdash e \circledcirc e' : \mathbf{bool}} \ \delta \in \{\mathbf{int}, \mathbf{real}\}, \circledcirc \in \{<, >, \leq, \geq\}
$$

$$
\frac{\pi \vdash e : \delta \qquad \pi \vdash e' : \delta}{\pi \vdash e \ominus e' : \mathbf{bool}} \ \ominus \in \{=, \neq\}
$$

Veamos ahora las reglas referidas a la parte del cálculo lambda simplemente tipado, además de la regla para el tipado de un identificador.

`Ty Rule`: Aplicación.

$$
\frac{\pi \vdash e : \theta \to \theta' \qquad \pi \vdash e' : \theta}{\pi \vdash ee' : \theta'}
$$

`Ty Rule`: Operador de punto fijo.

$$
\frac{\pi \vdash e : \theta \to \theta}{\pi \vdash \mathbf{rec}\ e : \theta}
$$

En la regla de tipado de un identificador se observa cómo hace falta consultar el contexto para determinar el tipo del identificador. Esta consulta ayuda a chequear, por supuesto el tipo, pero además que el identificador no esté libre en el contexto de una frase mas compleja, ya que si este identificador fuera libre para la frase entonces no podríamos justificar la pertenencia al contexto.

Ty Rule: Identificador.

$$\frac{\iota : \theta \in \pi}{\pi \vdash \iota : \theta}$$

En la regla anterior consultábamos el contexto, en la regla para la abstracción lambda vamos a agregar un identificador con su respectivo tipo. La abstracción lambda la podríamos haber definido de dos maneras bien distintas, con tipado explicito o tipado implícito, la diferencia está en si especificamos el tipo del identificador o no.

Ty Rule: Abstracción lambda.

$$\frac{\pi, \iota : \theta \vdash e : \theta'}{\pi \vdash \lambda \iota_\theta . e : \theta \to \theta'}$$

Para terminar con la reglas de inferencia de este lenguaje tenemos la regla de la expresión condicional.

Ty Rule: Expresión condicional.

$$\frac{\pi \vdash b : \mathbf{bool} \qquad \pi \vdash e : \theta \qquad \pi \vdash e' : \theta}{\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \theta}$$

Ahora que hemos terminado de definir todas las reglas de inferencia, podemos mencionar la razón por la cual en la definición de nuestros contexto no permitimos tener identificadores repetidos. Tomemos la frase $\lambda \iota_{\mathbf{int}} . \lambda \iota'_{\mathbf{real}} .\ \iota'$, la derivación para tipar esta frase sería

$$\cfrac{\cfrac{\cfrac{\iota' : \theta \in \iota : \mathbf{int}, \iota' : \mathbf{real}}{\iota : \mathbf{int}, \iota' : \mathbf{real} \vdash \iota' : \theta}}{\iota : \mathbf{int} \vdash \lambda \iota'_{\mathbf{real}} .\ \iota' : \mathbf{real} \to \theta}}{\vdash \lambda \iota_{\mathbf{int}} . \lambda \iota'_{\mathbf{real}} .\ \iota' : \mathbf{int} \to \mathbf{real} \to \theta}$$

luego tenemos que θ debe ser **real** y podemos concluir el juicio de tipado, $\vdash \lambda \iota_{\mathbf{int}} . \lambda \iota'_{\mathbf{real}} .\ \iota' : \mathbf{int} \to \mathbf{real} \to \mathbf{real}$. Analicemos ahora que sucedería si reemplazáramos ι' por ι, es decir tuviéramos la siguiente frase $\lambda \iota_{\mathbf{int}} . \lambda \iota_{\mathbf{real}} .\ \iota$, la derivación es exactamente igual pero ahora surge un problema, θ podría ser o bien **int** o bien **real**, surge la pregunta entonces ¿ como saber que tipo es el correcto ?
Por otro lado, incluso reemplazando el tipo **real** por **int**, es decir tenemos la frase $\lambda \iota_{\mathbf{int}} . \lambda \iota_{\mathbf{int}} .\ \iota$, para la cual tenemos una derivación del juicio de tipado $\vdash \lambda \iota_{\mathbf{int}} . \lambda \iota_{\mathbf{int}} .\ \iota : \mathbf{int} \to \mathbf{int} \to \mathbf{int}$. La pregunta que surge es ¿ Cuando aplicamos esta frase a dos frases enteras, que valor debe tomar ι ?; supongamos tenemos el juicio anterior aplicado a dos frases enteras, $\vdash (\lambda \iota_{\mathbf{int}} . \lambda \iota_{\mathbf{int}} .\ \iota)\ 1\ 2 : \mathbf{int}$ el resultado

de tal evaluación debería 2 si nos basamos en la semántica operacional. Esto sucede porque la primera ocurrencia de ι no tiene participación en el alcance de la segunda ocurrencia, es decir, acá los parámetros de nuestras abstracciones lambda llevan el mismo nombre pero son distintos parámetros, esto pensando en la función que define nuestro ejemplo como una función que toma dos enteros y retorna otro.
Para finalizar entonces notemos que si restringimos los contextos no perdemos expresividad para definir funciones y si solucionamos posibles problemas de tipado. La restricción entonces implica que la frase $\lambda\iota_{\mathbf{int}}.\lambda\iota_{\mathbf{int}}.\ \iota$ no tiene ningún juicio de tipado valido.

2.3. Semántica para $\lambda^{\to}$

Ya tenemos definida la sintáxis de $\lambda^{\to}$ en conjunto con las reglas de inferencia para los distintos juicios de tipado de cada frase. Empecemos esta sección presentando la categoría semántica, que nombraremos **DC**, esta sera una categoría cartesiana cerrada, la cual iremos adaptando según las características de nuestro lenguaje. Tanto para $\lambda^{\to}$ como para $\lambda^{\leqslant}$ esta sera **Dom**.

Antes de continuar hagamos un repaso de cual es la categoría **Dom**, esta tiene por objetos a dominios[1] y por flechas a funciones continuas[2] . Además como característica que nos va a interesar, tenemos la existencia del operador de punto fijo que representaremos como $\mathbf{Y}_D$ con D un dominio y cuya existencia la garantiza [5, P 2.5], como ya hemos mencionado es $\mathcal{CCC}$, donde el producto entre dos objetos D y D' es el producto cartesiano de conjuntos y el exponencial $D^{D'}$ es el conjunto de funciones continuas de D' en D. Mencionamos además que con $\perp$ vamos a representar la no terminación, esto va a hacer falta ya que nuestro lenguaje básico incluye operador de punto fijo. La idea es que para interpretar los tipos elegimos objetos de esta categoría, y para los juicios elegimos morfismos.

Por otro lado definimos S_δ como el conjunto de valores que representa a cada tipo tal que $S_{\mathbf{real}}$ será $\mathbb{R}$, $S_{\mathbf{int}}$ será $\mathbb{Z}$ y $S_{\mathbf{bool}}$ será $\{\mathtt{true}, \mathtt{false}\}$, todos con orden llano.

Además de la categoría semántica vamos a definir dos categorías mas, una sera de los tipos ($\langle \mathtt{Type} \rangle$) y otra de contextos ($\langle \mathtt{Context} \rangle$). Lo interesante sobre utilizar categorías, sobre todo para el significado de los juicios de tipado, será la capacidad de adaptar las ecuaciones semánticas a las distintas características del lenguaje.

Definición 3. *La categoría de tipos, que nombraremos Θ, sera una categoría discreta tal que la colección de objetos esta definida como,*

$$\Theta_0 = \{\theta \mid \theta \in \langle \mathtt{Type} \rangle\}$$

[1] Conjunto parcialmente ordenado con menor elemento.
[2] Una función f entre predominios es continua si preserva el limite de cadenas.

Definición 4. *La categoría de contextos, que nombraremos* Π, *sera una categoría discreta tal que la colección de objetos esta definida como,*

$$\Pi_0 = \{\pi \mid \pi \in \langle \mathrm{Context} \rangle\}$$

Ahora sí, con los dominios necesarios, podemos definir dos funtores, uno entre Θ y **DC**, y otro entre Π y **DC**. Esto implica definir funciones, que mapee objetos en objetos y flechas en flechas, como Θ y Π son discreta, la definición de estas ultimas funciones es directa, por lo tanto vamos a definir el mapeo de objetos en objetos. En general cuando definamos funtores vamos a explicitar los índices, pero luego cuando los utilicemos vamos a omitirlos.

Definición 5. *Sea* $[\![_]\!] : \Theta \to \textbf{DC}$ *un funtor, tal que*
$[\![\delta]\!]_0 = (S_\delta)_\perp$
$[\![\theta \to \theta']\!]_0 = [\![\theta']\!]_0^{[\![\theta]\!]_0}$

Notar que como **DC** es $\mathcal{CCC}$ entonces podemos asegurar que existe el objeto exponencial.

Definición 6. *Sea* $[\![_]\!] : \Pi \to \textbf{DC}$ *un funtor, tal que*
$[\![\pi]\!]_0 = \prod_{\iota \in \mathrm{dom}\ \pi} [\![\pi\iota]\!]$

Notar que como **DC** es $\mathcal{CCC}$ entonces tenemos productos finitos, como los contextos son siempre finitos, la definición es correcta.

Ahora estamos en condiciones de definir las ecuaciones semánticas, para quien esté familiarizado con la definición de semántica denotacional de lenguajes sin tipado podemos comentar que las ecuaciones semánticas se definen sobre un juicio de tipado y no sobre las frases en sí. Esto implica entonces, que para evaluar un programa este antes debe haber pasado por el chequeo de tipos, es decir, tenemos una derivación del juicio de tipado.

Una ecuación semántica entonces estará definida por un juicio de tipado valido, supongamos $\pi \vdash e : \theta$, un objeto de la categoría **DC**, determinado por el funtor $[\![\pi]\!]$, otro objeto de **DC**, determinado por $[\![\theta]\!]$ y un funtor sobre el juicio de tipado, tal que, $[\![\pi \vdash e : \theta]\!] : [\![\pi]\!] \to [\![\theta]\!]$.

Agregamos a nuestra lista de metavariables, η con tipo $[\![\pi]\!]$, donde π estará determinado por el contexto donde usemos η.

Vamos a definir funciones

$(_)_\odot : (S_\delta \to S_\delta) \to ((S_\delta)_\perp \to (S_\delta)_\perp)$
$f_\odot = (\iota_\uparrow \circ f)_{\perp\!\!\perp}$

$(_)_\circledcirc : (S_\delta \to S_\delta \to S_{\delta'}) \to ((S_\delta)_\perp \to (S_\delta)_\perp \to (S_{\delta'})_\perp)$
$f_\circledcirc\, z\, z' = (\lambda x.\ (\lambda x'.\ \iota_\uparrow\ (f\ x\ x'))_{\perp\!\!\perp} z')_{\perp\!\!\perp} z$

Ahora sí comencemos con las definiciones de las ecuaciones semánticas, un detalle es que las definiciones consideran **DC** como **Dom**.

`Denotal Sem`: Constantes.

$$[\![\pi \vdash b : \mathbf{bool}]\!]\eta = \iota_{\uparrow} b \quad [\![\pi \vdash i : \mathbf{int}]\!]\eta = \iota_{\uparrow} i \quad [\![\pi \vdash r : \mathbf{real}]\!]\eta = \iota_{\uparrow} r$$

`Denotal Sem`: Operadores básicos.

$$[\![\pi \vdash -e : \mathbf{int}]\!] = -_{\odot} \circ [\![\pi \vdash e : \mathbf{int}]\!]$$

$$[\![\pi \vdash -e : \mathbf{real}]\!] = -_{\odot} \circ [\![\pi \vdash e : \mathbf{real}]\!]$$

$$[\![\pi \vdash \neg e : \mathbf{bool}]\!] = \neg_{\odot} \circ [\![\pi \vdash e : \mathbf{bool}]\!]$$

$$[\![\pi \vdash e \otimes e' : \mathbf{int}]\!]\eta = \otimes_{\circledcirc} [\![\pi \vdash e : \mathbf{int}]\!]\eta \; [\![\pi \vdash e' : \mathbf{int}]\!]\eta$$

$$\text{con } \otimes \in \{+, -, *, / , \mathbf{rem}\}$$

$$[\![\pi \vdash e \otimes e' : \mathbf{real}]\!]\eta = \otimes_{\circledcirc} [\![\pi \vdash e : \mathbf{int}]\!]\eta \; [\![\pi \vdash e' : \mathbf{int}]\!]\eta$$

$$\text{con } \otimes \in \{+, -, *\}$$

$$[\![\pi \vdash e \owedge e' : \mathbf{bool}]\!]\eta = \owedge_{\circledcirc} [\![\pi \vdash e : \mathbf{int}]\!]\eta \; [\![\pi \vdash e' : \mathbf{int}]\!]\eta$$

$$\text{con } \owedge \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$$

$$[\![\pi \vdash e \olessthan e' : \mathbf{bool}]\!]\eta = \olessthan_{\circledcirc} [\![\pi \vdash e : \mathbf{int}]\!]\eta \; [\![\pi \vdash e' : \mathbf{int}]\!]\eta$$

$$\text{con } \delta \in \{\mathbf{int}, \mathbf{real}\}, \olessthan \in \{<, >, \leq, \geq\}$$

$$[\![\pi \vdash e \ominus e' : \mathbf{bool}]\!]\eta = \ominus_{\circledcirc} [\![\pi \vdash e : \mathbf{int}]\!]\eta \; [\![\pi \vdash e' : \mathbf{int}]\!]\eta$$

$$\text{con } \ominus \in \{=, \neq\}$$

Hasta aquí tenemos definida la semántica para las expresiones enteras, reales y booleanas. Definamos ahora la parte interesante del lenguaje, esto es, la expresión condicional, aplicación, abstracción lambda, operador de punto fijo e identificador. Parecido a como hicimos para los operadores, definamos una función que llamaremos **IF** que irá de $D \times D \times (S_{\mathbf{bool}})_{\perp}$ en D, con D un dominio y cuya definición es,

$$\mathbf{IF}\langle t,\ f,\ b\rangle = \begin{cases} \perp & \text{si } b = \perp \\ t & \text{si } b \\ f & \text{si } \neg b \end{cases}$$

`Denotal Sem`: Expresión condicional.

$$[\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \theta]\!]\eta = \\ \mathbf{IF}([\![\pi \vdash e : \theta]\!]\eta,\ [\![\pi \vdash e' : \theta]\!]\eta,\ [\![\pi \vdash b : \mathbf{bool}]\!]\eta)$$

`Denotal Sem`: Aplicación.

$$[\![\pi \vdash ee' : \theta']\!]\eta = [\![\pi \vdash e : \theta \to \theta']\!]\eta([\![\pi \vdash e' : \theta]\!]\eta)$$

`Denotal Sem`: Abstracción lambda.

$$[\![\pi \vdash \lambda\iota_\theta.e : \theta \to \theta']\!] = [\![\pi, \iota : \theta \vdash e : \theta']\!] \circ ext_{\iota,\theta}$$

$$\begin{aligned} \text{donde } ext_{\iota,\theta} &: [\![\pi]\!] \to [\![\theta]\!] \to [\![\pi, \iota : \theta]\!] \\ ext_{\iota,\theta}\eta\, x &= [\, \eta \mid \iota : x \,] \end{aligned}$$

`Denotal Sem`: Operador de punto fijo.

$$[\![\pi \vdash \mathbf{rec}\ e : \theta]\!]\eta = \mathbf{Y}_\theta([\![\pi \vdash e : \theta \to \theta]\!]\eta)$$

`Denotal Sem`: Identificador.

$$[\![\pi \vdash \iota : \theta]\!]\eta = \eta\iota$$

Estas ultimas ecuaciones nos terminan de definir la semántica de nuestro primer lenguaje $\lambda^{\to}$.

2.4. Continuidad de las ecuaciones semánticas de $\lambda^{\to}$

Ahora que tenemos nuestro lenguaje definido tanto sintácticamente como semánticamente, veamos algunas propiedades y características. Tomemos la ecuación semántica de la abstracción lambda y analicémosla un poco, podemos empezar notando que el significado que le estamos dando a la abstracción lambda de nuestro lenguaje se corresponde exactamente con la noción de función matemática, recordando que $[\![\theta \to \theta']\!] = [\![\theta']\!]^{[\![\theta]\!]}$

$$[\![\pi \vdash \lambda\iota_\theta.e : \theta \to \theta']\!]\eta : [\![\theta']\!]^{[\![\theta]\!]}$$

es decir, como ya mencionamos antes, el significado de la abstracción lambda es una función en **Dom**, que va de $[\![\theta]\!]$ en $[\![\theta']\!]$. Mirando ahora la parte derecha,

$$[\![\pi, \iota : \theta \vdash e : \theta']\!] \circ (ext_{\iota,\theta}\eta) : [\![\theta]\!] \to [\![\theta']\!]$$

vemos que efectivamente es una función en el dominio correcto, además en

$$[\![\pi, \iota : \theta \vdash e : \theta']\!] : [\![\pi, \iota : \theta]\!] \to [\![\theta']\!]$$

la aplicación del entorno $ext_\iota\eta\, x$ extendido tipa perfecto con $[\![\pi, \iota : \theta]\!]$ dando como resultado el tipo esperado,

$$[\![\pi, \iota : \theta \vdash e : \theta']\!](ext_{\iota,\theta}\eta\, x) : [\![\theta']\!].$$

Acabamos de ver un poco más de cerca la ecuación semántica de la abstracción lambda, siguiendo con la etapa de análisis de las ecuaciones que hemos

definido vamos a tomar la aplicación y realizar un análisis categórico, él cual además nos servirá más adelante para probar la continuidad de la aplicación.

Tomemos un juicio de tipado $\pi \vdash ee' : \theta'$ cualquiera; recordemos además que dados dos tipos θ y θ', $[\![\theta']\!]^{[\![\theta]\!]}$ es el objeto exponencial en **Dom**, es decir, que además de este objeto tenemos una flecha ϵ,

$$[\![\theta']\!]^{[\![\theta]\!]} \times [\![\theta]\!] \xrightarrow{\epsilon} [\![\theta']\!]$$

tal que para una flecha $D \times [\![\theta]\!] \xrightarrow{f} [\![\theta']\!]$, con D un dominio, existe una única flecha $\widetilde{f}$ que hace conmutar el diagrama

$$\begin{array}{ccc} [\![\theta']\!]^{[\![\theta]\!]} \times [\![\theta]\!] & \xrightarrow{\epsilon} & [\![\theta']\!] \\ \uparrow {\scriptstyle \widetilde{f} \times 1_{[\![\theta]\!]}} & \nearrow {\scriptstyle f} & \\ D \times [\![\theta]\!] & & \end{array}$$

Por otro lado, $[\![\pi \vdash e' : \theta]\!] : [\![\pi]\!] \to [\![\theta]\!]$ y, como ya vimos hace un momento, $[\![\pi \vdash e : \theta \to \theta']\!] : [\![\pi]\!] \to [\![\theta']\!]^{[\![\theta]\!]}$, luego podemos definir una flecha $f = \textbf{uncurry}([\![\pi \vdash e : \theta \to \theta']\!])$ luego, $f : [\![\pi]\!] \times [\![\theta]\!] \to [\![\theta']\!]$. Por lo tanto tenemos el siguiente diagrama,

$$\begin{array}{ccc} [\![\theta']\!]^{[\![\theta]\!]} \times [\![\theta]\!] & \xrightarrow{\epsilon} & [\![\theta']\!] \\ \uparrow {\scriptstyle \widetilde{f} \times 1_{[\![\theta]\!]}} & \nearrow {\scriptstyle f} & \\ [\![\pi]\!] \times [\![\theta]\!] & & \end{array}$$

donde claramente $\widetilde{f} = [\![\pi \vdash e : \theta \to \theta']\!]$, puesto que dado una flecha g, la transpuesta de $\textbf{uncurry}(g) = g$.

Ahora, si tomamos una flecha g tal que $g = [\![\pi \vdash e' : \theta]\!]$, podemos obtener la siguiente flecha,

$$[\![\pi]\!] \xrightarrow{\langle 1_{[\![\pi]\!]}, 1_{[\![\pi]\!]} \rangle} [\![\pi]\!] \times [\![\pi]\!] \xrightarrow{1_{[\![\pi]\!]} \times g} [\![\pi]\!] \times [\![\theta]\!] \xrightarrow{f} [\![\theta']\!]$$

luego podemos combinar los dos diagramas, donde $(1_{[\![\pi]\!]} \times g) \circ \langle 1_{[\![\pi]\!]}, 1_{[\![\pi]\!]} \rangle = \langle 1_{[\![\pi]\!]}, g \rangle$ y obtener,

$$
\begin{array}{ccc}
[\![\theta']\!]^{[\![\theta]\!]} \times [\![\theta]\!] & \xrightarrow{\epsilon} & [\![\theta']\!] \\
\uparrow \widetilde{f} \times 1_{[\![\theta]\!]} & \nearrow f & \\
[\![\pi]\!] \times [\![\theta]\!] & & f \circ \langle 1_{[\![\pi]\!]}, g \rangle \\
\uparrow \langle 1_{[\![\pi]\!]}, g \rangle & \nearrow & \\
[\![\pi]\!] & &
\end{array}
$$

Ahora usando este diagrama podemos definir la ecuación semántica de la aplicación como,

$$[\![\pi \vdash ee' : \theta']\!] = \epsilon \circ \langle \widetilde{f}, g \rangle$$

donde $\langle \widetilde{f}, g \rangle = (\widetilde{f} \times 1_{[\![\theta]\!]}) \circ \langle 1_{[\![\pi]\!]}, g \rangle$.

Algo importante de notar es que esta definición se corresponden exactamente con la definición previa que hicimos, para terminar con este análisis comprobemos que efectivamente pasa esto. Pasando en limpio la nueva ecuación de la aplicación es,

$$[\![\pi \vdash ee' : \theta']\!] = \epsilon \circ \langle [\![\pi \vdash e : \theta \to \theta']\!], [\![\pi \vdash e' : \theta]\!] \rangle$$

tomando un η cualquiera tendremos,

$$[\![\pi \vdash ee' : \theta']\!]\eta = \epsilon \circ \langle [\![\pi \vdash e : \theta \to \theta']\!], [\![\pi \vdash e' : \theta]\!] \rangle \eta =$$

$$\epsilon \circ \langle [\![\pi \vdash e : \theta \to \theta']\!]\eta, [\![\pi \vdash e' : \theta]\!]\eta \rangle =$$

$$[\![\pi \vdash e : \theta \to \theta']\!]\eta([\![\pi \vdash e' : \theta]\!]\eta)$$

Continuidad

Ahora estamos en condiciones de demostrar una propiedad importante que deben cumplir nuestras ecuaciones semánticas, ser funciones continuas: no hay que olvidar que cuando decimos que, dado un juicio $\pi \vdash e : \theta$, su semántica será un morfismo $[\![\pi]\!] \to [\![\theta]\!]$ en la categoría **Dom**. Estamos comprometidos a que $[\![\pi \vdash e : \theta]\!]$ sea una función continua.

Empecemos enunciando algunas proposiciones que nos van a hacer útil para la prueba de continuidad, las pruebas de estas propiedades son mas bien simples y se pueden encontrar en [5].

Proposición 1.

1. *Si* P *y* P' *son predominios,* $P \rightarrow P'$ *es un predominio en el cual el limite de la cadena de funciones* $f_0 \sqsubseteq'' f_1 \sqsubseteq'' \cdots$ *es una función tal que*

$$(\bigsqcup_{i=0}^{\infty}{}'' f_i)x = \bigsqcup_{i=0}^{\infty}{}' f_i x$$

para todo $x \in P$*. Además, si* P *es un dominio,* $P \rightarrow P'$ *es un dominio cuyo menor elemento es una función tal que* $\bot'' x = \bot'$ *para todo* $x \in P$*.*

2. *Supongamos* P *y* P' *son predominios, entonces:*

   (a) *Una función constante de* P *en* P' *es la función continua de* P *en* P'*.*

   (b) *La función identidad en* P *es una función continua de* P *en* P*.*

   (c) *Si f es una función continua de* P *en* P' *y g es una función continua de* P' *en* P''*, entonces* $g \circ f$ *es una función continua de* P *en* P''*.*

   (d) *Dada una cadena* $f_0 \sqsubseteq'' f_1 \sqsubseteq'' \cdots$ *tal que cada* f_i *es una función continua de* P *en* P' *y dada otra cadena* $x_0 \sqsubseteq'' x_1 \sqsubseteq'' \cdots$ *de elementos de* P*, entonces vale*

   $$\bigsqcup_{i=0}^{\infty}\bigsqcup_{j=0}^{\infty} f_i x_j = \bigsqcup_{k=0}^{\infty} f_k x_k$$

   (e) *Dadas* n *funciones* f_i *de* $P \rightarrow P_i$ *tal que cada* f_i *es continua, entonces la función* $\langle f_0, f_1, \ldots, f_{n-1}\rangle$ *que satisface,*

   $$\langle f_0, f_1, \ldots, f_{n-1}\rangle x = \langle f_0 x, f_1 x, \ldots, f_{n-1} x\rangle$$

   es una función continua de P *en* $P_0 \times P_1 \times \ldots \times P_{n-1}$*.*

   (f) *Dado un predominio* P *y un dominio* D*, el operador* $(_)_{\perp\!\!\perp}$ *que extiende funciones de* $P \rightarrow D$ *en funciones de* $P_\bot \rightarrow D$*, es continuo.*

   (g) $\mathbf{Y}_D$ *es una función continua.*

Teorema 1. *Dado un juicio de tipado* $\pi \vdash e : \theta$ *la ecuación semántica* $[\![\pi \vdash e : \theta]\!]$ *es una función continua.*

Demostración. En la prueba vamos a proceder por inducción en la estructura de la derivación de los juicios de tipado. Supongamos tenemos una cadena interesante $\eta_0 \sqsubseteq \eta_1 \sqsubseteq \cdots$ en $[\![\pi]\!]$ y probemos que

$$[\![\pi \vdash e : \theta]\!](\bigsqcup_{i=0}^{\infty} \eta_i) = \bigsqcup_{i=0}^{\infty} [\![\pi \vdash e : \theta]\!]\eta_i.$$

Nuestra hipótesis inductiva nos dirá que, suponiendo tenemos una deriación $\pi \vdash e : \theta$ tal que $[\![\pi \vdash e : \theta]\!]$ es una función continua de $[\![\pi]\!]$ en $[\![\theta]\!]$ entonces para una derivación mas compleja $\pi \vdash e : \theta$ que contenga a la anterior $[\![\pi \vdash e : \theta]\!]$ es una función continua de $[\![\pi]\!]$ en $[\![\theta]\!]$.

- Casos base.
  - Para probar los casos base de los juicios de las constantes usamos Prop 1.2.a, que nos asegura que una función constante es función continua.
  - Supongamos ahora que tenemos $\pi \vdash \iota : \theta$, luego

    $$[\![\pi \vdash \iota : \theta]\!](\bigsqcup_{i=0}^{\infty} \eta_i) = (\bigsqcup_{i=0}^{\infty} \eta_i)\iota = \bigsqcup_{i=0}^{\infty} (\eta_i \iota) = \bigsqcup_{i=0}^{\infty} ([\![\pi \vdash \iota : \theta]\!]\eta_i)$$

    luego podemos concluir que $[\![\pi \vdash \iota : \theta]\!]$ es una función continua.

- Casos inductivos
  - Supongamos tenemos $\pi \vdash ee' : \theta$, como ya anticipamos antes, vamos a usar la versión categória de la ecuación,

    $$[\![\pi \vdash ee' : \theta]\!] = \epsilon \circ \langle [\![\pi \vdash e : \theta \to \theta']\!], [\![\pi \vdash e' : \theta]\!] \rangle$$

    luego por hipótesis inductiva $[\![\pi \vdash e : \theta \to \theta']\!]$ y $[\![\pi \vdash e' : \theta]\!]$ son funciones continuas, entonces $\langle [\![\pi \vdash e : \theta \to \theta']\!], [\![\pi \vdash e' : \theta]\!] \rangle$ es una función continua, además como ϵ es la flecha del exponencial también es continua; para terminar la composición de funciones continuas es una función continua. Por lo tanto la ecuación semántica $[\![\pi \vdash ee' : \theta]\!]$ es una función continua.
  - Supongamos tenemos $\pi \vdash \mathbf{rec}\ e : \theta$, luego

    $$[\![\pi \vdash \mathbf{rec}\ e : \theta]\!](\bigsqcup_{i=0}^{\infty} \eta_i) = \mathbf{Y}_{[\![\theta]\!]}([\![\pi \vdash e : \theta \to \theta]\!](\bigsqcup_{i=0}^{\infty} \eta_i))$$

    Para hacer mas fácil la lectura nombramos a $[\![\pi \vdash e : \theta \to \theta]\!]\eta_i = f_i$. Aplicando hipótesis inductiva en $[\![\pi \vdash e : \theta \to \theta]\!](\bigsqcup_{i=0}^{\infty} \eta_i)$ y la definición del operador de punto fijo

    $$\mathbf{Y}_{[\![\theta]\!]}([\![\pi \vdash e : \theta \to \theta]\!](\bigsqcup_{i=0}^{\infty} \eta_i)) = \mathbf{Y}_{[\![\theta]\!]}(\bigsqcup_{i=0}^{\infty} f_i) = (\bigsqcup_{i=0}^{\infty} f_i)\ (\mathbf{Y}_{[\![\theta]\!]}(\bigsqcup_{i=0}^{\infty} f_i))$$

    luego usando las proposiciones 2 (g) y 2 (d)

    $$(\bigsqcup_{i=0}^{\infty} f_i)\ (\mathbf{Y}_{[\![\theta]\!]}(\bigsqcup_{i=0}^{\infty} f_i)) = (\bigsqcup_{i=0}^{\infty} f_i)\ (\bigsqcup_{j=0}^{\infty} (\mathbf{Y}_{[\![\theta]\!]} f_j) =$$
    $$\bigsqcup_{i=0}^{\infty} \bigsqcup_{j=0}^{\infty} f_i(\mathbf{Y}_{[\![\theta]\!]} f_j) = \bigsqcup_{k=0}^{\infty} f_k(\mathbf{Y}_{[\![\theta]\!]} f_k) = \bigsqcup_{k=0}^{\infty} \mathbf{Y}_{[\![\theta]\!]} f_k =$$
    $$\bigsqcup_{k=0}^{\infty} \mathbf{Y}_{[\![\theta]\!]}([\![\pi \vdash e : \theta \to \theta]\!]\eta_k) = \bigsqcup_{k=0}^{\infty} [\![\pi \vdash \mathbf{rec}\ e : \theta]\!]\eta_k.$$

    Por lo tanto, nuestra ecuación semántica $[\![\pi \vdash \mathbf{rec}\ e : \theta]\!]$ es una función continua.

- Supongamos tenemos el juicio de tipado $\pi \vdash$ **if** b **then** e **else** $e' : \theta$, probemos que la función **IF** es continua. Supongamos una cadena $\langle t_0, f_0, b_0 \rangle \sqsubseteq \langle t_1, f_1, b_1 \rangle \sqsubseteq \ldots$, luego vamos a querer probar que $\mathbf{IF}(\bigsqcup_{i=0}^{\infty} \langle t_i, f_i, b_i \rangle) = \bigsqcup_{i=0}^{\infty} (\mathbf{IF}\langle t_i, f_i, b_i \rangle)$.
  $\mathbf{IF}(\bigsqcup_{i=0}^{\infty} \langle t_i, f_i, b_i \rangle) = \mathbf{IF}(\langle \bigsqcup_{i=0}^{\infty} t_i, \bigsqcup_{i=0}^{\infty} f_i, \bigsqcup_{i=0}^{\infty} b_i \rangle) =$

  $$= \begin{cases} \bigsqcup_{i=0}^{\infty} \bot & \text{si } \bigsqcup_{i=0}^{\infty} b_i = \bot \\ \bigsqcup_{i=0}^{\infty} t_i & \text{si } \bigsqcup_{i=0}^{\infty} b_i \\ \bigsqcup_{i=0}^{\infty} f_i & \text{si } \neg \bigsqcup_{i=0}^{\infty} b_i \end{cases}$$

  hagamos ahora un análisis por casos.

  - $\bigsqcup_{i=0}^{\infty} b_i = \bot$, luego lo que pasa es que tenemos la cadena

    $\langle t_0, f_0, \bot \rangle \sqsubseteq \langle t_1, f_1, \bot \rangle \sqsubseteq \ldots$

    a la que si le aplicamos la función **IF**, tal que tenemos la cadena

    $\mathrm{IF}\langle t_0, f_0, \bot \rangle \sqsubseteq \mathrm{IF}\langle t_1, f_1, \bot \rangle \sqsubseteq \ldots$,

    obtenemos la cadena

    $\bot \sqsubseteq \bot \sqsubseteq \ldots$ cuyo supremo es $\bigsqcup_{i=0}^{\infty} \bot$.

  - $\bigsqcup_{i=0}^{\infty} b_i = true$, entonces sucede similar que el caso anterior con la salvedad de que tenemos dos tipos de cadena posibles,

    $\langle t_0, f_0, \bot \rangle \sqsubseteq \langle t_1, f_1, \bot \rangle \sqsubseteq \ldots \sqsubseteq \langle t_{j-1}, f_{j-1}, \bot \rangle \sqsubseteq$
    $\langle t_j, f_j, true \rangle \sqsubseteq \langle t_{j+1}, f_{j+1}, true \rangle \sqsubseteq \ldots$

    o

    $\langle t_0, f_0, true \rangle \sqsubseteq \langle t_1, f_1, true \rangle \sqsubseteq \ldots$

    y lo que sucede es que para cualquiera de las dos cadenas al aplicarle **IF** el supremo será $\bigsqcup_{i=0}^{\infty} t_i$.

  - $\bigsqcup_{i=0}^{\infty} b_i = false$, análogo al caso anterior.

  Por lo tanto $\mathbf{IF}(\bigsqcup_{i=0}^{\infty} \langle t_i, f_i, b_i \rangle) = \bigsqcup_{i=0}^{\infty} (\mathbf{IF}\langle t_i, f_i, b_i \rangle)$.

  Por lo tanto, nuestra ecuación semántica $[\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \theta]\!]$ es una función continua.

- Los casos para los operadores binarios y unários son similares a la prueba de expresión condicional.
- Supongamos tenemos $\pi \vdash \lambda\iota_\theta.e : \theta \to \theta'$, por la definición de la ecuación semántica tenemos

  $$[\![\pi \vdash \lambda\iota_\theta.e : \theta \to \theta']\!] = [\![\pi, \iota : \theta \vdash e : \theta']\!] \circ ext_{\iota,\theta},$$

  probando entonces que $[\![\pi, \iota : \theta \vdash e : \theta']\!]$ y $ext_{\iota,\theta}$ son funciones continuas podemos usar la parte 1 del lema anterior que nos dice que la composición de funciones continuas es una función continua. Por hipótesis inductiva tenemos que $[\![\pi, \iota : \theta \vdash e : \theta']\!]$ es función continua, luego resta probar que ext_ι es continua de $[\![\pi]\!]$ en $[\![\theta]\!] \to [\![\pi, \iota : \theta]\!]$.

  Supongamos entonces una cadena $\eta_0 \sqsubseteq \eta_1 \sqsubseteq \ldots$ en $[\![\pi]\!]$ y probemos que $ext_{\iota,\theta}(\bigsqcup_{i=0}^{\infty} \eta_i) = \bigsqcup_{i=0}^{\infty}(ext_{\iota,\theta}\eta_i)$. Tomemos un x en $[\![\theta]\!]$ y un ι' cualquiera,

  $$ext_{\iota,\theta}(\bigsqcup_{i=0}^{\infty} \eta_i)\ x\ \iota' = [\ (\bigsqcup_{i=0}^{\infty} \eta_i)\ |\ \iota : x\ ]\iota'$$

  - Supongamos $\iota' = \iota$, luego

    $$[\ (\bigsqcup_{i=0}^{\infty} \eta_i)\ |\ \iota : x\ ]\iota' = x = \bigsqcup_{i=0}^{\infty} x\ \bigsqcup_{i=0}^{\infty}([\eta_i\ |\ \iota : x\ ]\iota') = \bigsqcup_{i=0}^{\infty}(ext_{\iota,\theta}\eta_i\ x\ \iota')$$

  - Supongamos $\iota' \neq \iota$, luego

    $$[\ (\bigsqcup_{i=0}^{\infty} \eta_i)\ |\ \iota : x\ ]\iota' = (\bigsqcup_{i=0}^{\infty} \eta_i)\iota' = \bigsqcup_{i=0}^{\infty}(\eta_i\iota') = \bigsqcup_{i=0}^{\infty}([\ \eta_i\ |\ \iota : x\ ]\iota') = \bigsqcup_{i=0}^{\infty}(ext_{\iota,\theta}\eta_i\ x\ \iota')$$

  por lo tanto, $ext_{\iota,\theta}$ es una función continua y podemos concluir que la ecuación semántica para la abstracción lambda también lo es.

□

2.5. Corrección de la regla-β para $\lambda^{\to}$

Lo que vamos a hacer a continuación es probar que nuestra semántica denotacional es correcta con respecto a la regla de reducción β, para esto vamos a necesitar el teorema de substitución. Para enunciar y probar este teorema vamos a tener que introducir un nuevo tipo de juicio de tipado así como también algunos conceptos de substitución. La intención es evitar tanto tecnicismo como podamos, presentemos rápidamente las herramientas que vamos a necesitar.

Empecemos introduciendo el operador Δ, de substitución, que será un mapa entre $\langle Id \rangle$ y $\langle Phrase \rangle$ que representará una substitución de identificadores

por frases, y la función FV de $\langle \mathrm{Phrase} \rangle$ en $\mathcal{P}(\langle \mathrm{Id} \rangle)$ que determinara los identificadores libres de una frase.

Además, vamos a necesitar de una nueva forma de juicio de tipado cuyo significado será que dado un identificador ι y una substitución Δ, la substitución de ι por $\Delta\iota$ en una frase conserva el tipado. Dados π, π' contextos y Δ, vamos a representar el nuevo juicio como $\pi \vdash \Delta : \pi'$, cuya única regla es:

$$\frac{\pi \vdash \Delta\iota : \pi'\iota \quad \text{para todo } \iota \text{ en } \mathrm{dom}(\pi)}{\pi \vdash \Delta : \pi'}$$

Teorema 2 (De substitución)**.** *Sean* π *y* π' *contextos,* Δ *una substitución y supongamos tenemos dos juicios de tipado* $\pi' \vdash e : \theta$, $\pi \vdash \Delta : \pi'$. *Asumamos además que* $\eta'\iota = [\![\pi \vdash \Delta\iota : \pi'\iota]\!]\eta$ *para todo* $\iota \in \mathrm{dom}\, \pi'$, *entonces* $[\![\pi' \vdash e : \theta]\!]\eta' = [\![\pi \vdash e/\Delta : \theta]\!]\eta$.

Demostración. Procedamos por inducción en la estructura de la derivación de los juicios de tipado.

- Casos base
  - Supongamos $\pi \vdash b : \mathbf{bool}$, luego tenemos $[\![\pi \vdash b : \mathbf{bool}]\!]\eta' = \iota_{\uparrow} b = [\![\pi' \vdash b : \mathbf{bool}]\!]\eta = [\![\pi' \vdash b/\Delta : \mathbf{bool}]\!]\eta$. Análogo para constantes de otros tipos.
  - Supongamos $\pi' \vdash \iota : \theta$, luego $[\![\pi' \vdash \iota : \theta]\!]\eta' = \eta'\iota$ ahora por hipótesis tenemos que $\eta'\iota = [\![\pi \vdash \Delta\iota : \pi'\iota]\!]\eta = [\![\pi \vdash \iota/\Delta : \theta]\!]\eta$.
- Casos inductivos,
  - Supongamos $\pi' \vdash e \ominus e' : \mathbf{bool}$, donde los casos para los demás operadores binarios y unários, expresión condicional, aplicación y operador de punto fijo son análogos, luego tenemos

    $[\![\pi' \vdash e \ominus e' : \mathbf{bool}]\!]\eta' = \ominus_{\circledcirc}\ ([\![\pi' \vdash e : \mathbf{bool}]\!]\eta')\ ([\![\pi' \vdash e' : \mathbf{bool}]\!]\eta')$

    por hipótesis inductiva en $\pi' \vdash e : \mathbf{bool}$ y $\pi' \vdash e' : \mathbf{bool}$

    $\ominus_{\circledcirc}\ ([\![\pi' \vdash e : \mathbf{bool}]\!]\eta')\ ([\![\pi' \vdash e' : \mathbf{bool}]\!]\eta') =$
    $\ominus_{\circledcirc}\ ([\![\pi \vdash e/\Delta : \mathbf{bool}]\!]\eta)\ ([\![\pi \vdash e'/\Delta : \mathbf{bool}]\!]\eta) =$
    $[\![\pi \vdash (e/\Delta) \ominus (e'/\Delta) : \mathbf{bool}]\!]\eta =$
    $[\![\pi \vdash (e \ominus e')/\Delta : \mathbf{bool}]\!]\eta.$

  - Supongamos $\pi' \vdash \lambda\iota_{\theta}.e : \theta \to \theta'$, luego

    $[\![\pi' \vdash \lambda\iota_{\theta}.e : \theta \to \theta']\!]\eta' = \lambda x.[\![\pi', \iota : \theta \vdash e : \theta']\!][\eta' \mid \iota : x]$

    para poder aplicar la hipótesis inductiva supongamos vale

    para todo $\kappa \in \mathrm{dom}(\widehat{\pi})$, donde $\widehat{\pi} = \pi', \iota : \theta$ vale que $[\eta' \mid \iota : x]\kappa = [\![\pi, \iota' : \theta \vdash \Delta'\iota' : \widehat{\pi}\kappa]\!][\eta \mid \iota' : x]$

donde $\Delta' = \Delta \mid \iota \rightsquigarrow \iota'$ y $\iota' \notin \bigcup_{\kappa \in FV(e)-\{\iota\}} (FV(\Delta\kappa))$.

luego

$\lambda x.[\![\pi, \iota' : \theta \vdash e/\Delta' : \theta']\!][\eta \mid \iota' : x] = [\![\pi \vdash \lambda\iota'_\theta.e/\Delta' : \theta \to \theta']\!]\eta = [\![\pi \vdash (\lambda\iota_\theta.e)/\Delta' : \theta \to \theta']\!]\eta$

Para finalizar probemos la suposición que nos permitía aplicar hipótesis inductiva. Tomamos un κ cualquiera, y vamos a separar en dos casos, si $\kappa \neq \iota$ entonces es directo aplicando la hipótesis. Veamos que sucede si suponemos $\kappa = \iota$,

$[\eta' \mid \iota : x]\kappa = x$

y

$[\![\pi, \iota' : \theta \vdash \delta'\kappa : \widehat{\pi}\kappa]\!][\eta \mid \iota' : x] = [\![\pi, \iota' : \theta \vdash \iota' : \theta]\!][\eta \mid \iota' : x] = [\eta \mid \iota' : x]\iota' = x.$

□

Ahora que tenemos enunciado y demostrado el teorema de substitución podemos continuar y demostrar que nuestra semántica denótacional es correcta con respecto a la regla β.

Teorema 3 (Corrección regla β)**.** *Dado un juicio de tipado* $\pi \vdash (\lambda\iota_\theta.e)e' : \theta'$ *vale,*

$$[\![\pi \vdash (\lambda\iota_\theta.e)e' : \theta']\!] = [\![\pi \vdash e/\iota \rightsquigarrow e' : \theta']\!]$$

Demostración. La idea de la prueba será aplicar las distintas ecuaciones semánticas de la aplicación y abstracción lambda hasta llegar el punto de poder aplicar nuestro teorema de susbtitución. Supongamos un η cualquiera,

$$[\![\pi \vdash (\lambda\iota_\theta.e)e' : \theta']\!]\eta = ([\![\pi \vdash \lambda\iota_\theta.e : \theta \to \theta']\!]\eta)\ ([\![\pi \vdash e' : \theta]\!]\eta) =$$

$$(\lambda x.[\![\pi, \iota : \theta \vdash e : \theta']\!][\,\eta \mid \iota : x\,])\ ([\![\pi \vdash e' : \theta]\!]\eta) =$$

$$[\![\pi, \iota : \theta \vdash e : \theta']\!][\,\eta \mid \iota : ([\![\pi \vdash e' : \theta]\!]\eta)\,],$$

hasta este punto lo que hemos hecho es usar las definiciones de la aplicación y abstracción, y resolver la aplicación del operador lambda de nuestro modelo semántico. A continuación vamos a redefinir con nombre a algunas expresiones y ver como podemos aplicar el teorema de substitución, vamos a renombrar $[\,\eta \mid \iota : ([\![\pi \vdash e' : \theta]\!]\eta)\,] = \eta'$ y $\pi, \iota : \theta = \pi'$, luego obtenemos

$$[\![\pi, \iota : \theta \vdash e : \theta']\!][\,\eta \mid \iota : ([\![\pi \vdash e' : \theta]\!]\eta)\,] = [\![\pi' \vdash e : \theta']\!]\eta'$$

ahora, para poder aplicar substitución tenemos que probar que para todo $\iota' \in \operatorname{dom}(\pi')$, $\eta'\iota' = [\![\pi \vdash \Delta\iota' : \pi'\iota']\!]\eta$ donde Δ sera la substitución $\iota \rightsquigarrow e'$ y la identidad para todo otro ι'. Supongamos un ι' cualquiera,

- Caso $\iota' \neq \iota, \eta'\iota' = \eta\iota' = [\![\pi \vdash \iota' : \pi'\iota']\!]\eta = [\![\pi \vdash \Delta\iota' : \pi'\iota']\!]\eta$.
- Case $\iota' = \iota, \eta'\iota' = \eta'\iota = [\![\pi \vdash e : \theta]\!]\eta = [\![\pi \vdash \Delta\iota : \pi'\iota]\!]\eta$.

luego aplicando substitución obtenemos, $[\![\pi' \vdash e : \theta']\!]\eta' = [\![\pi \vdash e/\Delta : \theta']\!]\eta$

por lo tanto, $[\![\pi \vdash (\lambda\iota_\theta.e)e' : \theta']\!]\eta = [\![\pi \vdash e/\iota \rightsquigarrow e' : \theta']\!]\eta$.

□

2.6. Implementación en Idris

La implementación del lenguaje se encuentra en:
`https://github.com/alexgadea/thesis/tree/master/Prototypes/Idris/LambdaArrow`

Sintaxis de los tipos

```
data PType = IntExp | RealExp | BoolExp
          | PType :-> PType
```

Semántica de los tipos

```
evalTy : PType -> Type
evalTy IntExp    = Int
evalTy RealExp   = Float
evalTy BoolExp   = Bool
evalTy (Theta :-> Theta') = evalTy Theta -> evalTy Theta'
```

Los contextos los vamos a separar en dos "versiones"la sintáctica y la semántica, la idea es implementar los contextos sintácticos de forma que no haya repetición de nombres de identificador. Es decir, un contexto va a ser una lista de identificadores y types;

$$i_0 : pt_0, \ldots, i_n : pt_n$$

para contextos semánticos simplemente es una lista de la siguiente manera,

$$evalTy\ pt_0, \ldots, evalTy\ pt_n$$

y tenemos una correspondencia lugar a lugar con el contexto sintáctico para buscar el valor.

Versión sintáctica

```
mutual
    data Ctx : Type where
        CtxUnit : Ctx
        Prepend : (p:Ctx) -> (i:Identifier) -> (pt:PType) ->
                  Fresh p i -> Ctx
    -- Representa si un identificador es fresco para un contexto.
    data Fresh : Ctx -> Identifier -> Type where
        FUnit : (i:Identifier) -> Fresh CtxUnit i
        FCons : (i:Identifier) -> (pt':PType) -> (i':Identifier) ->
                (p:Ctx) -> (fi':Fresh p i') -> so (i/=i') -> (Fresh p i) ->
                Fresh (Prepend p i' pt' fi') i
```

```
-- Representa la pertenencia de un identificador en un contexto.
data InCtx : Ctx -> Identifier -> Type where
    InHead : (p:Ctx) -> (i:Identifier) -> (pt:PType) ->
             (fi:Fresh p i) -> InCtx (Prepend p i pt fi) i
    InTail : (p:Ctx) -> (i:Identifier) -> (pt:PhraseType) ->
             (j:Identifier) -> (fj:Fresh p j) ->
             InCtx p i -> InCtx (Prepend p j pt fj) i
```

Versión semántica

```
evalCtx : Ctx -> Type
evalCtx CtxUnit = ()
evalCtx (Prepend p _ pt _) = (evalCtx p, evalTy pt)

-- Buscar el valor de un identificador en un contexto semántico.
search : (p:Ctx) -> (i:Identifier) -> (pt:PType) ->
         InCtx p i -> evalCtx p -> evalTy pt
search (Prepend _ i pt _) i pt (InHead _ i pt fi) (eta,v) = v
search (Prepend ctx j pt _) i pt (InTail _ _ pt j _ inc) (eta,_) = search ctx i pt inc eta

-- Actualizar el valor de un identificador.
update : (p:Ctx) -> evalCtx p -> (i:Identifier) ->
         (pt:PType) -> evalTy pt -> (fi:Fresh p i) -> evalCtx (Prepend p i pt fi)
update p eta i pt z fi = (eta,z)
```

Sintaxis de $\lambda^{\rightarrow}$

```
using (Pi:Ctx, Theta:PType, Theta':PType)
    data TypeJugdmnt : Ctx -> PType -> Type where
        I     : (i:Identifier) -> InCtx Pi i -> TypeJugdmnt Pi Theta

        CInt  : Int   -> TypeJugdmnt Pi IntExp
        CBool : Bool  -> TypeJugdmnt Pi BoolExp
        CReal : Float -> TypeJugdmnt Pi RealExp

        Lam   : (i:Identifier) -> (pt:PhraseType) -> (fi:Fresh Pi i) ->
                TypeJugdmnt (Prepend Pi i pt fi) Theta' ->
                TypeJugdmnt Pi (pt :-> Theta')
        App   : TypeJugdmnt Pi (Theta :-> Theta') ->
                TypeJugdmnt Pi Theta -> TypeJugdmnt Pi Theta'
        Rec   : TypeJugdmnt Pi (Theta :-> Theta) -> TypeJugdmnt Pi Theta

        If    : TypeJugdmnt Pi BoolExp ->
                TypeJugdmnt Pi Theta -> TypeJugdmnt Pi Theta ->
                TypeJugdmnt Pi Theta

        BinOp : (evalTy a -> evalTy b -> evalTy c) ->
                TypeJugdmnt Pi a -> TypeJugdmnt Pi b -> TypeJugdmnt Pi c
        UnOp  : (evalTy a -> evalTy b) ->
                TypeJugdmnt Pi a -> TypeJugdmnt Pi b
```

Semántica de $\lambda^{\rightarrow}$

```
eval : {Pi:Ctx} -> {Theta:PType} -> TypeJugdmnt Pi Theta -> evalCtx Pi -> evalTy Theta
eval {Pi=p} {Theta=pt} (I i iIn) eta = search p i pt iIn eta
eval (CInt x) eta = x
eval (CBool x) eta = x
eval (CReal x) eta = x
eval {Pi=p} (Lam i pt fi b) eta = \z => eval b (update p eta i pt z fi)
eval (App e e')   eta = (eval e eta) (eval e' eta)
eval (Rec e) eta = fix (eval e eta)
eval (If b e e')  eta = if eval b eta then eval e eta else eval e' eta
eval (BinOp op x y) eta = op (eval x eta) (eval y eta)
eval (UnOp op x) eta = op (eval x eta)
```

Cálculo lambda tipado con subtipos

3

En el capítulo anterior definimos la sintáxis y semántica del lenguaje que llamamos $\lambda^{\rightarrow}$, para esto en el transcurso introducimos distintos conceptos sobre el tipado. Ahora vamos estudiar a $\lambda^{\leqslant}$, el cual tiene como fin introducir conceptos sobre el subtipado, por esta razón mantenemos el lenguaje exactamente igual al lenguaje del capitulo anterior. La principal diferencia entonces estará en la definición de la categoría de tipos y en que vamos a agregar una nueva forma de juicio de tipado así como nuevas reglas.

Entre los tipos del lenguaje $\lambda^{\rightarrow}$, y por lo tanto los de $\lambda^{\leqslant}$, tenemos a **int** y **real** los cuales representan los conjuntos de enteros y reales matemáticos. Algo interesante a pensar es que los enteros forman parte de los reales; es decir, $\mathbb{Z} \subseteq \mathbb{R}$, luego surge la pregunta: ¿Existirá una forma de expresar esta relación como una relación entre los $\langle Type \rangle$?. La respuesta es sí y es el subtipado.

3.1. Sintaxis para el subtipado

Comencemos introduciendo la nueva forma de juicio de tipado, esta axiomatiza la relación de que θ es subtipo de θ' cuando $\theta \leq \theta'$.

Primero veamos reglas de inferencia generales a cualquier sistema de tipos, empecemos discutiendo una idea intuitiva de las reglas que serían deseables. Supongamos tenemos una frase e con tipo θ y además tenemos que $\theta \leq \theta'$ entonces podemos probar e tiene tipo θ'.

`Ty Rule:` Subsunción.

$$\frac{\pi \vdash e : \theta \qquad \theta \leq \theta'}{\pi \vdash e : \theta'}$$

Supongamos tenemos que la expresión e tiene tipo **int** y además que **int** es subtipo de **real**, luego quisiéramos poder decir que e tiene tipo **real**, además si suponemos un tipo **nat** que es subtipo de **int**, entonces deberíamos poder decir que **nat** es subtipo de **real**, es decir, tener transitividad entre los tipos, cualquier tipo es subtipo de él mismo, es decir, los tipos son reflexivos. Con esto vemos que la relación de subtipado es un preorden.

`Ty Rule:` Trans.

$$\frac{\theta \leq \theta' \qquad \theta' \leq \theta''}{\theta \leq \theta''}$$

`Ty Rule:` Reflex.

$$\frac{}{\theta \leq \theta}$$

Para finalizar, supongamos tenemos $\theta_0 \leq \theta'_0$ y $\theta_1 \leq \theta'_1$ y además que tenemos una expresión e que tiene tipo $\theta'_0 \to \theta_1$. Luego e puede aplicarse a elementos de tipo θ_0 y el resultado de tal aplicación puede ser un elemento de tipo θ'_1.

`Ty Rule`: Func.

$$\frac{\theta_0 \leq \theta'_0 \qquad \theta_1 \leq \theta'_1}{\theta'_0 \to \theta_1 \leq \theta_0 \to \theta'_1}$$

Estas reglas que mencionamos tiene la particularidad de ser generales para cualquier sistema de tipado, definamos ahora mas reglas en relación a nuestros tipos y lenguaje concreto, esto será definir la relación entre enteros y reales y vamos a agregar una mas, tal vez no sea lo más recomendado, por cuestiones semánticas, en cuanto al diseño del lenguaje pero es practico considerarla, para tener casos simples de transitividad, que es la relación entre booleanos y enteros.

`Ty Rule`: boolToint.

$$\frac{}{\textbf{bool} \leq \textbf{int}}$$

`Ty Rule`: intToreal.

$$\frac{}{\textbf{int} \leq \textbf{real}}$$

3.2. Semántica para $\lambda^{\leqslant}$

Ahora que tenemos definido la nueva forma de juicio de tipado para la relación entre los tipos, vamos a actualizar nuestra categoría de tipos, esta dejará de ser una categoría discreta y la clave está en que ahora nuestra relación entre los tipos determinará las flechas. Antes vimos que la relación que surgía entre los tipos daba lugar a un preorden entre estos, luego nuestra categoría de tipos será este preorden visto como categoría.

Definición 7. *La categoría de tipos, que nombraremos Θ, se define como sigue*

$$\Theta_0 = \{\theta \mid \theta \in \langle \text{Type} \rangle\}$$
$$\Theta_1(\theta, \theta') = \{\theta \xrightarrow{*} \theta' \mid \theta \leq \theta'\}$$

El preorden entre los tipos nos implica que dados dos tipos θ y θ', $|\Theta_1(\theta, \theta')| \leq 1$.

Además esta nueva definición nos impacta directamente en el funtor semántico $[\![\ ]\!] : \Theta \to \mathbf{DC}$, ahora tenemos que definir como actúa el funtor con respecto a las flechas.

Definición 8. *Sea $[\![\ ]\!] : \Theta \to \mathbf{DC}$ un funtor, tal que*

$$[\![\delta]\!]_0 = (S_\delta)_\perp$$
$$[\![\theta \to \theta']\!]_0 = [\![\theta']\!]_0^{[\![\theta]\!]_0}$$

$$[\![\textbf{\textit{bool}} \leq \textbf{\textit{int}}]\!]_1\ x = \begin{cases} 0 & \textit{si } x \\ 1 & \textit{si } \neg x \end{cases}$$

$[\![\textbf{\textit{int}} \leq \textbf{\textit{real}}]\!]_1 = \mathcal{J}$ *donde* $\mathcal{J}$ *la inyección de enteros en reales.*
$[\![\theta \leq \theta]\!]_1 = 1_{[\![\theta]\!]}$
$[\![\theta \leq \theta'']\!]_1 = [\![\theta' \leq \theta'']\!]_1 \circ [\![\theta \leq \theta']\!]_1$
$[\![(\theta_0 \to \theta_0') \leq (\theta_1 \to \theta_1')]\!]_1 = \lambda f \in [\![\theta_0 \to \theta_0']\!]_0 \,.\, [\![\theta_0' \leq \theta_1']\!]_1 \circ f \circ [\![\theta_1 \leq \theta_0]\!]_1$

Antes de seguir con la nueva definición de la categoría de contextos analicemos la definición $[\![(\theta_0 \to \theta_0') \leq (\theta_1 \to \theta_1')]\!]$. La idea será ver que la definición que dimos es correcta y además mostrar de que manera la podemos construir, comencemos notando que,

$$[\![\theta_1 \leq \theta_0]\!] : [\![\theta_1]\!] \to [\![\theta_0]\!] = \mathrm{Hom}_{\mathbf{Dom}}([\![\theta_1]\!], [\![\theta_0]\!])$$

$$[\![\theta_0' \leq \theta_1']\!] : [\![\theta_0']\!] \to [\![\theta_1']\!] = \mathrm{Hom}_{\mathbf{Dom}}([\![\theta_0']\!], [\![\theta_1']\!])$$

y definamos entonces dos funtores, uno covariante $\mathrm{Hom}([\![\theta_1]\!], _)$ y otro contravariante $\mathrm{Hom}(_, [\![\theta_1']\!])$.

Tomemos una función f en $[\![\theta_0 \to \theta_0']\!]$ cualquiera, luego

$$\mathrm{Hom}([\![\theta_1]\!], f) : \mathrm{Hom}([\![\theta_1]\!], [\![\theta_0]\!]) \to \mathrm{Hom}([\![\theta_1]\!], [\![\theta_0']\!]),$$

usando lo que notamos al principio podemos hacer, es decir, $[\![\theta_1 \leq \theta_0]\!] : \mathrm{Hom}_{\mathbf{Dom}}([\![\theta_1]\!], [\![\theta_0]\!])$, obtenemos

$$\mathrm{Hom}([\![\theta_1]\!], f)[\![\theta_1 \leq \theta_0]\!] = f \circ [\![\theta_1 \leq \theta_0]\!] : \mathrm{Hom}([\![\theta_1]\!], [\![\theta_0']\!]).$$

Si ahora hacemos algo similar usando el otro funtor tenemos,

$$\mathrm{Hom}(f \circ [\![\theta_1 \leq \theta_0]\!], [\![\theta_1']\!]) : \mathrm{Hom}([\![\theta_0']\!], [\![\theta_1']\!]) \to \mathrm{Hom}([\![\theta_1]\!], [\![\theta_1']\!]),$$

y aplicando el funtor como antes podemos llegar a la ecuación propuesta,

$$\mathrm{Hom}(f \circ [\![\theta_1 \leq \theta_0]\!], [\![\theta_1']\!])[\![\theta_0' \leq \theta_1']\!] = [\![\theta_0' \leq \theta_1']\!] \circ f \circ [\![\theta_1 \leq \theta_0]\!].$$

Luego podemos mencionar que el subtipado para un tipo $\theta \to \theta'$ es covariante para θ y contravariante para θ'.

La definición de la relación $\leq$ entre tipos nos permite ademas actualizar la definición de la categoría de contexto, de manera tal de definir $\leq$ entre contextos para que luego, como pasa con los tipos, esta relación sea una flecha en la categoría. Dados π y π' tal que $\mathrm{dom}\ \pi = \mathrm{dom}\ \pi'$, diremos que $\pi \leq \pi'$ cuando para todo $\iota \in \mathrm{dom}\ \pi$ se cumple $\pi\iota \leq \pi'\iota$.

Definición 9. *La categoría de contextos, que nombraremos* Π, *se define como sigue*

$$\Pi_0 = \{\pi \mid \pi \in \langle \mathrm{Context} \rangle\}$$
$$\Pi_1(\pi, \pi') = \{\pi \longrightarrow \pi' \mid \pi \leq \pi'\}$$

De igual manera que cuando dimos la nueva definición de Θ, podemos dar una actualización a la definición de Π.

Definición 10. *Sea* $[\![\ ]\!] : \Pi \to$ ***DC*** *un funtor, tal que*

$$[\![\pi]\!]_0 = \prod_{\iota \in \mathrm{dom}\ \pi} [\![\pi\iota]\!]$$
$$[\![\pi \leq \pi']\!]_1 = \prod_{\iota \in \mathrm{dom}\ \pi} [\![\pi\iota \leq \pi'\iota]\!]$$

Esta ultima definición da por terminado el trabajo de acomodar los dominios categóricos, además de las nuevas formas de jucios de tipado y su semántica respectiva.

Para completar la semántica del lenguaje $\lambda^{\leqslant}$, nos falta definir una ecuación semántica que relacione un juicio de tipado con una relación de orden entre dos tipos.

`Denotal Sem`: Subsunción.

$$[\![\pi \vdash e : \theta']\!] = [\![\theta \leq \theta']\!] \circ [\![\pi \vdash e : \theta]\!]$$

3.3. Continuidad y coherencia

Al igual que como hicimos para el lenguaje $\lambda^{\rightarrow}$ vamos a probar la continuidad de las ecuaciones semánticas de $\lambda^{\leqslant}$. En cuanto a esta prueba lo interesante es que es realmente simple, ya que usando la prueba de continuidad de $\lambda^{\rightarrow}$ simplemente nos resta probar la continuidad de Subsunción. Por otro lado además vamos a probar coherencia, esta propiedad describe que dados dos o mas derivaciones para un mismo juicio de tipado, todas estas derivaciones tiene el mismo significado semántico. Coherencia se vuelve interesante para nuestro lenguaje $\lambda^{\leqslant}$ ya que este tiene subtipado y esto es el que genera que un juicio pueda tener mas de una derivación, claramente esto no pasa para $\lambda^{\rightarrow}$ para el cual por cada juicio existe una sola derivación.

Empecemos probando la continuidad de las ecuaciones semánticas y luego introduzcamos mas detalles sobre la coherencia de $\lambda^{\leqslant}$.

Teorema 4. *Dado un juicio de tipado* $\pi \vdash e : \theta$ *la ecuación semántica* $[\![\pi \vdash e : \theta]\!]$ *es una función continua.*

Demostración. En la prueba vamos a proceder por inducción en la estructura de la derivación de los juicios de tipado. Además como ya mencionamos antes, solamente nos resta probar el caso inductivo para la semántica denotacional de Subsunción, supongamos entonces un juicio de tipado $\pi \vdash e : \theta'$, luego

$$[\![\pi \vdash e : \theta']\!] = [\![\theta \leq \theta']\!] \circ [\![\pi \vdash e : \theta]\!]$$

por hipótesis inductiva obtenemos que $[\![\pi \vdash e : \theta]\!]$ es una función continua, además por construcción de nuestra categória de tipos tenemos que $[\![\theta \leq \theta']\!]$ es una función continua también, por lo tanto utilizando que la composición de funciones continuas es una función continua concluimos que $[\![\pi \vdash e : \theta']\!]$ es función continua.

$\square$

Retomemos coherencia con un ejemplo simple considerando el identificador ι con tipo **real**, para esta frase y tipo existen cuatro derivaciones posibles.

Usando Subsunción:

$$\frac{\dfrac{\iota : \mathbf{int} \in \pi}{\pi \vdash \iota : \mathbf{int}} \qquad \mathbf{int} \leq \mathbf{real}}{\pi \vdash \iota : \mathbf{real}}$$

Usando Subsunción′:

$$\frac{\dfrac{\dfrac{\iota : \mathbf{bool} \in \pi'}{\pi' \vdash \iota : \mathbf{bool}} \qquad \mathbf{bool} \leq \mathbf{int}}{\pi' \vdash \iota : \mathbf{int}} \qquad \mathbf{int} \leq \mathbf{real}}{\pi' \vdash \iota : \mathbf{real}}$$

Usando Subsunción″:

$$\frac{\dfrac{\iota : \mathbf{bool} \in \pi''}{\pi'' \vdash \iota : \mathbf{bool}} \qquad \dfrac{\mathbf{bool} \leq \mathbf{int} \qquad \mathbf{int} \leq \mathbf{real}}{\mathbf{bool} \leq \mathbf{real}}}{\pi'' \vdash \iota : \mathbf{real}}$$

Usando la regla del identificador:

$$\frac{\iota : \mathbf{real} \in \pi'''}{\pi''' \vdash \iota : \mathbf{real}}$$

Ahora bien, algo importante a notar es que mencionábamos que coherencia enunciaba la igualdad semántica de mismos juicios de tipado y en nuestro ejemplo estamos considerando juicios de tipado que no son iguales debido a los diferentes contextos. Esto se debe por un lado a que tiene sentido hablar de coherencia de juicios de tipado cuando consideramos juicios para frases cerradas ya que en ese caso siempre consideramos el contexto vacío, pero por otro lado para realizar la prueba de coherencia necesitamos una propiedad mas general que considere la igualdad de juicios de tipado con diferentes contextos.

Otra cosa que surge de analizar este ejemplo es cómo saber cuantas derivaciones posibles existen para una determinada frase y tipo, acabamos de ver que incluso para el identificador con un tipo simple existen muchas opciones. Recordando que nos interesa seriamente saber la cantidad de derivaciones que pueda tener una frase para poder asegurar que todas tienen exactamente el mismo significado semántico. Para hacer frente a esto es que usamos el lema de inversión, pero antes de enunciarlo hagamos una ultima observación. Adelantando que vamos a utilizar inducción estructural en la derivación del

juicio de tipado para probar coherencia, notar que podemos agrupar tres de las derivaciones en un solo paso que es aplicar Subsunción como ultima regla para obtener el juicio de tipado final. En conclusión nuestro lema de inversión nos dirá que dado un juicio de tipado cualquiera la ultima regla aplicada para concluirlo fue, o bien la regla respectiva a su frase o bien Subsunción.

Lema 1 (De inversión). *Sea* $\pi \vdash e : \theta$ *un juicio de tipado valido cualquiera, entonces sucedió alguna de la siguientes:*

- *Si* e *es constante, la ultima regla aplicada fue (Constantes) o (Subsunción)*
- *Si* e *es identificador, la ultima regla aplicada fue (Identificador) o (Subsunción)*
- *Si* $e = \odot e'$*, la ultima regla aplicada fue* ($\odot$) *o (Subsunción)*
- *Si* $e = e' \circledcirc e''$*, la ultima regla aplicada fue* ($\circledcirc$) *o (Subsunción)*
- *Si* $e = \textbf{\textit{if}}\ b\ \textbf{\textit{then}}\ e'\ \textbf{\textit{else}}\ e''$*, la ultima regla aplicada fue (Expresión condicional) o (Subsunción)*
- *Si* $e = e'e''$*, la ultima regla aplicada fue (Aplicación) o (Subsunción)*
- *Si* $e = \lambda\iota_\theta.e'$*, la ultima regla aplicada fue (Abstracción lambda) o (Subsunción)*
- *Si* $e = \textbf{\textit{rec}}\ e'$*, la ultima regla aplicada fue (Operador de punto fijo) o (Subsunción)*

Para probar coherencia vamos a enunciar y probar un lema mas general, pero antes vamos a definir informalmente una función que será de utilidad para agilizar la prueba y definir algunas propiedades. Esta función estricta la nombraremos $\mathcal{J}_\theta^{\theta'}$ tal que va de $[\![\theta]\!]$ en $[\![\theta']\!]$ y los posibles casos se defininen así

$$\mathcal{J}_{\mathbf{bool}}^{\mathbf{int}} = \lambda b.\ \text{if } b \text{ then } 0 \text{ else } 1$$

$$\mathcal{J}_{\mathbf{int}}^{\mathbf{real}} = \mathcal{J} \text{ (la inyección en los reales)}$$

$$\mathcal{J}_{\mathbf{bool}}^{\mathbf{real}} = \mathcal{J}_{\mathbf{int}}^{\mathbf{real}} \circ \mathcal{J}_{\mathbf{bool}}^{\mathbf{int}}$$

Proposición 2. *Sea* $\mathcal{J}_\theta^{\theta'}$*, dados* z, z' *en* $[\![\theta]\!]$*, dos operadores unarios* $\odot : [\![\theta]\!] \to [\![\theta]\!]$ *y* $\odot' : [\![\theta']\!] \to [\![\theta']\!]$ *y dos operadores binarios* $\circledcirc : [\![\theta]\!] \to [\![\theta]\!] \to [\![\theta]\!]$ *y* $\circledcirc' : [\![\theta']\!] \to [\![\theta']\!] \to [\![\theta']\!]$ *cualesquiera entonces vale*

- $\mathcal{J}_\theta^{\theta'} \circ \iota_\uparrow = \iota_\uparrow \circ \mathcal{J}_\theta^{\theta'}$
- $\mathcal{J}_\theta^{\theta'} \circ \odot_\odot = \odot'_\odot \circ \mathcal{J}_\theta^{\theta'}$
- $\mathcal{J}_\theta^{\theta'}(z \circledcirc_\circledcirc z') = (\mathcal{J}_\theta^{\theta'} z) \circledcirc'_\circledcirc (\mathcal{J}_\theta^{\theta'} z')$

Lema 2. *Sean* e *una frase y* θ *un tipo cualesquiera y tal que tenemos juicios de tipado* $\pi \vdash e : \theta$ *y* $\pi' \vdash e : \theta$ *con* $\eta : [\![\pi]\!]$ *y* $\eta' : [\![\pi']\!]$ *entonces vale*

$$[\![\pi \vdash e : \theta]\!]\eta = [\![\pi' \vdash e : \theta]\!]\eta'$$

donde o bien pasa $\pi \leq \pi'$ *y* $[\![\pi \leq \pi']\!]\eta = \eta'$, *o bien se da* $\pi' \leq \pi$ *y* $[\![\pi' \leq \pi]\!]\eta' = \eta$

Demostración. En la prueba vamos a proceder por inducción en la estructura de la derivación de los juicios de tipado, como ya mencionamos antes, la idea es usar inversión.

- Casos base
  - Supongamos e es constante y tomemos un $\eta : [\![\pi]\!]$ cualquiera, luego por inversión tenemos

    Usando (Constante)

    $[\![\pi \vdash e : \theta]\!]\eta = \iota_{\uparrow} e$

    Usando (Subsunción)

    $[\![\pi \vdash e : \theta]\!]\eta = [\![\theta' \leq \theta]\!]([\![\pi \vdash e : \theta']\!]\eta) = \mathcal{J}^{\theta}_{\theta'}(\iota_{\uparrow} e) = \iota_{\uparrow}(\mathcal{J}^{\theta}_{\theta'} e)$

    Empecemos notando que por la definición que hicimos de la función $\mathcal{J}^{\theta'}_{\theta}$ podemos suponer sin que se nos escape ningún caso de que cuando usamos primero Subsunción, después $[\![\pi \vdash e : \theta']\!]\eta = \iota_{\uparrow} e$. Por otro lado restaría probar para este caso que $e : [\![\theta]\!]$ es igual a $\mathcal{J}^{\theta}_{\theta'} e : [\![\theta]\!]$, pero esto es directo de la definición de la función $\mathcal{J}^{\theta}_{\theta'}$.

  - Supongamos $e = \iota$ y tomemos $\eta : [\![\pi]\!]$, $\eta' : [\![\pi']\!]$ cualesquiera tal que $\pi' \leq \pi$ y $[\![\pi' \leq \pi]\!]\eta' = \eta$, notar que estamos suponiendo η's y π's distintos ya que no puede existir en un mismo contexto un identificador con dos tipos distintos, como antes usando inversión tenemos

    Usando (Identificador)

    $[\![\pi \vdash \iota : \theta]\!]\eta = \eta\iota = ([\![\pi' \leq \pi]\!]\eta')\iota = [\![\pi'\iota \leq \pi\iota]\!](\eta'\iota) = [\![\pi'\iota \leq \theta]\!](\eta'\iota)$

    Usando (Subsunción)

    $[\![\pi' \vdash \iota : \theta]\!]\eta' = [\![\theta' \leq \theta]\!]([\![\pi' \vdash \iota : \theta']\!]\eta')$

    Ahora bien, si $\iota : \theta' \in \pi'$ entonces la igualdad es directa ya que

    $[\![\theta' \leq \theta]\!]([\![\pi' \vdash \iota : \theta']\!]\eta') = [\![\pi'\iota \leq \theta]\!](\eta'\iota)$

por el contrario si pasa que $\iota : \theta'' \in \pi'$ donde $\theta' \neq \theta''$ entonces es porque la ultima regla en $[\![\pi' \vdash \iota : \theta']\!]\eta'$ no fue la del identificador sino la de Subsunción nuevamente. Supongamos entonces que $\iota : \theta_m \in \pi'$ tal que $\theta_m \leq \theta_{m-1} \leq \ldots \leq \theta_1 \leq \theta'$, notando que si tenemos esta cadena es porque en la derivación del juicio de tipado $\pi' \vdash \iota : \theta'$ nunca podemos aplicar la regla para tipar el identificador y es siempre Subsunción. Esto nos implica entonces

$$[\![\pi' \vdash \iota : \theta']\!]\eta' =$$
$$([\![\theta_1 \leq \theta']\!] \circ \ldots \circ [\![\theta_m \leq \theta_{m-1}]\!])([\![\pi' \vdash \iota : \theta_m]\!]\eta') =$$
$$([\![\theta_1 \leq \theta']\!] \circ \ldots \circ [\![\theta_m \leq \theta_{m-1}]\!])(\eta'\iota)$$

reescribiendo nuestras ecuaciones, tenemos

$$[\![\pi'\iota \leq \theta]\!](\eta'\iota) = [\![\theta_m \leq \theta]\!](\eta'\iota)$$

$$[\![\pi' \vdash \iota : \theta]\!]\eta' = ([\![\theta' \leq \theta]\!] \circ [\![\theta_1 \leq \theta']\!] \circ \ldots \circ [\![\theta_m \leq \theta_{m-1}]\!])(\eta'\iota)$$

luego restaría probar $[\![\theta_m \leq \theta]\!] = [\![\theta' \leq \theta]\!] \circ [\![\theta_1 \leq \theta']\!] \circ \ldots \circ [\![\theta_m \leq \theta_{m-1}]\!]$. Pero la prueba es directa de notar que $(\theta' \leq \theta_i) \circ (\theta_j \leq \theta') = \theta_j \leq \theta_i$, por la definición de la categoría de tipos Θ, además por ser $[\![_]\!]$ funtor, $[\![\theta' \leq \theta_i]\!] \circ [\![\theta_j \leq \theta']\!] = [\![(\theta' \leq \theta_i) \circ (\theta_j \leq \theta')]\!]$.

- Casos inductivos

  Supongamos un $e : \theta$ tal que vale $[\![\pi \vdash e : \theta]\!]\eta = [\![\pi' \vdash e : \theta]\!]\eta'$ con o bien $\pi \leq \pi'$ y $[\![\pi \leq \pi']\!]\eta = \eta'$ o bien $\pi' \leq \pi$ y $[\![\pi' \leq \pi]\!]\eta' = \eta$ y probemos que para un nuevo $e : \theta$ de mayor complejidad vale $[\![\pi \vdash e : \theta]\!]\eta = [\![\pi' \vdash e : \theta]\!]\eta'$ tal que $\pi \leq \pi'$ y $[\![\pi \leq \pi']\!]\eta = \eta'$ o tal que $\pi' \leq \pi$ y $[\![\pi' \leq \pi]\!]\eta' = \eta$

  Las pruebas de los distintos casos de inducción van a proceder de la siguiente manera, vamos a partir de suponer la ultima regla fue la del comando en si y probar que llegamos a que la ultima regla fue Subsunción o viceversa, además vamos a asumir que nuestros programas siempre terminan, para los casos en los que no es directo demostrar la igualdad.

  - Supongamos $e = \odot e'$, donde el caso para los operadores binarios es anágolo, luego

    $$[\![\pi \vdash \odot e' : \theta]\!]\eta = \odot_{\odot}([\![\pi \vdash e' : \theta]\!]\eta)$$

    ahora bien, por inversión tenemos que la ultima regla utilizada en la derivación del juicio de tipado de $\pi \vdash e' : \theta$ puede haber sido Subsunción o la regla respectiva a e, es decir

$$\text{D:}\ \frac{\dfrac{\vdots}{\pi \vdash e' : \theta'} \qquad \theta' \leq \theta}{\pi \vdash e' : \theta} \qquad\qquad \text{D':}\ \frac{\vdots}{\pi \vdash e' : \theta}$$

además por hipótesis inductiva tenemos que $[\![D]\!] = [\![D']\!]$ y por lo tanto podemos resolver $[\![\pi \vdash e' : \theta]\!]\eta$ utilizando indistintamente cualquier derivación, en particular tomamos D

$$\odot_{\odot}([\![\pi \vdash e' : \theta]\!]\eta) = \odot_{\odot}([\![\theta' \leq \theta]\!][\![\pi \vdash e' : \theta']\!]\eta) = \odot_{\odot}(\mathcal{J}^{\theta}_{\theta'}[\![\pi \vdash e' : \theta']\!]\eta)$$

usando la propiedad sobre la inyección $\mathcal{J}^{\theta}_{\theta'}$ tenemos

$$\odot_{\odot}(\mathcal{J}^{\theta}_{\theta'}[\![\pi \vdash e' : \theta']\!]\eta) = \mathcal{J}^{\theta}_{\theta'}(\odot_{\odot}[\![\pi \vdash e' : \theta']\!]\eta) = [\![\theta' \leq \theta]\!](\odot_{\odot}[\![\pi \vdash e' : \theta']\!]\eta) = [\![\theta' \leq \theta]\!]([\![\pi \vdash \odot e' : \theta']\!]\eta) = [\![\pi \vdash \odot e' : \theta]\!]\eta$$

Y con esto hemos finalizado la prueba de este caso, repasando, partimos de suponer la ultima regla usada fue $\odot$ y llegamos a que la ultima regla usada fue Subsunción.

- Supongamos $e = \textbf{if}\ b\ \textbf{then}\ e'\ \textbf{else}\ e''$ y $[\![\pi \vdash b : \textbf{bool}]\!]\eta = \iota_{\uparrow}\ \mathit{true}$

$$\begin{aligned}
&[\![\pi \vdash \textbf{if}\ b\ \textbf{then}\ e'\ \textbf{else}\ e'' : \theta]\!]\eta = \\
&\qquad (\lambda b.\ \mathtt{if}\ b\ \mathtt{then}\ [\![\pi \vdash e' : \theta]\!]\eta \\
&\qquad\qquad \mathtt{else}\ [\![\pi \vdash e'' : \theta]\!]\eta)_{\perp\!\!\!\perp}([\![\pi \vdash b : \textbf{bool}]\!]\eta) = \\
&[\![\pi \vdash e' : \theta]\!]\eta
\end{aligned}$$

usando inversión e hipótesis inductiva como en el caso del operador unario, tenemos que

$$\begin{aligned}
&[\![\pi \vdash e' : \theta]\!]\eta = [\![\theta' \leq \theta]\!]([\![\pi \vdash e' : \theta']\!]\eta) = \\
&[\![\theta' \leq \theta]\!](\ (\lambda b.\ \mathtt{if}\ b\ \mathtt{then}\ [\![\pi \vdash e' : \theta']\!]\eta \\
&\qquad\qquad \mathtt{else}\ [\![\pi \vdash e'' : \theta']\!]\eta)_{\perp\!\!\!\perp}([\![\pi \vdash b : \textbf{bool}]\!]\eta)) = \\
&[\![\theta' \leq \theta]\!]([\![\pi \vdash \textbf{if}\ b\ \textbf{then}\ e'\ \textbf{else}\ e'' : \theta']\!]\eta) = \\
&[\![\pi \vdash \textbf{if}\ b\ \textbf{then}\ e'\ \textbf{else}\ e'' : \theta]\!]\eta
\end{aligned}$$

Análogo si suponemos $[\![\pi \vdash b : \textbf{bool}]\!]\eta = \iota_{\uparrow}\ \mathit{false}$.

- Supongamos $e = e'e''$, vamos a proceder de manera similar a como lo hicimos en el caso del operador unario, luego usando la definición de la aplicación tenemos

$$[\![\pi \vdash e'e'' : \theta]\!]\eta = [\![\pi \vdash e' : \widehat{\theta} \to \theta]\!]\eta([\![\pi \vdash e'' : \widehat{\theta}]\!]\eta)$$

ahora bien, usando inversión en ambos juicios de tipado tenemos

$$\dfrac{D_{e'}:\ \dfrac{\vdots}{\pi \vdash e' : \widehat{\theta}' \to \theta'} \qquad \widehat{\theta}' \to \theta' \leq \widehat{\theta} \to \theta}{\pi \vdash e' : \widehat{\theta} \to \theta}$$

$$D'_{e'}: \quad \cfrac{\vdots}{\pi \vdash e' : \widehat{\theta} \to \theta}$$

$$D_{e''}: \quad \cfrac{\cfrac{\vdots}{\pi \vdash e'' : \widehat{\widehat{\theta}}} \qquad \widehat{\widehat{\theta}} \leq \widehat{\theta}}{\pi \vdash e'' : \widehat{\theta}}$$

$$D'_{e''}: \quad \cfrac{\vdots}{\pi \vdash e'' : \widehat{\theta}}$$

luego por hipótesis inductiva en cada juicio de tipado tenemos, $[\![D_{e'}]\!] = [\![D'_{e'}]\!]$ y $[\![D_{e''}]\!] = [\![D'_{e''}]\!]$ y por lo tanto podemos usar para resolver $[\![\pi \vdash e' : \widehat{\theta} \to \theta]\!]$ y $[\![\pi \vdash e'' : \widehat{\theta}]\!]$ cualquiera de las derivaciones según corresponda. En particular tomamos $D_{e'}$ y $D_{e''}$, desarrollemos cada ecuación por separado

$[\![\pi \vdash e' : \widehat{\theta} \to \theta]\!]\eta = [\![\widehat{\theta}' \to \theta' \leq \widehat{\theta} \to \theta]\!]([\![\pi \vdash e' : \widehat{\theta}' \to \theta']\!]\eta) =$
$[\![\theta' \leq \theta]\!] \circ ([\![\pi \vdash e' : \widehat{\theta}' \to \theta']\!]\eta) \circ [\![\widehat{\theta} \leq \widehat{\theta}']\!]$

$[\![\pi \vdash e'' : \widehat{\theta}]\!]\eta = [\![\widehat{\widehat{\theta}} \leq \widehat{\theta}]\!]([\![\pi \vdash e'' : \widehat{\widehat{\theta}}]\!]\eta)$

reemplazando ahora en la ecuación principal obtenemos

$[\![\pi \vdash e' : \widehat{\theta} \to \theta]\!]\eta([\![\pi \vdash e'' : \widehat{\theta}]\!]\eta) =$
$([\![\theta' \leq \theta]\!] \circ ([\![\pi \vdash e' : \widehat{\theta}' \to \theta']\!]\eta) \circ [\![\widehat{\theta} \leq \widehat{\theta}']\!])\ ([\![\widehat{\widehat{\theta}} \leq \widehat{\theta}]\!]([\![\pi \vdash e'' : \widehat{\widehat{\theta}}]\!]\eta))$
$=$
$([\![\theta' \leq \theta]\!] \circ ([\![\pi \vdash e' : \widehat{\theta}' \to \theta']\!]\eta) \circ [\![\widehat{\theta} \leq \widehat{\theta}']\!] \circ [\![\widehat{\widehat{\theta}} \leq \widehat{\theta}]\!])\ ([\![\pi \vdash e'' : \widehat{\widehat{\theta}}]\!]\eta)$

ahora utilizando la composición de morfismos y que $[\![_]\!]$ es funtor, tenemos que $[\![\widehat{\theta} \leq \widehat{\theta}']\!] \circ [\![\widehat{\widehat{\theta}} \leq \widehat{\theta}]\!] = [\![\widehat{\widehat{\theta}} \leq \widehat{\theta}']\!]$, luego

$([\![\theta' \leq \theta]\!] \circ ([\![\pi \vdash e' : \widehat{\theta}' \to \theta']\!]\eta) \circ [\![\widehat{\theta} \leq \widehat{\theta}']\!] \circ [\![\widehat{\widehat{\theta}} \leq \widehat{\theta}]\!])\ ([\![\pi \vdash e'' : \widehat{\widehat{\theta}}]\!]\eta)$
$=$
$([\![\theta' \leq \theta]\!] \circ ([\![\pi \vdash e' : \widehat{\theta}' \to \theta']\!]\eta) \circ [\![\widehat{\widehat{\theta}} \leq \widehat{\theta}']\!])\ ([\![\pi \vdash e'' : \widehat{\widehat{\theta}}]\!]\eta) =$
$([\![\theta' \leq \theta]\!] \circ ([\![\pi \vdash e' : \widehat{\theta}' \to \theta']\!]\eta))([\![\widehat{\widehat{\theta}} \leq \widehat{\theta}']\!]\ ([\![\pi \vdash e'' : \widehat{\widehat{\theta}}]\!]\eta)) =$
$([\![\theta' \leq \theta]\!] \circ [\![\pi \vdash e' : \widehat{\theta}' \to \theta']\!]\eta)\ ([\![\pi \vdash e'' : \widehat{\theta}']\!]\eta) =$
$[\![\theta' \leq \theta]\!]([\![\pi \vdash e' : \widehat{\theta}' \to \theta']\!]\eta\ ([\![\pi \vdash e'' : \widehat{\theta}']\!]\eta)) =$
$[\![\theta' \leq \theta]\!]([\![\pi \vdash e'e'' : \theta']\!]\eta) =$
$[\![\pi \vdash e'e'' : \theta]\!]\eta$

- Supongamos $e = \lambda\iota_\theta.e'$ y un $z : [\![\theta]\!]$ cualquiera, luego

  $$
  \begin{aligned}
  &[\![\pi \vdash \lambda\iota_\theta.e' : \theta \to \theta']\!]\eta\, z = \\
  &([\![\widehat{\theta} \to \widehat{\theta}' \leq \theta \to \theta']\!]([\![\pi \vdash \lambda\iota_{\widehat{\theta}}.e' : \widehat{\theta} \to \widehat{\theta}']\!]\eta))z = \\
  &([\![\widehat{\theta}' \leq \theta']\!] \circ ([\![\pi, \iota : \widehat{\theta} \vdash e' : \widehat{\theta}']\!] \circ (\mathrm{ext}_{\iota,\widehat{\theta}}\eta)) \circ [\![\theta \leq \widehat{\theta}]\!])z = \\
  &[\![\pi, \iota : \widehat{\theta} \vdash e' : \theta']\!](\mathrm{ext}_{\iota,\widehat{\theta}}\, \eta\, ([\![\theta \leq \widehat{\theta}]\!]z)) = \\
  &[\![\pi, \iota : \widehat{\theta} \vdash e' : \theta']\!][\, \eta \mid \iota : [\![\theta \leq \widehat{\theta}]\!]z\,]
  \end{aligned}
  $$

  Esta vez empezamos suponiendo que la ultima regla fue Subsunción, desarrollamos utilizando la definición de la ecuación para el subtipado del tipo flecha y resolvemos la composición. Todo esto deja listo para aplicar hipótesis inductiva

  $$
  \begin{aligned}
  &[\![\pi, \iota : \widehat{\theta} \vdash e' : \theta']\!][\, \eta \mid \iota : [\![\theta \leq \widehat{\theta}]\!]z\,] = \\
  &[\![\pi, \iota : \theta \vdash e' : \theta']\!][\, \eta \mid \iota : z\,] = \\
  &([\![\pi, \iota : \theta \vdash e' : \theta']\!] \circ (\mathrm{ext}_{\iota,\theta}\eta))z = \\
  &[\![\pi \vdash \lambda\iota_\theta.e' : \theta \to \theta']\!]\eta\, z
  \end{aligned}
  $$

  Lo que nos restaría probar entonces es que

  $$[\![\pi, \iota : \theta \leq \pi, \iota : \widehat{\theta}]\!][\, \eta \mid \iota : z\,] = [\, \eta \mid \iota : [\![\theta \leq \widehat{\theta}]\!]z\,]$$

  pero esto es directo usando la definición, supongamos un ι', luego

  - Si $\iota' \neq \iota$,

    $$
    \begin{aligned}
    &([\![\pi, \iota : \theta \leq \pi, \iota : \widehat{\theta}]\!][\, \eta \mid \iota : z\,])\iota' = \\
    &[\![\pi\iota' \leq \pi\iota']\!](\eta\iota') = \eta\iota' = \\
    &[\, \eta \mid \iota : [\![\theta \leq \widehat{\theta}]\!]z\,]\iota'
    \end{aligned}
    $$

  - Si $\iota' = \iota$,

    $$
    \begin{aligned}
    &([\![\pi, \iota : \theta \leq \pi, \iota : \widehat{\theta}]\!][\, \eta \mid \iota : z\,])\iota' = \\
    &[\![(\pi, \iota : \theta)\iota' \leq (\pi, \iota : \widehat{\theta})\iota']\!]([\, \eta \mid \iota : z\,]\iota') = \\
    &[\![\theta \leq \widehat{\theta}]\!]\, z = \\
    &[\, \eta \mid \iota : [\![\theta \leq \widehat{\theta}]\!]z\,]\iota'
    \end{aligned}
    $$

- Supongamos $e = \mathbf{rec}\ e'$, luego

  $$
  \begin{aligned}
  &[\![\pi \vdash \mathbf{rec}\ e' : \theta]\!]\eta = [\![\theta' \leq \theta]\!]([\![\pi \vdash \mathbf{rec}\ e' : \theta']\!]\eta) = \\
  &[\![\theta' \leq \theta]\!](\mathbf{Y}_\theta([\![\pi \vdash e' : \theta' \to \theta']\!]\eta)) = \\
  &[\![\theta' \leq \theta]\!](\bigsqcup_{i=0}^{\infty}([\![\pi \vdash e' : \theta' \to \theta']\!]\eta)^i \bot_{\theta'}) =
  \end{aligned}
  $$

  ahora usando que $[\![\theta' \leq \theta]\!]$ es continua y como se uso en alguna otra ocasión, que $[\![\theta \leq \theta']\!]\bot_\theta = \bot_{\theta'}$, tenemos

$$
\begin{aligned}
&[\![\theta' \leq \theta]\!](\bigsqcup_{i=0}^{\infty}([\![\pi \vdash e' : \theta' \to \theta']\!]\eta)^i \bot_{\theta'}) = \\
&\bigsqcup_{i=0}^{\infty}(([\![\theta' \leq \theta]\!] \circ ([\![\pi \vdash e' : \theta' \to \theta']\!]\eta)^i)\bot_{\theta'}) = \\
&\bigsqcup_{i=0}^{\infty}(([\![\theta' \leq \theta]\!] \circ ([\![\pi \vdash e' : \theta' \to \theta']\!]\eta)^i \circ [\![\theta \leq \theta']\!])\bot_{\theta}) = \\
&\bigsqcup_{i=0}^{\infty}(([\![\theta' \to \theta' \leq \theta \to \theta]\!]([\![\pi \vdash e' : \theta' \to \theta']\!]\eta)^i)\bot_{\theta}) = \\
&\bigsqcup_{i=0}^{\infty}(([\![\pi \vdash e' : \theta \to \theta]\!]\eta)^i \bot_{\theta}) = \mathbf{Y}_{\theta}([\![\pi \vdash e' : \theta \to \theta]\!]\eta) = \\
&[\![\pi \vdash \mathbf{rec}\ e' : \theta]\!]\eta
\end{aligned}
$$

□

Si ahora fijamos los $\pi = \pi'$ en nuestro lema, obtenemos la propiedad de coherencia. Es decir que dadas D y D′ derivaciones del juicio de tipado $\pi \vdash e : \theta$ entonces $[\![D]\!] = [\![D']\!]$.

Algo a comentar es que el enunciado de nuestro lema que nos permite probar coherencia tiene una interesante similitud con la proposición en [3, Prop 4] que se utiliza para probar coherencia. Ambas son más generales que la propiedad en sí de coherencia, en particular existe una generalización sobre los contextos. Pero además la proposición 4 considera una generalización sobre el tipo de los juicios de tipado, esto debido a que el sistema de tipos que se considera contiene intersección.

3.4. Implementación en Idris

La implementación del lenguaje se encuentra en:
https://github.com/alexgadea/thesis/tree/master/Prototypes/Idris/LambdaLeq

Sintaxis del subtipado

```
data (<~) : PhraseType -> PhraseType -> Type where
    IntExpToRealExp : IntExp  <~ RealExp
    BoolExpToIntExp : BoolExp <~ IntExp

    Reflx : (t:PhraseType) -> t <~ t
    Trans : {t:PhraseType} -> {t':PhraseType} -> {t'':PhraseType} ->
            t <~ t' -> t' <~ t'' -> t <~ t''

    SubsFun : {t0:PhraseType} -> {t0':PhraseType} ->
              {t1:PhraseType} -> {t1':PhraseType} ->
              t0 <~ t0' -> t1 <~ t1' -> (t0' :-> t1) <~ (t0 :-> t1')
```

Semántica para el subtipado

```
evalLeq : {t:PType} -> {t':PType} -> t <~ t' -> evalTy t -> evalTy t'
evalLeq IntExpToRealExp    = prim__intToFloat
evalLeq BoolExpToIntExp    = prim__boolToInt
evalLeq {t'=t} (Reflx t)   = id
evalLeq (Trans leq leq')   = evalLeq leq' . evalLeq leq
evalLeq (SubsFun leq leq') = \f => evalLeq leq' . f . evalLeq leq
```

Semántica para el lenguaje $\lambda^{\leqslant}$

```
using (Pi:Ctx, Theta:PType, Theta':PType)
    data TypeJugdmnt : Ctx -> PhraseType -> Type where
        I     : (i:Identifier) -> InCtx Pi i -> TypeJugdmnt Pi Theta
        CInt  : Int   -> TypeJugdmnt Pi IntExp
        CBool : Bool  -> TypeJugdmnt Pi BoolExp
        CReal : Float -> TypeJugdmnt Pi RealExp

        Lam   : (i:Identifier) -> (pt:PhraseType) -> (fi:Fresh Pi i) ->
                TypeJugdmnt (Prepend Pi i pt fi) Theta' ->
                TypeJugdmnt Pi (pt :-> Theta')
        App   : TypeJugdmnt Pi (Theta :-> Theta') -> TypeJugdmnt Pi Theta ->
                TypeJugdmnt Pi Theta'
        Rec   : TypeJugdmnt Pi (Theta :-> Theta) -> TypeJugdmnt Pi Theta

        If    : TypeJugdmnt Pi BoolExp -> TypeJugdmnt Pi Theta ->
                TypeJugdmnt Pi Theta -> TypeJugdmnt Pi Theta
        BinOp : (evalTy a -> evalTy b -> evalTy c) ->
                TypeJugdmnt Pi a -> TypeJugdmnt Pi b -> TypeJugdmnt Pi c
        UnOp  : (evalTy a -> evalTy b) -> TypeJugdmnt Pi a -> TypeJugdmnt Pi b

        Subs    : Theta <~ Theta' -> TypeJugdmnt Pi Theta -> TypeJugdmnt Pi Theta'

eval : {Pi:Ctx} -> {Theta:PType} -> TypeJugdmnt Pi Theta -> evalCtx Pi -> evalTy Theta
eval (Subs leq p) eta = evalLeq leq $ eval p eta
eval {Pi=p} {Theta=pt} (I i iIn) eta = search p i pt iIn eta
eval (CInt x)    eta = x
eval (CBool x)    eta = x
eval (CReal x)    eta = x
eval {Pi=p} (Lam i pt fi b) eta = \z => eval b (update p eta i pt z fi)
eval (App e e')   eta = (eval e eta) (eval e' eta)
eval (Rec e) eta = fix (eval e eta)
eval (If b e e')  eta = if eval b eta then eval e eta else eval e' eta
eval (BinOp op x y) eta = op (eval x eta) (eval y eta)
eval (UnOp op x) eta = op (eval x eta)
```

4 Lenguaje Algol-like

Al final del capítulo anterior terminamos con el estudio del lenguaje $\lambda^{\leqslant}$, este era un lenguaje funcional con subtipado. En este capítulo vamos a tomar a $\lambda^{\leqslant}$ y vamos a agregar aspectos imperativos, a este nuevo lenguaje lo llamaremos λ^{like}.

Este nuevo lenguaje que vamos a estudiar pertenece a la clase de lenguaje Algol-like; combinan aspectos funcionales con imperativos y se basan en evaluación normal. Una propiedad importante de este tipo de lenguajes es la forma en la que se evalúa la aplicación de procedimientos en un programa, lo que sucede es que la evaluación del programa ocurre en dos etapas, en la primera se reduce el programa hasta que la aplicación del procedimiento desaparece quedando de esta manera la parte imperativa por evaluar, que es lo que se realiza en una segunda etapa.

A continuación presentamos los cinco principio que según [4] capturan la esencia de los lenguajes Algol-like.

1. Algol-like se obtiene de un lenguaje imperativo simple imponiendo un sistema para los procedimientos basado en el calculo lambda "fully typedz utilizando call-by-name. Donde con "fully typed"se refiere a que todos los errores de tipo deben ser errores sintácticos.

2. Existe dos clases de tipos: los *Data Types* que representarán los conjuntos de valores para expresiones y variables y los *Phrase Types* que representarán los conjuntos de valores para las frases e identificadores.

3. El orden de evaluación para las partes de una expresión y su conversión implícita debería estar indeterminada, pero el significado del lenguaje es independiente de la indeterminación.

4. La definición de procedimientos, recursión, expresiones condicionales pueden ser de cualquier *Phrase Types*.

5. El lenguaje contiene stack discipline y su definición debe hacer esta disciplina obvia.

El punto 3 se refiere a que tal como pasaba en $\lambda^{\leqslant}$, para una frase o expresión puede existir más de un juicio de tipado pero sin embargo la evaluación de los juicios de tipado resulta en un mismo valor semántico.

Sobre el ultimo punto hagamos un pequeño paréntesis para comentar a que nos referimos con stack discipline: esta tiene que ver con el compactado del ambiente y el almacén al terminar de ejecutar un comando con la finalidad de perder toda referencia a las variables introducidas por este mismo.
Además, vamos a presentar una variante de semántica para el lenguaje en la cual no tenemos stack discipline. Lo interesante es que lo vamos a lograr

cambiando realmente muy poco sobre las ecuaciones semánticas y los dominios originales que contemplan stack discipline.

4.1. Sintaxis de λ^{like}

Como ya mencionamos λ^{like} sera una extensión de $\lambda^{\leqslant}$, agregando las construcciones para un lenguaje imperativo simple. Pero antes de hacer esta extensión hay algo a acomodar en cuantos a los lenguajes de donde se extiende λ^{like}: si consideramos el segundo punto de los principios de algol, el sistema de tipos de $\lambda^{\leqslant}$ es demasiado simple, en consecuencia debemos extenderlo.

Comencemos entonces acomodando nuestro sistema de tipos, empecemos tomando los tipos **bool**, **int** y **real** de $\langle Type \rangle$ y separándolos por un lado, para introducir la gramática de los *Data Types*.

$\langle Data\ \ Types \rangle$ ::= **bool** | **int** | **real**

Por otro lado en $\langle Type \rangle$ tenemos el operador funcional $\rightarrow$, este vamos a moverlo a nuestra otra clase de tipos *Phrase Types*, esto recordado que nuestros procedimientos según el punto 4 deben tener este tipo. Vamos a agregar las construcciones de tipos que representan los valores para los comandos, aceptadores, variables y expresiones. A este ultimo tipo se lo puede considerar como el nuevo tipo que representa a los tipos como los teníamos antes, es decir, si una frase antes tenía tipo **int** ahora tendrá tipo **intexp**(resión). Dicho lo anterior no hay duda sobre la utilidad de δ**exp** como tipo, ahora repasemos de los demás tipos que vamos a introducir, el tipo de los comandos no será otra cosa que el representante de las frases de la parte imperativa de nuestro lenguaje, el tipo de las variables encapsulará las dos posibles utilizaciones de un identificador de variable, es decir, ya sea como valor o como almacenamiento, para este primero es que tenemos el tipo de las expresiones y para el segundo es donde aparece el tipo de los aceptadores, este básicamente se encarga de representar a un identificador de variables como un almacén de valores; es decir, si una frase tiene tipo δ**acc** puede aparecer en el lado izquierdo de una asignación pero no en el lado derecho.

$\langle Phrase\ \ Types \rangle$ ::= **comm**
| **boolacc** | **intacc** | **realacc**
| **boolexp** | **intexp** | **realexp**
| **boolvar** | **intvar** | **realvar**
| $\langle Phrase\ \ Types \rangle \rightarrow \langle Phrase\ \ Types \rangle$

Ahora que hemos actualizado el lenguaje de tipos necesario para λ^{like}, presentemos la gramática de los programas; como mencionamos anteriormente este lenguaje va a ser una extensión de $\lambda^{\leqslant}$, agregando las construcciones usuales de un lenguaje imperativo simple. Estas construcciones serán los comandos para la declaración de variables (**new**δ**var**), asignación(**:=**), comando neutro (**skip**), concatenación de comandos (**;**) y un comando para iteraciones (**while**).

$$
\begin{aligned}
\langle \mathrm{Phrase} \rangle ::= {} & \langle \mathrm{PBool} \rangle \mid \langle \mathrm{PInt} \rangle \mid \langle \mathrm{PReal} \rangle \\
& \mid \odot\ \langle \mathrm{Phrase} \rangle \mid \langle \mathrm{Phrase} \rangle \circledcirc \langle \mathrm{Phrase} \rangle \\
& \mid \mathbf{if}\ \langle \mathrm{Phrase} \rangle\ \mathbf{then}\ \langle \mathrm{Phrase} \rangle\ \mathbf{else}\ \langle \mathrm{Phrase} \rangle \\
& \mid \langle \mathrm{Id} \rangle \\
& \mid \langle \mathrm{Phrase} \rangle \langle \mathrm{Phrase} \rangle \\
& \mid \lambda\ \langle \mathrm{Id} \rangle_\theta\ .\ \langle \mathrm{Phrase} \rangle \\
& \mid \mathbf{rec} \langle \mathrm{Phrase} \rangle \\
& \mid \mathbf{new}\ \delta\mathbf{var}\ \langle \mathrm{Id} \rangle \mathbf{:=} \langle \mathrm{Phrase} \rangle\ \mathbf{in}\ \langle \mathrm{Phrase} \rangle \\
& \mid \langle \mathrm{Phrase} \rangle \mathbf{:=} \langle \mathrm{Phrase} \rangle \mid \mathbf{skip} \mid \langle \mathrm{Phrase} \rangle\ \mathbf{;}\ \langle \mathrm{Phrase} \rangle \\
& \mid \mathbf{while}\ \langle \mathrm{Phrase} \rangle\ \mathbf{do}\ \langle \mathrm{Phrase} \rangle
\end{aligned}
$$

$$\langle \mathrm{PBool} \rangle ::= \mathrm{True} \mid \mathrm{False}$$

$$\langle \mathrm{PNat} \rangle ::= 0 \mid 1 \mid 2 \mid \ldots$$

$$\langle \mathrm{PInt} \rangle ::= \ldots \mid -2 \mid -1 \mid \langle \mathrm{PNat} \rangle$$

$$
\begin{aligned}
\langle \mathrm{PReal} \rangle ::= {} & \langle \mathrm{PNat} \rangle . \{\langle \mathrm{PNat} \rangle\}+ \\
& \mid -\langle \mathrm{PNat} \rangle . \{\langle \mathrm{PNat} \rangle\}+
\end{aligned}
$$

donde

$\langle Id \rangle$ es un conjunto numerable.

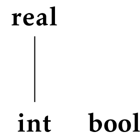
$\theta \in \langle Phrase\ \ Types \rangle$

$\odot \in \{-, \neg\}$ y

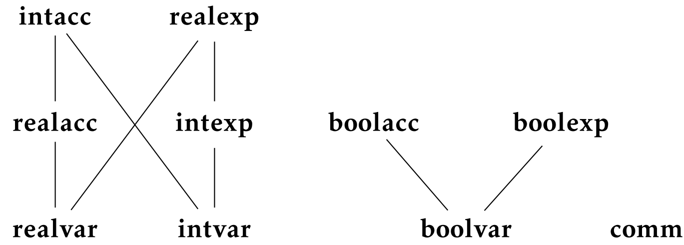
$\circledcirc \in \{+, -, *, /, \div, \mathbf{rem}, \wedge, \vee, \Rightarrow, \Leftrightarrow, =, \neq, <, >, \leq, \geq\}$

4.2. Reglas de inferencia para λ^{like}

La modificación de los tipos nos obliga a actualizar la relación de subtipado, ya que ahora, como mencionábamos antes, el tipo de una expresión entera no será **int** sino **intexp**. Pero antes de acómodar esto definamos una nueva relación de orden para los tipos de $\langle Data\ \ Types \rangle$; en esta nueva relación vamos a retirar **bool** $\leq$ **int**, cuyo fin en $\lambda^{\leqslant}$ era tener casos de transitividad entre los tipos que no funcionales, luego nos quedará la siguiente relación entre los tipos de datos.



Ahora utilizando este diagrama, que básicamente nos dice que **int** es subtipo de **real** y que no ocurre nada más interesante, presentamos la relación de los tipos de $\langle Phrase\ \ Types \rangle$,



Expliquemos un poco que estamos viendo: por un lado respetando la idea de que **int** es ahora **intexp** y **real** es **realexp**, entonces vamos a tener **intexp** $\leq$ **realexp**; esta relación se da vuelta si pensamos en el tipo de los aceptadores, ya que si podemos almacenaren un valor de tipo **realexp** entonces podemos almacenar un valor de tipo **intexp** y entonces tenemos **realacc** $\leq$ **intacc**. Finalmente si pensamos que una variable puede ser usada tanto como valor o contenedor, vamos a tener que, para cualquier $\delta \in \langle Data\ \ Types \rangle$, δ**var** $\leq$ δ**exp** y δ**var** $\leq$ δ**acc**.

Por último podemos prestar especial atención al supremo que aparece entre **realvar** e **intvar**, el cual podría determinar un tipo particular de variable que vista como aceptador tiene tipo **intacc** y vista como valor tiene tipo **realexp**. De hecho, si nuestro lenguaje soportara el operador de tipos intersección (&), podríamos definir a este tipo particular de variable como **intacc** & **realexp** y, más aun, podríamos definir a nuestro tipo actual δ**var** como δ**acc** & δ**exp**.

Dicho esto escribamos las nuevas reglas de tipado para estas relaciones nuevas y aprovechemos para actualizar nuestras metavaribles en relación a nuestros nuevos tipos.

$$\theta \in \langle Phrase\ \ Types \rangle \quad \text{y} \quad \delta \in \langle Data\ \ Types \rangle$$

`Ty Rule:` intexpTorealexp.

$$\frac{}{\textbf{intexp} \leq \textbf{realexp}}$$

`Ty Rule:` realaccTointacc.

$$\frac{}{\textbf{realacc} \leq \textbf{intacc}}$$

`Ty Rule:` δvarToδexp.

$$\frac{}{\delta\textbf{var} \leq \delta\textbf{exp}}$$

`Ty Rule:` δvarToδacc.

$$\frac{}{\delta\textbf{var} \leq \delta\textbf{acc}}$$

Además de estas reglas de inferencia para el subtipado, nos van a hacer falta actualizar las reglas de inferencia de $\lambda^{\leqslant}$ con los tipos correspondientes del lenguaje λ^{like} y luego faltará también agregar reglas para los juicios de tipado de las nuevas frases. Pero antes vamos a necesitar definir el nuevo lenguaje de los contextos, como ha venido ocurriendo, esto será actualizar los tipos viejos por los nuevos.

Definición 11. *Un contexto estará definido por la siguiente gramática,*

$\langle \text{Context} \rangle ::= \varnothing \mid \langle \text{Context} \rangle, \langle \text{Id} \rangle : \langle \textit{Phrase Types} \rangle$

tal que dado cualquier contexto $\iota_0 : \theta_0, \ldots, \iota_n : \theta_n$, *los identificadores* $\iota_0, \ldots, \iota_n$ *son todos distintos.*

Ahora sí, reescribamos las reglas de $\lambda^{\leqslant}$ con los nuevos tipos, básicamente lo que vamos a hacer es cambiar un tipo δ por $\delta\mathbf{exp}$,

`Ty Rule`: Constantes.

$$\frac{}{\pi \vdash b : \mathbf{boolexp}} \qquad \frac{}{\pi \vdash i : \mathbf{intexp}} \qquad \frac{}{\pi \vdash r : \mathbf{realexp}}$$

`Ty Rule`: Operadores básicos.

$$\frac{\pi \vdash e : \mathbf{boolexp}}{\pi \vdash \neg e : \mathbf{boolexp}} \qquad \frac{\pi \vdash e : \mathbf{intexp}}{\pi \vdash -e : \mathbf{intexp}} \qquad \frac{\pi \vdash e : \mathbf{realexp}}{\pi \vdash -e : \mathbf{realexp}}$$

$$\frac{\pi \vdash e : \mathbf{intexp} \qquad \pi \vdash e' : \mathbf{intexp}}{\pi \vdash e \otimes e' : \mathbf{intexp}} \ \otimes \in \{+, -, *, /\ , \mathbf{rem}\}$$

$$\frac{\pi \vdash e : \mathbf{realexp} \qquad \pi \vdash e' : \mathbf{realexp}}{\pi \vdash e \otimes e' : \mathbf{realexp}} \ \otimes \in \{+, -, *\}$$

$$\frac{\pi \vdash e : \mathbf{boolexp} \qquad \pi \vdash e' : \mathbf{boolexp}}{\pi \vdash e \oslash e' : \mathbf{boolexp}} \ \oslash \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$$

$$\frac{\pi \vdash e : \delta \qquad \pi \vdash e' : \delta}{\pi \vdash e \lessdot e' : \mathbf{boolexp}} \ \delta \in \{\mathbf{intexp}, \mathbf{realexp}\}, \lessdot \in \{<, >, \leq, \geq\}$$

$$\frac{\pi \vdash e : \delta\mathbf{exp} \qquad \pi \vdash e' : \delta\mathbf{exp}}{\pi \vdash e \ominus e' : \mathbf{boolexp}} \ \ominus \in \{=, \neq\}$$

`Ty Rule`: Aplicación.

$$\frac{\pi \vdash e : \theta \to \theta' \qquad \pi \vdash e' : \theta}{\pi \vdash ee' : \theta'}$$

`Ty Rule`: Operador de punto fijo.

$$\frac{\pi \vdash e : \theta \to \theta}{\pi \vdash \mathbf{rec}\ e : \theta}$$

`Ty Rule`: Identificador.

$$\frac{\iota : \theta \in \pi}{\pi \vdash \iota : \theta}$$

`Ty Rule`: Abstracción lambda.

$$\frac{\pi, \iota : \theta \vdash e : \theta'}{\pi \vdash \lambda\iota_\theta . e : \theta \to \theta'}$$

`Ty Rule`: Expresión condicional.

$$\frac{\pi \vdash b : \mathbf{boolexp} \qquad \pi \vdash e : \theta \qquad \pi \vdash e' : \theta}{\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \theta}$$

con $\theta \in \{\delta\mathbf{var}, \delta\mathbf{acc}, \delta\mathbf{exp}, \mathbf{comm}\}$.

Hasta aquí lo que hemos hecho es recompilar y actualizar las reglas de inferencia para la parte aplicativa del lenguaje, ahora definamos las regla de los comandos **skip**, **while** y **;** que no presentan ninguna particularidad que valga la pena comentar y luego sigamos con la regla **new** δ**var**. Pero antes notamos el hecho de que el juicio de tipado para la expresión condicional no es todo lo general que plantea el punto 4 del comienzo del capítulo, la respuesta para esta decisión se encuentra al momento de definir la ecuación semántica para este comando.

`Ty Rule`: **skip**.

$$\frac{}{\pi \vdash \mathbf{skip} : \mathbf{comm}}$$

`Ty Rule`: **while**.

$$\frac{\pi \vdash b : \mathbf{boolexp} \qquad \pi \vdash e : \mathbf{comm}}{\pi \vdash \mathbf{while}\ b\ \mathbf{do}\ e : \mathbf{comm}}$$

`Ty Rule`: Composición.

$$\frac{\pi \vdash e : \mathbf{comm} \qquad \pi \vdash e' : \mathbf{comm}}{\pi \vdash e ; e' : \mathbf{comm}}$$

En la regla de inferencia para la declaración de variable, ocurre algo muy similar a lo que ocurre con la regla de la abstracción lambda en cuanto a que se extiende el contexto, la principal diferencia será que solamente vamos a poder agregar el identificador con un phrase type particular, en tanto el tipo de un identificador podrá ser de la forma δ**var**.

`Ty Rule`: Declaración de variable.

$$\frac{\pi \vdash e : \delta\mathbf{exp} \qquad \pi, \iota : \delta\mathbf{var} \vdash e' : \mathbf{comm}}{\pi \vdash \mathbf{new}\ \delta\mathbf{var}\ \iota := e\ \mathbf{in}\ e' : \mathbf{comm}}$$

Antes mencinábamos que el tipo δ**var** encapsulaba de manera bien separada la noción de un identificador utilizado como contenedor de un valor o utilizado como un valor en si mismo. En la regla de la asignación se ve cómo para guardar un valor hace falta que la parte izquierda tenga tipo δ**acc**, es decir el tipo δ**var** visto como contenedor y la parte derecha tenga tipo δ**exp**, que en el caso de ser un identificador es el tipo δ**var** visto como valor, gracias al subtipado $\delta\mathbf{var} \leq \delta\mathbf{exp}$.

`Ty Rule`: Asignación.

$$\frac{\pi \vdash e : \delta\mathbf{acc} \qquad \pi \vdash e' : \delta\mathbf{exp}}{\pi \vdash e := e' : \mathbf{comm}}$$

4.3. Semántica para λ^{like}

A nivel sintáctico vimos que λ^{like} está compuesto por las frases de $\lambda^{\leqslant}$ más frases que representan la parte imperativa; de hecho al presentar las reglas de inferencia para las frases de λ^{like}, las reglas de la parte de $\lambda^{\leqslant}$ sufrieron un único cambio simple en cuanto a los nombre de los tipos y pudimos aprovechar la definición completa. Ahora vamos a dar la semántica de λ^{like} y no vamos a poder hacer algo similar con la semántica, es decir aprovechar las ecuaciones semánticas que definimos para $\lambda^{\leqslant}$. La razón principal es que el significado de nuestros tipos ya no será tan simple debido a que aparecen cuestiones relacionadas con el stack discipline.

Además para definir las dos semánticas que pretendemos para λ^{like}, con y sin stack discipline, vamos a tener definiciones extras, una para el juicio de tipado de la declaración de variables imperativas y otra para la categoría de estados de la parte imperativa. Lo interesante de notar sobre estas definiciones extras es que dadas todas las ecuaciones semánticas de nuestro lenguaje, variando una sola ecuación vamos a decidir sobre el tipo de implementación en relación al stack discipline.

Comencemos adaptando la categoría de tipos Θ, lo único por hacer será cambiar la clase de tipos de $\lambda^{\leqslant}$ por los de λ^{like}.

Definición 12. *La categoría de tipos, que nombraremos Θ, se define como sigue*

$\Theta_0 = \{\theta \mid \theta \in \langle \mathit{Phrase\ Types} \rangle\}$
$\Theta_1(\theta, \theta') = \{\theta \longrightarrow \theta' \mid \theta \leq \theta'\}$

Sobre la categoría de contextos no hay nada que cambiar, luego lo que nos queda por actualizar es la categoría **DC**; recordemos que **DC** en $\lambda^{\leqslant}$ la habíamos definido como la categoría concreta **Dom**. Ahora esto no es suficiente y vamos a necesitar un enfoque bastante distinto debido a que la separación que existe entre el calculo lambda y la parte imperativa nos induce una separacíon entre

los ambientes y los estados. Así, por ejemplo, sucede que $[\![\textbf{intexp}]\!]$ ya no será un $S_{\textbf{int}}$ que represente el conjunto de enteros, si no que pasará a ser una función de estados en $S_{\textbf{int}}$.

Pero antes de continuar, definamos que será un estado en nuestra semántica, para el cual tendremos dos versiones, una versión será un conjunto de valores pero con la particularidad de tener forma, donde aclarando un poco a que nos referimos con forma, es que si tomáramos un estado $(1, \mathsf{True})$ que pertenece a $\mathbb{Z} \times \mathbb{B}$, diremos que este estado tiene forma $\langle \mathbb{Z}, \mathbb{B} \rangle$. Las formas serán de bastante importancia ya que ayudarán a mantener estados bien tipado. Para la otra versión un estado será una función de identificadores en valores.

Definición 13. *Un estado será un elemento perteneciente a* $S_1 \times \ldots \times S_n$*, con* $1 \leq n$ *con* S_i *cualquiera de los conjuntos* S_{int}*,* S_{real} *o* S_{bool}*. Diremos además que este estado tiene forma* $\langle S_1, \ldots, S_n \rangle$

Además dados dos estados σ *y* σ'*, diremos que* σ *extiende a* σ' *cuando la forma de* σ *sea tramo inicial de la forma de* σ'*.*

Introduzcamos algunas funciones útiles que nos van a servir para operar sobre los estados y las formas de los estados. Vamos a usar $+\!\!+$ para definir la concatenación de estados y formas de estado, además para operar sobre los estados vamos a tener tres funciones básicas, head, tail y last.

Definición 14. *Sea* σ *un estado con forma* $\alpha +\!\!+ \alpha'$ *luego definimos* $\mathsf{head}_\alpha \sigma$ *y* $\mathsf{tail}_\alpha \sigma$ *como los únicos estados tales que* $\mathsf{head}_\alpha \sigma +\!\!+ \mathsf{tail}_\alpha \sigma = \sigma$*.*

Además dado un estado σ *con forma* $\alpha +\!\!+ \langle S \rangle$*, tenemos que* $\mathsf{last}_S \sigma$ *sera el ultimo valor contenido en el estado* σ*.*

Hasta aquí entonces hemos definido la primera versión de estados, esta será la que se corresponda con la semántica con stack discipline. Pasemos a definir la segunda versión, la cual se corresponderá entonces con la semántica sin stack discipline. Como ya mencionamos antes estos estados ahora serán funciones de identificadores en valores, en un sentido similar a cómo es un ambiente, la razón principal por la cual vamos a tener esta segunda versión se basa en que vamos a querer dejar de lado la idea de que los estados tengan forma y el problema que trae esto es que, justamente la forma es lo que nos permite actualizar o consultar los valores de nuestro estado, por lo tanto necesitamos otro tipo de estado con otro mecanismo para realizar esas acciones. Para resolver esto usamos una idea similar a la propuesta en [5] para el lenguaje Iswim.

Definición 15. *Un estado será un elemento perteneciente a* $\Sigma = \bigcup_{I \in \mathrm{IdF}} I \to S$*, donde* IdF *será un conjunto de subconjuntos finitos de* $\langle \mathrm{Id} \rangle$ *y* $S = S_{int} + S_{real} + S_{bool}$

Ahora con los estados definidos, vamos a dar las dos definiciones de la categoría de estados, una será la categoría en la que básicamente cada objeto será un conjunto de estados con la misma forma y una flecha entre dos objetos determinará como contraer la forma de un objeto y como transformar un estado

con cierta forma en un estado más grande. Y la otra categoría tendrá un solo objeto que agrupará a todos los estados sin importar la forma y cuya única flecha será la identidad.

Definición 16. *La categoría de estados, que nombraremos **SD**, se define como sigue,*

$$\boldsymbol{SD}_0 = \{C \mid C \text{ es el conjunto de todos los estados con determinada forma}\}$$
$$\boldsymbol{SD}_1(C, C') = \{C \xrightarrow{(h,s)} C' \mid C' \text{ extiende a } C\}$$

$$\begin{array}{lll} \textit{donde} & h : C' \to C & s : (C \to C_\perp) \to (C' \to C'_\perp) \\ & h = \mathtt{head}_C & s\ c\ \widehat{\sigma} = (\lambda\sigma.\ \sigma +\!\!+\ (\mathtt{tail}_{\overline{C}}\ \widehat{\sigma}))_\perp\ (c(h\ \widehat{\sigma})) \\ & C' = C +\!\!+\ \overline{C} & \end{array}$$

Notar que si tomamos un objeto C de **SD** podemos hablar de que este objeto tiene cierta forma, básicamente la forma de los estados que agrupa, luego podemos hablar de que dado otro objeto C', este extiende a C y esta extensión se determina de la misma manera que para los estados. Además aprovechamos que podemos hablar acerca de que C tiene forma, para hacer un abuso de notación en $\mathtt{head}$ y $\mathtt{tail}$.

Ahora definamos la segunda categoría de estados que mencionamos anteriormente, recordemos que esta será la categoría de estados en la semántica sin stack discipline.

Definición 17. *La categoría de estados, que nombraremos $\overline{\mathbf{SD}}$, se define como sigue,*

$$\overline{\boldsymbol{SD}}_0 = \{\ \Sigma\ \}$$

y cuya única flecha es $(1_h, 1_s) : \Sigma \to \Sigma$, *tal que*

$$\begin{array}{ll} 1_h : \Sigma \to \Sigma & 1_s : (\Sigma \to \Sigma_\perp) \to (\Sigma \to \Sigma_\perp) \\ 1_h\sigma = \sigma & 1_s\ c = c \end{array}$$

Una aclaración importante es que, salvo aclaración, cuando hablemos de la categoría **C** vamos a estar refiriéndonos tanto a **SD** como a $\overline{\mathbf{SD}}$. Dicho esto, estamos en condiciones de definir quien será **DC**, una idea inicial es pensar en la categoría funtorial $\mathbf{Dom}^{\mathbf{C}}$ como sugiere [5, Cap 19], pero parece existir un problema según se menciona en [2] que tiene que ver con que $\mathbf{Dom}^{\mathbf{C}}$ no es inmediatamente $\mathcal{CCC}$ a pesar de que **Dom** lo sea y habíamos mencionado que íbamos a querer que nuestra categoría **DC** sea $\mathcal{CCC}$. En esta ultima referencia entonces se plantea el uso de **PDom**, luego **DC** nos queda como la categoría funtorial $\mathbf{PDom}^{\mathbf{C}}$.

Para la definición de las ecuaciones semánticas primero vamos a tener que acomodar la semántica de nuestros tipos y contextos. Como ya mencionamos antes la representación que vamos a necesitar, por ejemplo, para el tipo **intexp** ya no será tan simple como antes. La nueva definición semántica de **intexp** será un funtor entre la categoría de estados **C** y la categoría funtorial **DC**.

Notar que como nuestra categoría **DC** es ahora una categoría funtorial, vamos a tener que la semántica de un objeto θ de Θ será un funtor,

$$[\![\theta]\!] : \mathbf{C} \to \mathbf{PDom}$$

tal que aplicado a un objeto C, de **C** es,

$$[\![\theta]\!]C : \mathbf{PDom}$$

y aplicado a una flecha $C \xrightarrow{(h,s)} C'$ es,

$$[\![\theta]\!](h, s) : [\![\theta]\!]C \to [\![\theta]\!]C'.$$

en general, el comportamiento de este funtor aplicado a una flecha será la traducción del estado actual con forma C a un estado extendido con forma C'. Además, la semántica de una flecha $\theta \xrightarrow{\leq} \theta'$ de Θ será una transformación natural, indexada por objetos de **C**,

$$[\![\theta \xrightarrow{\leq} \theta']\!]C : [\![\theta]\!]C \to [\![\theta']\!]C.$$

Definición 18. *Sea* $[\![_]\!] : \Theta \to \boldsymbol{DC}$ *un funtor, tal que*

$$[\![\delta\boldsymbol{exp}]\!]_0 C = C \to (S_\delta)_\perp$$
$$[\![\delta\boldsymbol{exp}]\!]_0 (h, s)\ e = e \circ h$$

$$[\![\boldsymbol{comm}]\!]_0 C = C \to C_\perp$$
$$[\![\boldsymbol{comm}]\!]_0 (h, s)\ c = s\ c$$

$$[\![\delta\boldsymbol{acc}]\!]_0 C = S_\delta \to [\![\boldsymbol{comm}]\!]C$$
$$[\![\delta\boldsymbol{acc}]\!]_0 (h, s)\ a = s \circ a$$

$$[\![\delta\boldsymbol{var}]\!]_0 C = [\![\delta\boldsymbol{acc}]\!]C \times [\![\delta\boldsymbol{exp}]\!]C$$
$$[\![\delta\boldsymbol{var}]\!]_0 (h, s)\ (a, e) = ([\![\delta\boldsymbol{acc}]\!](h, s)\ a, [\![\delta\boldsymbol{exp}]\!](h, s)\ e)$$

$$[\![\theta \to \theta']\!]_0 C = \boldsymbol{Nat}([\![\theta]\!](C + \!\!+\ _), [\![\theta']\!](C + \!\!+\ _))$$
$$[\![\theta \to \theta']\!]_0 (h, s)\ f\ \widehat{C} = f(\overline{C} + \!\!+\ \widehat{C})$$
$$\textit{donde } C + \!\!+\ \overline{C} = C'$$

$$[\![\boldsymbol{intexp} \leq \boldsymbol{realexp}]\!]_1 C\ e = \mathcal{J} \circ e \quad \textit{con } \mathcal{J} \textit{ la inyección de enteros en reales.}$$

$$[\![\boldsymbol{realacc} \leq \boldsymbol{intacc}]\!]_1 C\ a = a \circ \mathcal{J}$$

$$[\![\delta\boldsymbol{var} \leq \delta\boldsymbol{exp}]\!]_1 C\ (a, e) = e$$

$$[\![\delta\boldsymbol{var} \leq \delta\boldsymbol{acc}]\!]_1 C\ (a, e) = a$$

$$[\![\theta \leq \theta]\!]_1 C = 1_{[\![\theta]\!]C}$$

$$[\![\theta \leq \theta'']\!]_1 C = [\![\theta' \leq \theta'']\!]_1 C \circ [\![\theta \leq \theta']\!]_1 C$$

$$\begin{aligned}&[\![(\theta_0 \to \theta'_0) \leq (\theta_1 \to \theta'_1)]\!]_1 C\ f\ \widehat{C} \\ &\qquad = [\![\theta'_0 \leq \theta'_1]\!]_1 (C +\!\!+\ \widehat{C}) \circ (f\ \widehat{C}) \circ [\![\theta_1 \leq \theta_0]\!]_1 (C +\!\!+\ \widehat{C})\end{aligned}$$

Ahora que hemos definido las ecuaciones semánticas para nuestros tipos hagamos un repaso sobre las más relevantes de ellas. En determinados momentos vamos a hablar acerca de *alcance con forma*, a lo que nos vamos a referir es, no solo al contexto en el cual un identificador tiene sentido, sino además a la forma que debe tener un estado en ese contexto.

- Un elemento de $[\![\delta\mathbf{exp}]\!]C$, como ya hemos mencionado, será una función de un estado en un conjunto S_δ de valores; la motivación de esta definición es poder disponer del estado para obtener el valor de una variable imperativa cuando se la utiliza como expresión.

- $[\![\delta\mathbf{exp}]\!](h, s)\ e$, acá $e : C \to (S_\delta)_\perp$, luego hacemos la transformación componiendo con $h : C' \to C$, para obtener $e \circ h : C' \to (S_\delta)_\perp$.

- Un elemento de $[\![\mathbf{comm}]\!]C$, esta tal vez sea la más intuitiva, será una función de un estado con forma C en otro con la misma forma C, como en todos los casos contemplando la no terminación.

- $[\![\mathbf{comm}]\!](h, s)\ c$, parecido a como pasaba para $\delta\mathbf{exp}$, tenemos un $c : C \to C_\perp$ y tenemos que $s : (C \to C_\perp) \to (C' \to C'_\perp)$, luego $s\ c : C' \to C'_\perp$.

- Un elemento de $[\![\delta\mathbf{acc}]\!]C$, el tipo representa a la variable como contenedora de valores, luego la semántica será una función que toma un valor en S_δ, un estado y devuelve un nuevo estado con el valor de la variable modificado.

- $[\![\delta\mathbf{acc}]\!](h, s)\ a$, como mencionábamos antes, $a : S_\delta \to (C \to C_\perp)$ y $s : (C \to C_\perp) \to (C' \to C'_\perp)$, por lo tanto con la composición $s \circ a : S_\delta \to (C' \to C'_\perp)$ obtenemos la nueva función que actualiza valores de una variable en un estado extendido.

- $[\![\theta \to \theta']\!]C$, para interpretar esta ecuación podemos empezar cuestionando porque no podría estar definida de la siguiente manera,

  $$[\![\theta \to \theta']\!]C = [\![\theta]\!]C \to [\![\theta']\!]C$$

  y el problema está en que uno debe preguntarse bajo que alcance se realiza la aplicación: básicamente puede suceder que la aplicación suceda en un alcance que tiene nuevas declaraciones y como consecuencia de esto, el estado tenga diferente forma. La solución entonces se basa en obtener cuánto se extendió el estado y hacerlo de una manera uniforme.

- $[\![\theta \to \theta']\!](h, s)\ f$, pensemos que f puede aplicarse en un alcance con, por lo menos, forma C y queremos que pase a, por lo menos, poder aplicarse en un alcance con forma C', entonces ampliamos hasta por lo menos la forma C', con $\overline{C}$ y agregamos la posibilidad de seguir ampliando con $\widehat{C}$.

- $[\![(\theta_0 \to \theta_0') \leq (\theta_1 \to \theta_1')]\!]_1 C\ f\ \widehat{C}$, lo que sucede con esta ecuación no es muy diferente a lo que pasa con la definición hecha en $\lambda^{\leqslant}$, el único cuidado que hay que tener es el de extender la forma de C.

Pasemos ahora a definir la semántica de nuestros contextos, como hicimos con los tipos; empecemos analizando los tipos de las ecuaciones que vamos a definir. Sea π un objeto de Π luego tenemos que

$$[\![\pi]\!]C : \mathbf{PDom},$$

$$[\![\pi]\!](h, s) : [\![\pi]\!]C \to [\![\pi]\!]C' \text{ y}$$

$$[\![\pi \xrightarrow{\leq} \pi']\!]_1 C : [\![\pi]\!]C \to [\![\pi']\!]C$$

Definición 19. *Sea* $[\![_]\!] : \Pi \to \boldsymbol{DC}$ *un funtor, tal que*

$$[\![\pi]\!]_0 C = \prod_{\iota \in \mathrm{dom}\ \pi} [\![\pi\iota]\!]C$$

$$[\![\pi]\!]_0(h, s)\ \eta = \prod_{\iota \in \mathrm{dom}\ \pi} [\![\pi\iota]\!](h, s)(\eta\iota)$$

$$[\![\pi \leq \pi']\!]_1 C = \prod_{\iota \in \mathrm{dom}\ \pi} [\![\pi\iota \leq \pi'\iota]\!]C(\eta\iota)$$

De esta definición la ecuación que más nos va a interesar será $[\![\pi]\!]_0(h, s)\ \eta$, que nos servirá para acomodar un ambiente η a una determinada forma cuando lo extendemos agregando un valor; veremos además que esto pasa solamente para la abstracción lambda y la declaración de variable imperativa.

Ahora que hemos definido la semántica de contextos y tipos, podemos continuar con la semántica de los juicios de tipado de las frases. Como hicimos antes vamos a definir un funtor, solo que este tomará un juicio de tipado $\pi \vdash e : \theta$ y devolverá una transformación natural del funtor $[\![\pi]\!]$ en el funtor $[\![\theta]\!]$ indexada por objetos de **C**. Nota: vamos a escribir el objeto que indexa no como subíndice sino como un argumento mas.

Además vamos a escribir las ecuaciones utilizando las funciones $(_)_{\odot}$ y $(_)_{\circledcirc}$ que definimos en $\lambda^{\rightarrow}$.

`Denotal Sem`: Constantes.

$$[\![\pi \vdash b : \mathbf{boolexp}]\!]C\ \eta\ \sigma = \iota_{\uparrow}\ b$$

$$[\![\pi \vdash i : \mathbf{intexp}]\!]C\ \eta\ \sigma = \iota_{\uparrow}\ i$$

$$[\![\pi \vdash r : \mathbf{realexp}]\!]C\ \eta\ \sigma = \iota_{\uparrow}\ r$$

`Denotal Sem`: Operadores unarios.

$$[\![\pi \vdash \neg b : \mathbf{boolexp}]\!]C = \neg_{\odot} \circ [\![\pi \vdash b : \mathbf{boolexp}]\!]C$$

$$[\![\pi \vdash -i : \mathbf{intexp}]\!]C = -_{\odot} \circ [\![\pi \vdash i : \mathbf{intexp}]\!]C$$

$$[\![\pi \vdash -r : \mathbf{realexp}]\!]C = -_{\odot} \circ [\![\pi \vdash r : \mathbf{realexp}]\!]C$$

`Denotal Sem`: Operadores binarios

$$\begin{aligned}&[\![\pi \vdash b \oslash b' : \mathbf{boolexp}]\!]C\ \eta\ \sigma = \\ &\qquad \oslash_{\circledcirc}\ ([\![\pi \vdash b : \mathbf{boolexp}]\!]C\ \eta\ \sigma)\ ([\![\pi \vdash b' : \mathbf{boolexp}]\!]C\ \eta\ \sigma)\end{aligned}$$

$$\text{con } \oslash \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$$

$$\begin{aligned}&[\![\pi \vdash i \oplus i' : \delta\mathbf{exp}]\!]C\ \eta\ \sigma = \\ &\qquad \oplus_{\circledcirc}\ ([\![\pi \vdash i : \delta\mathbf{exp}]\!]C\ \eta\ \sigma)\ ([\![\pi \vdash i' : \delta\mathbf{exp}]\!]C\ \eta\ \sigma)\end{aligned}$$

$$\text{con } \oplus \in \{+, -, *, /\ \mathbf{rem}\},\ \delta \in \{\mathbf{int}, \mathbf{real}\}$$

$$\begin{aligned}&[\![\pi \vdash e \ominus e' : \mathbf{boolexp}]\!]C\ \eta\ \sigma = \\ &\qquad \ominus_{\circledcirc}\ ([\![\pi \vdash e : \delta\mathbf{exp}]\!]C\ \eta\ \sigma)\ ([\![\pi \vdash e' : \delta\mathbf{exp}]\!]C\ \eta\ \sigma)\end{aligned}$$

$$\text{con } \ominus \in \{\leq, \geq, <, >\},\ \delta \in \{\mathbf{int}, \mathbf{real}\}$$

$$\begin{aligned}&[\![\pi \vdash e \ominus e' : \mathbf{boolexp}]\!]C\ \eta\ \sigma = \\ &\qquad \ominus_{\circledcirc}\ ([\![\pi \vdash e' : \delta\mathbf{exp}]\!]C\ \eta\ \sigma)\ ([\![\pi \vdash e : \delta\mathbf{exp}]\!]C\ \eta\ \sigma)\end{aligned}$$

$$\text{con } \ominus \in \{=, \neq\}$$

`Denotal Sem`: Expresión condicional.

$$\begin{aligned}&[\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \delta\mathbf{acc}]\!]C\ \eta\ z\ \sigma = \\ &\qquad (\lambda b.\ \mathtt{if}\ b\ \mathtt{then}\ [\![\pi \vdash e : \delta\mathbf{acc}]\!]C\ \eta\ z\ \sigma \\ &\qquad\qquad \mathtt{else}\ [\![\pi \vdash e' : \delta\mathbf{acc}]\!]C\ \eta\ z\ \sigma)_{\perp\!\!\perp}\ ([\![\pi \vdash b : \mathbf{boolexp}]\!]C\ \eta\ \sigma)\end{aligned}$$

$$\begin{aligned}&[\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \delta\mathbf{exp}]\!]C\ \eta\ \sigma = \\ &\qquad (\lambda b.\ \mathtt{if}\ b\ \mathtt{then}\ [\![\pi \vdash e : \delta\mathbf{exp}]\!]C\ \eta\ \sigma \\ &\qquad\qquad \mathtt{else}\ [\![\pi \vdash e' : \delta\mathbf{exp}]\!]C\ \eta\ \sigma)_{\perp\!\!\perp}\ ([\![\pi \vdash b : \mathbf{boolexp}]\!]C\ \eta\ \sigma)\end{aligned}$$

$$\begin{aligned}&[\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \mathbf{comm}]\!]C\ \eta\ \sigma = \\ &\qquad (\lambda b.\ \mathtt{if}\ b\ \mathtt{then}\ [\![\pi \vdash e : \mathbf{comm}]\!]C\ \eta\ \sigma \\ &\qquad\qquad \mathtt{else}\ [\![\pi \vdash e' : \mathbf{comm}]\!]C\ \eta\ \sigma)_{\perp\!\!\perp}\ ([\![\pi \vdash b : \mathbf{boolexp}]\!]C\ \eta\ \sigma)\end{aligned}$$

$$\begin{aligned}&[\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \delta\mathbf{var}]\!]C\eta = \\ &\quad \langle\ [\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \delta\mathbf{acc}]\!]C\eta\ ,\ [\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \delta\mathbf{exp}]\!]C\eta\ \rangle\end{aligned}$$

En este punto vamos a hacer un paréntesis para comentar la restricción en el tipado del condicional. Recordando, íbamos a evitar que el juicio de tipado $\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \theta \to \theta'$, sea valido; basándonos en las ecuaciones semánticas para los distintos juicios de tipado de la expresión condicional que

hemos dado, la razón entonces esta en notar que para evaluar el comando condicional tenemos que "rescatar" el estado para poder evaluar el juicio de la guarda $\pi \vdash b : \textbf{boolexp}$, entonces para el caso en que nuestra expresión condicional tipa a $\delta\textbf{exp}$ o **comm** es sencillo porque el estado esta "a la mano", ya para $\delta\textbf{acc}$ no es tan cierto pero no es una limitación vemos que primero tenemos que "rescatar" un z y después recién aparece el estado.
Veamos ahora algunas de las complicaciones cuando el tipo de nuestro juicio de tipado para la expresión condicional tiene varias flechas($\rightarrow$), supongamos tenemos el juicio $\pi \vdash \textbf{if}\ b\ \textbf{then}\ f\ \textbf{else}\ f' : \theta \rightarrow \theta' \rightarrow \theta''$ y recordemos que la semántica de $\theta_0 \rightarrow \theta_1$ se puede resumir en, tomar una posible expansión del estado, un valor para agregar al ambiente que será el valor pasado por argumento y finalmente aparece nuestro estado en alguna de las formas que sabemos "rescatarlo" siempre y cuando θ_1 no sea algo de tipo flecha.
La pregunta que surge entonces es ¿ Cómo saber cuantos $z's$ y $C's$ hay que rescatar hasta encontrarse el estado?; Continuando con el ejemplo, f podría ser la expresión $\lambda\iota.e$ tal que, abusando de notación, $\iota : \theta \rightarrow \theta'$. Esto implicaría rescatar un solo z y un solo C para encontrar el estado, una primera idea podría ser usar la información del tipo de ι ya que tenemos tipado explicito, luego tomaríamos este z y C y evaluaríamos cada caso de la expresión condicional pasando estos valores, parecido a como hacemos cuando tenemos el tipo $\delta\textbf{acc}$.
Esta entonces parecería ser una posible solución, pero surge otra complicación supongamos f' es igual a $(\lambda\iota'.\lambda\iota.e)e'$ es decir, f' tipa correctamente pero sin embargo no es una abstracción lambda, si no que hay una aplicación y esto implica entonces que rescatar el estado es distinto para una componente que para otra de la expresión condicional. En resumen, no es tan sencillo saber de qué forma "rescatar" el estado para evaluar la guarda.

Una solución podría llegar a tener que ver con siempre tener el estado "a mano" para poder rescatarlo", con esta idea en mente proponemos entonces la siguiente idea. Lo primero será agregar un tipo nuevo a nuestros $\langle \textit{Phrase Types} \rangle$, para esto entonces vamos a tener que acomodar la sintaxis y dar nuevas reglas de tipado, además de dar el significado semántico, tanto del tipo nuevo como de su nuevo juicio de tipado.

$\langle \textit{Phrase Types} \rangle ::= \langle \textit{Phrase Types} \rangle \mid \textbf{IF}\ \langle \textit{Phrase Types} \rangle$

Ahora con este nuevo tipo entonces lo que hacemos es actualizar el juicio de tipado de la expresión condicional de la siguiente manera

$$\frac{\pi \vdash b : \textbf{boolexp} \qquad \pi \vdash e : \theta \qquad \pi \vdash e' : \theta}{\pi \vdash \textbf{if}\ b\ \textbf{then}\ e\ \textbf{else}\ e' : \textbf{IF}\ \theta}$$

prestando especial atención a que ahora no restringimos θ, sacando $\rightarrow$. Donde el significado de nuestro tipo $\textbf{IF}\ \theta$ será un funtor de $\textbf{C} \longrightarrow \textbf{PDom}$ tal que, tomando una flecha $(h, s) : C \longrightarrow C'$

$[\![\textbf{IF}\ \theta]\!]C : \textbf{PDom}$
$[\![\textbf{IF}\ \theta]\!]C = C \rightarrow [\![\theta]\!]C$

$$[\![\mathbf{IF}\ \theta]\!](h,s) : [\![\mathbf{IF}\ \theta]\!]C \longrightarrow [\![\mathbf{IF}\ \theta]\!]C'$$
$$[\![\mathbf{IF}\ \theta]\!](h,s)\ \mathrm{if} = \mathrm{if} \circ h$$

para finalizar, actualizamos nuestra ecuación semántica para el juicio de tipado. Es interesante que con esta nueva definición no nos harán falta separa por casos en el tipo θ

$$[\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \mathbf{IF}\ \theta]\!]C\ \eta : [\![\mathbf{IF}\ \theta]\!]C$$
$$[\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \mathbf{IF}\ \theta]\!]C\ \eta : C \to [\![\theta]\!]C$$
$$[\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \mathbf{IF}\ \theta]\!]C\ \eta\ \sigma = (\lambda b.\ \mathrm{if}\ b\ \mathrm{then}\ [\![\pi \vdash e : \theta]\!]C\ \eta\ \ \mathrm{else}\ [\![\pi \vdash e' : \theta]\!]C\ \eta)_{\perp\!\!\!\perp}\ ([\![\pi \vdash b : \mathbf{boolexp}]\!]C\ \eta\ \sigma)$$

Ahora bien, esta posible solución no será estudiada en detalle y solamente vale como ejercicio para tener una idea mas formal de lo que sería practico tener como ecuación semántica para la expresión condicional. En el resto del capítulo vamos a seguir considerando la idea original.

Una aclaración importante sobre las ecuaciones para **rec** y **while** es que vamos a usar el operador **Y** de punto fijo bajo la seguridad de que $[\![\theta]\!]C$ es un dominio para cualquier θ.

`Denotal Sem`: Aplicación.

$$[\![\pi \vdash ee' : \theta']\!]C\ \eta = [\![\pi \vdash e : \theta \to \theta']\!]C\ \eta\ \langle\rangle\ ([\![\pi \vdash e' : \theta]\!]C\ \eta)$$

`Denotal Sem`: Operador de punto fijo.

$$[\![\pi \vdash \mathbf{rec}\ e : \theta]\!]C\ \eta = \mathbf{Y}_{[\![\theta]\!]C}\ ([\![\pi \vdash e : \theta \to \theta]\!]C\ \eta)\langle\rangle$$

`Denotal Sem`: Identificador.

$$[\![\pi \vdash \iota : \theta]\!]C\ \eta = \eta\ \iota \qquad \text{cuando } \iota : \theta \in \pi.$$

`Denotal Sem`: **skip**.

$$[\![\pi \vdash \mathbf{skip} : \mathbf{comm}]\!]C\ \eta\ \sigma = \iota_{\uparrow}\ \sigma$$

`Denotal Sem`: Composición.

$$[\![\pi \vdash e;e' : \mathbf{comm}]\!]C\ \eta = ([\![\pi \vdash e' : \mathbf{comm}]\!]C\ \eta)_{\perp\!\!\!\perp} \circ ([\![\pi \vdash e : \mathbf{comm}]\!]C\ \eta)$$

`Denotal Sem`: **while**.

$$[\![\pi \vdash \mathbf{while}\ b\ \mathbf{do}\ e : \mathbf{comm}]\!]C\ \eta = \mathbf{Y}_{[\![\mathbf{comm}]\!]C}\ (\lambda c.\ \lambda\sigma.\ (\lambda b.\ \mathrm{if}\ b\ \mathrm{then}\ c_{\perp\!\!\!\perp}([\![\pi \vdash e : \mathbf{comm}]\!]C\ \eta\ \sigma)\ \ \mathrm{else}\ \iota_{\uparrow}\ \sigma)_{\perp\!\!\!\perp}\ ([\![\pi \vdash b : \mathbf{boolexp}]\!]C\ \eta\ \sigma))$$

`Denotal Sem`: Asignación.

$$[\![\pi \vdash e := e' : \mathbf{comm}]\!] C\ \eta\ \sigma = (\lambda x.\ [\![\pi \vdash e : \delta\mathbf{acc}]\!] C\ \eta\ x\ \sigma)_{\perp\!\!\perp}([\![\pi \vdash e' : \delta\mathbf{exp}]\!] C\ \eta\ \sigma)$$

`Denotal Sem`: Abstracción lambda.

$$[\![\pi \vdash \lambda\iota_\theta.\ e : \theta \to \theta']\!] C\ \eta\ C'\ z = [\![\pi, \iota : \theta \vdash e : \theta']\!]\ (C + \!\! + \ C')\ [\ [\![\pi]\!](h,s)\ \eta \mid \iota : z\ ]$$

con $(h,s) : C \longrightarrow (C + \!\! + \ C')$

`Denotal Sem`: Subsunción.

$$[\![\pi \vdash e : \theta]\!] C = [\![\theta' \leq \theta]\!] C \circ [\![\pi \vdash e : \theta']\!] C$$

Hasta aquí hemos definido todas las ecuaciones semánticas de λ^{like} salvo para el comando **new** δ**var**, para este vamos a tener dos definiciones y con estas, en conjunto con las dos categorías de estados, vamos a completar las dos semánticas que estudiamos.

`Denotal Sem`: Declaración de variable (Stack discipline).

$$[\![\pi \vdash \mathbf{new}\ \delta\mathbf{var}\ \iota := ei\ \mathbf{in}\ c : \mathbf{comm}]\!] C\ \eta\ \sigma = H_\perp((\lambda\, v.\ [\![\pi, \iota : \delta\mathbf{var} \vdash c : \mathbf{comm}]\!]\ (C + \!\! + \ \langle S_\delta \rangle)\ \eta_{ext}\ \sigma_{ext})_{\perp\!\!\perp} ([\![\pi \vdash ei : \delta\mathbf{exp}]\!] C\ \eta\ \sigma))$$

con $(h,s) : C \longrightarrow (C + \!\! + \ \langle S_\delta \rangle)$

$$a = \lambda z.\ \lambda\widehat{\sigma}.\ \iota_\uparrow((head_C\widehat{\sigma}) + \!\! + \ \langle z \rangle)$$
$$e = \iota_\uparrow \circ last_{S_\delta}$$
$$\sigma_{ext} = \sigma + \!\! + \ \langle v \rangle$$
$$\eta_{ext} = [\ [\![\pi]\!](h,s)\ \eta \mid \iota : \langle a, e \rangle\ ]$$
$$H = head_C$$

Expliquemos la ecuación detenidamente, recordando que esta es la versión con stack discipline: el comportamiento que este comando debería tener es evaluar el comando c en un estado y ambiente extendidos con el identificador ι, este con el valor inicial correspondiente de haber evaluado ei y al terminar la evaluación de c contraer el estado y el ambiente, tirando de esa manera toda referencia al identificador ι.

Suponiendo siempre que la evaluación de nuestras frases y comandos termina, lo primero que hacemos es evaluar el juicio de tipado $\pi \vdash ei : \delta\mathbf{exp}$ con el ambiente y estado sin extender; esta evaluación nos retorna un valor que vamos a nombrar z, este será el valor para inicializar el identificador ι. Luego pasamos a evaluar $\pi \vdash c : \mathbf{comm}$, acá vamos a tener que extender el ambiente y el estado, así como también el conjunto de estados con el tipo básico del identificador.
Extender el estado será simplemente pegar por atrás el valor z al estado σ; extender el ambiente ya no es tan simple: por un lado si ocurre que agregamos el identificador ι con el par aceptador-evaluador por atrás pero no al ambiente

tal cual lo teníamos, sino a un nuevo ambiente que es el resultado de acomodar el ambiente original a la extensión del estado, mas adelante vamos a ver por qué esto es importante. Sobre el par aceptador-evaluador podemos notar que evaluar ι como expresión será evaluar e, es decir, será obtener el ultimo elemento del estado y evaluar ι como aceptador será recortar el estado hasta justo antes del valor actual y pegar por atrás el nuevo valor.
Finalmente al terminar de evaluar $\pi \vdash c : \textbf{comm}$ obtenemos un estado con forma $C +\!\!+ \langle S_\delta \rangle$ y necesitamos contraerlo para eliminar el valor de ι, esto lo hace $H_\perp$ recortando el estado hasta justo antes del valor, tal como pasaba con la evaluación de ι como aceptador pero sin agregar ningún valor nuevo.

Repasemos con un ejemplo todo lo comentado anteriormente.

Ejemplo Supongamos tenemos un comando (programa),

> **new intvar** $\iota := 0$ **in**
>   **new intvar** $\kappa := 1$ **in**
> $\kappa := \iota$;
> **skip**

y la siguiente derivación que nos prueba que esta bien tipado para obtener el juicio de tipado $\vdash \textbf{new}\iota \textbf{ ; skip} : \textbf{comm}$, donde $\textbf{new}\iota$ es el primer comando **newintvar**, además vamos a llamar $\textbf{new}\kappa$ al otro comando **newintvar**. Dividimos la derivación en tres partes, en las que $\pi_{\xi_0,\ldots,\xi_n}$, representa el contexto con $\xi_0, \ldots, \xi_n$ identificadores.

$$\dfrac{\dfrac{\dfrac{}{\pi_{\iota,\kappa} \vdash \kappa : \textbf{intvar}} \quad \textbf{intvar} \leq \textbf{intexp}}{\pi_{\iota,\kappa} \vdash \kappa : \textbf{intacc}} \qquad \dfrac{\dfrac{}{\pi_{\iota,\kappa} \vdash \iota : \textbf{intvar}} \quad \textbf{intvar} \leq \textbf{intexp}}{\pi_{\iota,\kappa} \vdash \iota : \textbf{intexp}}}{\pi_\iota, \kappa : \textbf{intvar} \vdash \kappa := \iota : \textbf{comm}}$$

$$\dfrac{\dfrac{}{\pi_\iota \vdash 1 : \textbf{intexp}} \qquad \dfrac{}{\pi_\iota, \kappa : \textbf{intvar} \vdash \kappa := \iota : \textbf{comm}}}{\iota : \textbf{intvar} \vdash \textbf{new}\kappa : \textbf{comm}}$$

$$\dfrac{\dfrac{\dfrac{}{\vdash 0 : \textbf{intexp}} \qquad \dfrac{}{\iota : \textbf{intvar} \vdash \textbf{new}\kappa : \textbf{comm}}}{\vdash \textbf{new}\iota : \textbf{comm}} \qquad \dfrac{}{\vdash \textbf{skip} : \textbf{comm}}}{\vdash \textbf{new}\iota \textbf{ ; skip} : \textbf{comm}}$$

Ahora que tenemos nuestro juicio de tipado $\vdash \textbf{new}\iota \textbf{ ; skip} : \textbf{comm}$ podemos continuar con la evaluación, podemos empezar evaluando a vista y notar que a lo que pretendemos que evalué este juicio es a $\langle\rangle$, el estado vacío, pero además y es la principal razón por la que esta el comando **skip**, es ver que luego de ejecutar un **new** δ**var** el comando que siga a este no sabe nada acerca de lo que paso dentro y con los identificadores. Tomamos ambiente y estado vacíos,

$$[\![\vdash \textbf{new}\iota \textbf{ ; skip} : \textbf{comm}]\!]\langle\rangle[\,]\langle\rangle = \\ \qquad ((\,[\![\vdash \textbf{skip} : \textbf{comm}]\!]\langle\rangle[\,])_{\perp\!\!\perp} \circ [\![\vdash \textbf{new}\iota : \textbf{comm}]\!]\langle\rangle[\,])\langle\rangle$$

ahora pasemos a evaluar juicio de tipado para $\mathbf{new}\iota$

$$\begin{aligned}&[\![\vdash \mathbf{new}\iota : \mathbf{comm}]\!]\langle\rangle[\,]\langle\rangle = \\ &\quad H_{\perp}((\lambda\, v.\ [\![\iota : \mathbf{intvar} \vdash \mathbf{new}\kappa : \mathbf{comm}]\!]\ (\langle\rangle + \langle S_{\mathbf{int}}\rangle)\ \eta_{ext}\ \sigma_{ext})_{\perp\!\!\!\perp} \\ &\qquad\qquad ([\![\vdash 0 : \mathbf{intexp}]\!]\langle\rangle\ [\,]\ \langle\rangle))\end{aligned}$$

donde η_{ext} será simplemente agregar el identificador ι y el par $\langle a_\iota, e_\iota\rangle$ y σ_{ext} será la lista de valores con un único valor, el 0.

$$[\![\vdash \mathbf{new}\iota : \mathbf{comm}]\!]\langle\rangle[\,]\langle\rangle = H_{\perp}[\![\iota : \mathbf{intvar} \vdash \mathbf{new}\kappa : \mathbf{comm}]\!]\ \langle S_{\mathbf{int}}\rangle\ [\ \iota : \langle a_\iota, e_\iota\rangle\ ]\ \langle 0\rangle$$

$$\begin{aligned}\text{con } a_\iota &= \lambda v.\ \lambda\widehat{\sigma}.\ \iota_{\uparrow}((\mathtt{head}_{\langle\rangle}\,\widehat{\sigma}) + \langle v\rangle) \\ e_\iota &= \iota_{\uparrow} \circ \mathtt{last}_{S_{\mathbf{int}}}\end{aligned}$$

evaluamos el $\mathbf{new}\kappa$ como hicimos con $\mathbf{new}\iota$ pero con el cuidado de que ahora no tenemos forma, ni ambiente, ni estado vacíos

$$\begin{aligned}&[\![\pi_\iota \vdash \mathbf{new}\kappa : \mathbf{comm}]\!]\ \langle S_{\mathbf{int}}\rangle\ [\ \iota : \langle a_\iota, e_\iota\rangle\ ]\ \langle 0\rangle = \\ &\quad H_{\perp}((\lambda\, v.\ [\![\pi_\iota, \kappa : \mathbf{intvar} \vdash \kappa := \iota : \mathbf{comm}]\!]\ (\langle S_{\mathbf{int}}\rangle + \langle S_{\mathbf{int}}\rangle)\ \eta_{ext}\ \sigma_{ext})_{\perp\!\!\!\perp} \\ &\qquad\qquad ([\![\pi_\iota \vdash 1 : \mathbf{intexp}]\!]\langle\rangle\ [\,]\ \langle\rangle))\end{aligned}$$

como antes, tenemos que ver quiénes serán η_{new} y σ_{new}; empezando por este último es sencillo y es simplemente pegar por atrás el valor 1. En cuanto a η_{new} ya no es tan directo, lo que sucede es que vamos a tener que acomodar el antiguo ambiente, $[\ \iota : \langle a_\iota, e_\iota\rangle\ ]$, y luego agregar el identificador κ con el par $\langle a_\kappa, e_\kappa\rangle$.
Para acomodar el ambiente utilizamos $[\![\pi_\iota]\!](h, s)$ donde $\langle S_{\mathbf{int}}\rangle \xrightarrow{(h,s)} \langle S_{\mathbf{int}}, S_{\mathbf{int}}\rangle$, es decir, este funtor aplicado a la flecha (h, s) toma un ambiente de tipo $[\![\pi_\iota]\!]\langle S_{\mathbf{int}}\rangle$ y nos retorna un ambiente con tipo $[\![\pi_\iota]\!]\langle S_{\mathbf{int}}, S_{\mathbf{int}}\rangle$.

$$[\![\pi_\iota]\!](h, s)\ [\ \iota : \langle a_\iota, e_\iota\rangle\ ] = [\ \iota : [\![\pi_\iota\iota]\!](h, s)\langle a_\iota, e_\iota\rangle\ ] =$$

$$[\ \iota : [\![\mathbf{intvar}]\!](h, s)\langle a_\iota, e_\iota\rangle\ ] = [\ \iota : \langle[\![\mathbf{intacc}]\!](h, s)a_\iota, [\![\mathbf{intacc}]\!](h, s)e_\iota\rangle\ ] =$$

$$[\ \iota : \langle s \circ a_\iota, e_\iota \circ h\rangle\ ]$$

ahora sí podemos extender el ambiente $[\ \iota : \langle s \circ a_\iota, e_\iota \circ h\rangle \mid \kappa : \langle a_\kappa, e_\kappa\rangle\ ]$ y este será nuestro nuevo ambiente η_{new}. Luego podemos avanzar un poco más con la evaluación y pasar en limpio la ultima ecuación,

$$\begin{aligned}&[\![\pi_\iota \vdash \mathbf{new}\kappa : \mathbf{comm}]\!]\ \langle S_{\mathbf{int}}\rangle\ [\ \iota : \langle a_\iota, e_\iota\rangle\ ]\ \langle 0\rangle = \\ &\quad H_{\perp}[\![\pi_{\iota,\kappa} \vdash \kappa := \iota : \mathbf{comm}]\!]\ \langle S_{\mathbf{int}}, S_{\mathbf{int}}\rangle\ [\ \iota : \langle s \circ a_\iota, e_\iota \circ h\rangle \mid \kappa : \langle a_\kappa, e_\kappa\rangle\ ]\ \langle 0, 1\rangle\end{aligned}$$

$$\begin{aligned}\text{con } a_\kappa &= \lambda v.\ \lambda\widehat{\sigma}.\ \iota_{\uparrow}((\mathtt{head}_{\langle S_{\mathbf{int}}\rangle}\,\widehat{\sigma}) + \langle v\rangle) \\ e_\kappa &= \iota_{\uparrow} \circ \mathtt{last}_{S_{\mathbf{int}}} \\ (s \circ a_\iota)\ x\ \widehat{\sigma} &= (\lambda\sigma.\ \sigma + \mathtt{tail}_{\langle S_{\mathbf{int}}\rangle}\,\widehat{\sigma})_{\perp}(a_\iota\ x\ (\mathtt{head}_{\langle S_{\mathbf{int}}\rangle}\,\widehat{\sigma})) \\ e_\iota \circ h &= e_\iota \circ \mathtt{head}_{\langle S_{\mathbf{int}}\rangle}.\end{aligned}$$

Haciendo un repaso general de lo que llevamos hecho, tenemos evaluado hasta el comando de asignación, esto implica haber evaluado los comandos para introducir variables imperativas, lo mas importante a repasar son los pares aceptor-evaluador de los identificadores ι y κ. Empezando con ι, vemos que usarlo como expresión es utilizar como e_ι que sabe cómo retornar el valor asociado a ι dado un estado con forma $\langle S_{\mathbf{int}} \rangle$, luego acomodando esto al introducir κ, lo que sucede es que usar como expresión a ι es utilizar $e_\iota \circ head_{\langle S_{\mathbf{int}} \rangle}$, es decir, dado un estado con forma $\langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle$, primero nos quedamos con la parte inicial sin la extensión del identificador κ y luego ya podemos utilizar e_ι en un estado con forma $\langle S_{\mathbf{int}} \rangle$. Notar que si fuera el caso general de agregar n variables imperativas anidadas, supongamos $\xi_1, \ldots, \xi_n$, entonces para rescatar la variable i-esima vamos a hacer $n - i$ $head$'s y luego aparece la definición e_{ξ_i} original de cuando se introdujo el identificador.
Si en cambio usamos a ι como aceptador de valor, entonces será utilizar a_ι que sabe recortar un estado con forma $\langle S_{\mathbf{int}} \rangle$ de manera de perder justo el valor asociado a ι; al agregar κ hay que acomodar la función a utilizar cuando usamos ι como aceptador, para esto vamos a usar $s : (\langle S_{\mathbf{int}} \rangle \to \langle S_{\mathbf{int}} \rangle_\perp) \to (\langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle \to \langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle_\perp)$, esta función básicamente va a tomar $a\ x : \langle S_{\mathbf{int}} \rangle \to \langle S_{\mathbf{int}} \rangle_\perp$, recortar un la parte inicial de un estado con forma $\langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle$, aplicar $a\ x$ en ese estado y luego concatenar el resultado de esa aplicación con la parte final del estado original con forma $\langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle$.
Notar que para las funciones e_κ y a_κ el comportamiento es análogo al comportamiento para e_ι y a_ι cuando no se agregan variables imperativas nuevas. Hecho este repaso, podemos continuar con la evaluación del juicio de tipado de la asignación, acá vamos a ver como es que utilizamos un identificador tanto como aceptor y como valor.

$$\begin{aligned}
&[\![\pi_{\iota,\kappa} \vdash \kappa := \iota : \mathbf{comm}]\!]\ \langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle\ \eta_{\iota,\kappa}\ \langle 0, 1 \rangle = \\
&\qquad (\lambda x.\ [\![\pi_{\iota,\kappa} \vdash \kappa : \mathbf{intacc}]\!] \langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle\ \eta_{\iota,\kappa}\ x\ \langle 0, 1 \rangle)_{\perp\!\!\perp} \\
&\qquad\qquad ([\![\pi_{\iota,\kappa} \vdash \iota : \mathbf{intexp}]\!]\ \langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle\ \eta_{\iota,\kappa}\ \langle 0, 1 \rangle)
\end{aligned}$$

evaluemos detenidamente el juicio de tipado para ι: es importante notar que en la derivación de nuestro juicio de tipado para tipar ι no alcanza con simplemente buscar en el contexto, ya que ι tiene tipo **intvar** entonces hay que usar una regla de subtipado, por lo tanto esto impacta de la siguiente manera en nuestra evaluación

$$\begin{aligned}
&[\![\pi_{\iota,\kappa} \vdash \iota : \mathbf{intexp}]\!]\ \langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle\ \eta_{\iota,\kappa}\ \langle 0, 1 \rangle = \\
&([\![\mathbf{intvar} \leq \mathbf{intexp}]\!]\ \langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle \circ [\![\pi_{\iota,\kappa} \vdash \iota : \mathbf{intvar}]\!]\ \langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle)\ \eta_{\iota,\kappa}\ \langle 0, 1 \rangle = \\
&[\![\mathbf{intvar} \leq \mathbf{intexp}]\!]\ \langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle (\eta_{\iota,\kappa}\ \iota)\ \langle 0, 1 \rangle = \\
&[\![\mathbf{intvar} \leq \mathbf{intexp}]\!]\ \langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle (\langle s \circ a_\iota, e_\iota \circ h \rangle)\ \langle 0, 1 \rangle = (e_\iota \circ h)\ \langle 0, 1 \rangle = \\
&(e_\iota \circ head_{\langle S_{\mathbf{int}} \rangle}) \langle 0, 1 \rangle = \iota_\uparrow(last_{S_{\mathbf{int}}}(head_{\langle S_{\mathbf{int}} \rangle}\ \langle 0, 1 \rangle)) = \iota_\uparrow(last_{S_{\mathbf{int}}} \langle 0 \rangle) = \iota_\uparrow 0
\end{aligned}$$

volviendo ahora con ι evaluado como expresión

$$\begin{aligned}
&[\![\pi_{\iota,\kappa} \vdash \kappa := \iota : \mathbf{comm}]\!]\ \langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle\ \eta_{\iota,\kappa}\ \langle 0, 1 \rangle = \\
&\qquad [\![\pi_{\iota,\kappa} \vdash \kappa : \mathbf{intacc}]\!] \langle S_{\mathbf{int}}, S_{\mathbf{int}} \rangle\ \eta_{\iota,\kappa}\ 0\ \langle 0, 1 \rangle
\end{aligned}$$

ahora evaluando similar a como hicimos con ι y teniendo el mismo cuidado con el subtipado del identificador κ de **intvar** a **intacc**

$$\begin{aligned}
&[\![\pi_{\iota,\kappa} \vdash \kappa : \mathbf{intacc}]\!]\langle S_{\mathbf{int}}, S_{\mathbf{int}}\rangle\, \eta_{\iota,\kappa}\; 0\; \langle 0, 1\rangle = \\
&([\![\mathbf{intvar} \leq \mathbf{intacc}]\!]\; \langle S_{\mathbf{int}}, S_{\mathbf{int}}\rangle \circ [\![\pi_{\iota,\kappa} \vdash \kappa : \mathbf{intvar}]\!]\; \langle S_{\mathbf{int}}, S_{\mathbf{int}}\rangle)\; \eta_{\iota,\kappa}\; 0\; \langle 0, 1\rangle = \\
&[\![\mathbf{intvar} \leq \mathbf{intacc}]\!]\; \langle S_{\mathbf{int}}, S_{\mathbf{int}}\rangle(\eta_{\iota,\kappa}\kappa)\; 0\; \langle 0, 1\rangle = \\
&[\![\mathbf{intvar} \leq \mathbf{intacc}]\!]\; \langle S_{\mathbf{int}}, S_{\mathbf{int}}\rangle\langle a_\kappa, e_\kappa\rangle\; 0\; \langle 0, 1\rangle = a_\kappa\; 0\; \langle 0, 1\rangle = \\
&\iota_\uparrow((\mathrm{head}_{\langle S_{\mathbf{int}}\rangle}\langle 0, 1\rangle) +\!\!+\; \langle 0\rangle) = \iota_\uparrow(\langle 0\rangle +\!\!+\; \langle 0\rangle) = \iota_\uparrow\langle 0, 0\rangle
\end{aligned}$$

de nuevo, reescribiendo ahora con la evaluación terminada de κ, tenemos

$$[\![\pi_{\iota,\kappa} \vdash \kappa := \iota : \mathbf{comm}]\!]\; \langle S_{\mathbf{int}}, S_{\mathbf{int}}\rangle\; \eta_{\iota,\kappa}\; \langle 0, 1\rangle = \iota_\uparrow\langle 0, 0\rangle$$

y con esto ya casi terminamos, ahora volviendo a la evaluación de la composición entre **new**ι y **skip**,

$$\begin{aligned}
&([\![\vdash \mathbf{skip} : \mathbf{comm}]\!]\langle\rangle[\,])_{\perp\!\!\!\perp}([\![\vdash \mathbf{new}\iota : \mathbf{comm}]\!]\langle\rangle[\,]\langle\rangle) = \\
&(\lambda\sigma.\; \iota_\uparrow\sigma)_{\perp\!\!\!\perp}(H_\perp(H_\perp\iota_\uparrow\langle 0, 0\rangle)) = \\
&(\lambda\sigma.\; \iota_\uparrow\sigma)_{\perp\!\!\!\perp}((\mathrm{head}_{\langle\rangle})_\perp((\mathrm{head}_{\langle S_{\mathbf{int}}\rangle})_\perp\iota_\uparrow\langle 0, 0\rangle)) = \\
&(\lambda\sigma.\; \iota_\uparrow\sigma)_{\perp\!\!\!\perp}(\mathrm{head}_{\langle\rangle} \circ \mathrm{head}_{\langle S_{\mathbf{int}}\rangle})_\perp\iota_\uparrow\langle 0, 0\rangle = \\
&(\lambda\sigma.\; \iota_\uparrow\sigma)_{\perp\!\!\!\perp}((\mathrm{head}_{\langle\rangle}(\mathrm{head}_{\langle S_{\mathbf{int}}\rangle}\; \langle 0, 0\rangle))) = (\lambda\sigma.\; \iota_\uparrow\sigma)_{\perp\!\!\!\perp}(\iota_\uparrow\langle\rangle) = \iota_\uparrow\langle\rangle
\end{aligned}$$

y con esto terminamos la evaluación del juicio de tipado completo y además con el resultado esperado.

A continuación vamos a introducir nuestra versión de ecuación semántica que en conjunto con la segunda categoría de estados que dimos en la definición 17 conforman la semántica sin stack discipline. Lo que pretendemos con esta semántica sería, que al terminar de evaluar el comando **new**δ**var** no exista compactación del almacén, a diferencia de lo que paso en el ejemplo anterior que al terminar de evaluar un comando **new**δ**var** el siguiente comando no sabe nada acerca de los identificadores y valores a los que se evaluó.

En el momento en que introducimos las definiciones de estados mencionábamos que, para esta segunda versión, íbamos a abandonar el hecho de que los estados tengan forma, ahora que avanzamos más podemos comentar la razón: uno puede pensar para la versión con stack discipline que la forma del estado está totalmente relacionada con la cantidad, y el tipo, de identificadores que se han introducido y son accesibles en cierto comando anidado del **new**δ**var**, de esta manera cuando terminábamos de evaluar un **new**δ**var** automáticamente perdíamos un elemento del estado. La cuestión ahora es que, nosotros queremos que la cantidad de valores de un estado no tenga relación con la cantidad de comandos para introducir identificadores.

`Denotal Sem`: Declaración de variable (Sin stack discipline).

$$\begin{aligned}&[\![\pi \vdash \textbf{new}\ \delta\textbf{var}\ \iota := e\ \textbf{in}\ c : \textbf{comm}]\!] C\, \eta\, \sigma = \\ &\quad H_{\perp}((\lambda\, \nu.\, [\![\pi, \iota : \delta\textbf{var} \vdash c : \textbf{comm}]\!]\, C\, \eta_{new}\, \sigma_{new})_{\perp\!\!\!\perp}\, ([\![\pi \vdash e : \delta\textbf{exp}]\!] C\, \eta\, \sigma))\end{aligned}$$

$$\begin{aligned}\text{con }& a\, \nu\, \sigma = \iota_{\uparrow}[\, \sigma \mid \iota : \iota_{\delta}\nu\,] \\ & e\, \sigma = (\lambda\nu.\nu)_{\delta}\sigma\iota \\ & \sigma_{new} = [\, \sigma \mid \iota : \iota_{\delta}\nu\,] \\ & \eta_{new} = [\, \eta \mid \iota : \langle a, e\rangle\,] \\ & H = 1_{C}\end{aligned}$$

De la misma forma que hicimos con la primera ecuación semántica de este comando, repasemos cada parte de la nueva ecuación. En general todo sobre la nueva ecuación es igual, los detalles importantes están en las funciones a, e y H, empecemos notando que H la cual en la otra semántica era la encargada de recortar el estado cuando terminaba la evalución del cuerpo, ahora es la identidad, es decir, vamos a retornar el estado tal cual estaba al terminar de evaluar el cuerpo. La función para evaluar el identificador como aceptador, es decir a, sobrescribe el valor antiguo del estado, lo nuevo que aparece acá es el ι_{δ} esta función es la inyección de S_{δ} en S, recordando que S es la unión disjunta de $S_{\textbf{int}}$, $S_{\textbf{real}}$ y $S_{\textbf{bool}}$.
Para explicar la función para evaluar el identificador como valor, primero vamos a introducir la función $(_)_{\delta}$ que dada una f de S_{δ} en S_{δ} se define así,

$$f_{\delta}(\iota_{\delta}x) = f\, x$$

luego simplemente será aplicar el estado al identificador, la única salvedad importante es que $\sigma\iota : S$ y nosotros necesitamos algo de tipo S_{δ}, para esto utilizamos $(\lambda\nu.\ \nu)_{\delta}$.

Existe una observación importantes que surge de pensar el siguiente caso, supongamos tenemos un $e : \Sigma \to S_{\textbf{int}}$ y un $\sigma\iota : S_{\textbf{bool}}$, luego e podría estar definido así $(\lambda\nu.\nu)_{\textbf{int}}\sigma\iota$, esto es un problema ya que $(\lambda\nu.\nu)_{\textbf{int}}$ espera algo de tipo entero y $\sigma\iota$ es algo de tipo booleano, para manejar esto deberíamos introducir todo un manejo extra de errores de tipos en momento de evaluación, y como consecuencia nuestra semántica sin stack discipline generaría una nueva clase de errores que la semántica con stack discipline no genera ya que, según el primer principio, todos los errores de tipos deben ser sintácticos. Por esta razón vamos a completar la definición de $(_)_{\delta}$ de la siguiente manera $f_{\delta}(\iota_{\delta'}x) = \perp$.

Pero es interesante pensar en qué casos nuestra semántica sin stack discipline generaría estos errores, nos damos cuenta que esto puede pasar solo cuando intentamos evaluar juicios de tipado no cerrados, es decir que contienen variables libre. Analicemos por qué esta afirmación, empecemos suponiendo que tenemos un comando cerrado y que tenemos un σ cualquiera, es decir que por ejemplo tiene identificadores con valores asociados, luego cada vez que aparezca un identificador como comando, lo que tiene que haber ocurrido es que este identificador es un subcomando de uno o varios **new**δ**var** y por lo tanto al evaluar este comando pisamos la posible existencia de un identificador asociado a un valor del tipo inadecuado. El problema entonces surge cuando

evaluamos un comando no cerrado, ya que es posible que nos encontremos con un identificador como comando a evaluar, sin que este sea subcomando de uno o varios **new**δ**var** y por lo tanto el tipo del valor asociado al identificador en el estado no sea el correcto. Otro problema que podría surgir sería que no existe el identificador ι en el dominio de nuestro estado σ, es decir, estaríamos intentando utilizar un identificador que no aparece en el estado.

La solución a esto es pensar que, para los programas que vamos a querer evaluar o que pretendemos sean programas validos a evaluar, van a ser siempre cerrados, por lo tanto este comportamiento extraño que podría llegar a tener nuestro evaluador sin stack discipline puede ser despreciable basándonos en esta suponsición, y que será una restricción importante en caso de comparar las semánticas de uno y otro.

4.4. Naturalidad de las ecuaciones semánticas

Para el lenguaje $\lambda^{\rightarrow}$, y por lo tanto también para $\lambda^{\leqslant}$, probamos que las ecuaciones estuvieran bien definidas, es decir que fueran funciones continuas. Ahora vamos a hacer lo mismo para las ecuaciones de λ^{like} con la diferencia de que por las definiciones categóricas que hicimos vamos a probar que nuestras ecuaciones sean transformaciones naturales. Dado un juicio de tipado valido, $\pi \vdash e : \theta$, vamos a querer probar que el siguiente diagrama conmuta, donde $(h, s) : C \longrightarrow C'$ en $\mathbf{C}$

$$\begin{array}{ccc} [\![\pi]\!]C & \xrightarrow{[\![\pi \vdash e : \theta]\!]C} & [\![\theta]\!]C \\ {\scriptstyle [\![\pi]\!](h,s)}\downarrow & & \downarrow{\scriptstyle [\![\theta]\!](h,s)} \\ [\![\pi]\!]C' & \xrightarrow{[\![\pi \vdash e : \theta]\!]C'} & [\![\theta]\!]C' \end{array}$$

Proposición 3. *Sea* $(h, s) : C \longrightarrow C'$ *y sea* $f : C \rightarrow C_{\perp}$ *y* $f' : C \rightarrow C_{\perp}$, *entonces vale*

$$s(f'_{\perp\!\!\perp} \circ f) = s(f')_{\perp\!\!\perp} \circ s(f)$$

Teorema 5 (de naturalidad)**.** *Dado un juicio de tipado valido* $\pi \vdash e : \theta$ *y una flecha* $C \xrightarrow{(h,s)} C'$ *en* $\mathbf{C}$ *se cumple,*

$$[\![\theta]\!](h, s) \circ [\![\pi \vdash e : \theta]\!]C = [\![\pi \vdash e : \theta]\!]C' \circ [\![\pi]\!](h, s)$$

Demostración. Procedamos por inducción en la estructura de la derivación de los juicios de tipado, supongamos un $\eta : [\![\pi]\!]C$ y un $\sigma' : C'$ tal que $\sigma : C$, $\overline{\sigma} : \overline{C}$, $C +\!\!+ \overline{C} = C'$ y $\sigma +\!\!+ \overline{\sigma} = \sigma'$.

- Casos base

  - Para la introducción de constantes consideramos $\pi \vdash b : \mathbf{boolexp}$, los casos de **intexp** y **realexp** son análogos.

    $[\![\mathbf{boolexp}]\!](h, s)([\![\pi \vdash b : \mathbf{boolexp}]\!]C\,\eta)\sigma' = [\![\mathbf{boolexp}]\!](h, s)(\lambda\sigma.\,\iota\, b)\sigma'$
    $=$
    $((\lambda\sigma.\,\iota\, b)\circ h)\sigma' = (\lambda\sigma.\,\iota\, b)(h\sigma') = \iota\, b = [\![\pi \vdash b : \mathbf{boolexp}]\!]C'([\![\pi]\!](h, s)\eta)\sigma'$

  - Supongamos $\pi \vdash \iota : \theta$,

    por un lado tenemos, $[\![\theta]\!](h, s)([\![\pi \vdash \iota : \theta]\!]C\,\eta) = [\![\theta]\!](h, s)(\eta\iota)$

    y por el otro, $[\![\pi \vdash \iota : \theta]\!]C'([\![\pi]\!](h, s)\eta) = ([\![\pi]\!](h, s)\eta)\iota$

    donde utilizando la definición de $[\![\pi]\!](h, s)\eta$, obtenemos

    $([\![\pi]\!](h, s)\eta)\iota = [\![\pi\iota]\!](h, s)(\eta\iota) = [\![\theta]\!](h, s)(\eta\iota)$

- Casos inductivos

  - Para introducción de operadores unarios consideramos $\pi \vdash \neg e : \mathbf{boolexp}$, los demás casos, incluidos los operadores binarios, son análogos.

    $[\![\mathbf{boolexp}]\!](h, s)([\![\pi \vdash \neg e : \mathbf{boolexp}]\!]C\eta)\sigma' =$
    $[\![\mathbf{boolexp}]\!](h, s)(\lambda\sigma.\,\neg_{\odot}([\![\pi \vdash e : \mathbf{boolexp}]\!]C\eta\sigma))\sigma' =$
    $((\lambda\sigma.\,\neg_{\odot}([\![\pi \vdash e : \mathbf{boolexp}]\!]C\eta\sigma)) \circ h)\sigma' =$
    $\neg_{\odot}([\![\pi \vdash e : \mathbf{boolexp}]\!]C\eta(h\sigma')) =$
    $\neg_{\odot}(([\![\pi \vdash e : \mathbf{boolexp}]\!]C\eta \circ h)\sigma')) =$
    $\neg_{\odot}([\![\mathbf{boolexp}]\!](h, s)([\![\pi \vdash e : \mathbf{boolexp}]\!]C\eta)\sigma')$

    hasta acá ha sido simplemente reescribir usando definiciones, de esta manera llegamos a poder aplicar la hipótesis inductiva,

    $\neg_{\odot}([\![\mathbf{boolexp}]\!](h, s)([\![\pi \vdash e : \mathbf{boolexp}]\!]C\eta)\sigma')$
    $\neg_{\odot}([\![\pi \vdash e : \mathbf{boolexp}]\!]C([\![\pi]\!](h, s)\eta)\sigma') =$
    $[\![\pi \vdash \neg e : \mathbf{boolexp}]\!]C([\![\pi]\!](h, s)\eta)\sigma'$

  - Supongamos $\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \theta$, para este caso nos vamos a enfocar en probar el caso para $\theta = \delta\mathbf{exp}$ ya que los demás son análogos o incluso iguales pensando en **comm**.

    $[\![\delta\mathbf{exp}]\!](h, s)([\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \delta\mathbf{exp}]\!]C\eta)\sigma' =$
    $([\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \delta\mathbf{exp}]\!]C\eta \circ h)\sigma' =$
    $[\![\pi \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e' : \delta\mathbf{exp}]\!]C\eta(h\sigma') =$
    $(\lambda b.\ \mathtt{if}\ b\ \mathtt{then}\ [\![\pi \vdash e : \delta\mathbf{exp}]\!]C\ \eta\ (h\sigma')$
    $\qquad \mathtt{else}\ [\![\pi \vdash e' : \delta\mathbf{exp}]\!]C\ \eta\ (h\sigma'))_{\perp\!\!\perp}$
    $\qquad\qquad ([\![\pi \vdash b : \mathbf{boolexp}]\!]C\ \eta\ (h\sigma'))$

resolviendo el otro lado de la igualdad,

$$
\begin{aligned}
&[\![\pi \vdash \textbf{if } b \textbf{ then } e \textbf{ else } e' : \delta\textbf{exp}]\!] C'([\![\pi]\!](h,s)\eta)\sigma' = \\
&(\lambda b.\ \text{if } b \text{ then } [\![\pi \vdash e : \delta\textbf{exp}]\!] C'\ ([\![\pi]\!](h,s)\eta)\ \sigma' \\
&\qquad\quad \text{else } [\![\pi \vdash e' : \delta\textbf{exp}]\!] C'\ ([\![\pi]\!](h,s)\eta)\ \sigma')_{\perp\!\!\perp} \\
&\qquad\qquad\qquad ([\![\pi \vdash b : \textbf{boolexp}]\!] C'\ ([\![\pi]\!](h,s)\eta)\ \sigma')
\end{aligned}
$$

ahora vamos a usar hipótesis inductiva en el juicio de tipado de b,

$$[\![\pi \vdash b : \textbf{boolexp}]\!] C\ \eta\ (h\sigma') = [\![\pi \vdash b : \textbf{boolexp}]\!] C'\ ([\![\pi]\!](h,s)\eta)\ \sigma'$$

y entonces vamos a suponer que la evaluación de juicio de tipado $\pi \vdash b : \textbf{boolexp}$ es verdadera, luego tenemos que ver que

$$[\![\pi \vdash e : \delta\textbf{exp}]\!] C\ \eta\ (h\sigma') = [\![\pi \vdash e : \delta\textbf{exp}]\!] C'\ ([\![\pi]\!](h,s)\eta)\ \sigma'$$

pero esto es directo usando hipótesis inductiva. Por otro lado, si supusiéramos la evaluación $\pi \vdash b : \textbf{boolexp}$ es falso, entonces es directo de la misma manera que antes.

- Supongamos $\pi \vdash ee' : \theta'$,

$$
\begin{aligned}
&[\![\pi \vdash ee' : \theta']\!] C'([\![\pi]\!](h,s)\eta) = \\
&[\![\pi \vdash e' : \theta \to \theta']\!] C'([\![\pi]\!](h,s)\eta)\langle\rangle([\![\pi \vdash e : \theta]\!] C'([\![\pi]\!](h,s)\eta)) = \\
&([\![\theta \to \theta']\!](h,s)([\![\pi \vdash e' : \theta \to \theta']\!] C\eta))\langle\rangle([\![\theta]\!](h,s)([\![\pi \vdash e : \theta]\!] C\eta)) = \\
&[\![\pi \vdash e' : \theta \to \theta']\!] C\eta\overline{C}([\![\theta]\!](h,s)([\![\pi \vdash e : \theta]\!] C\eta))
\end{aligned}
$$

hasta acá usamos la definición de la ecuación semántica de la aplicación, aplicamos hipótesis inductiva dos veces y usamos la definición del funtor $[\![\theta \to \theta']\!]$ aplicado a (h,s) y $([\![\pi \vdash e' : \theta \to \theta']\!] C\eta)$, donde $\overline{C}$ es tal que $C +\!\!+\ \overline{C} = C'$.
Ahora si analizamos un poco más la definición de este funtor que mencionábamos nos damos cuenta que $[\![\pi \vdash e' : \theta \to \theta']\!] C\eta$ es una transformación natural indexada por objetos de $\mathbf{C}$ que va del funtor $[\![\theta]\!]$ al funtor $[\![\theta']\!]$, luego el siguiente diagrama debe conmutar,

$$
\begin{array}{ccc}
[\![\theta]\!] C & \xrightarrow{\ [\![\pi \vdash e' : \theta \to \theta']\!] C\eta\langle\rangle\ } & [\![\theta']\!] C \\
{\scriptstyle [\![\theta]\!](h,s)} \big\downarrow & & \big\downarrow {\scriptstyle [\![\theta']\!](h,s)} \\
[\![\theta]\!] C' & \xrightarrow{\ [\![\pi \vdash e' : \theta \to \theta']\!] C'\eta\overline{C}\ } & [\![\theta']\!] C'
\end{array}
$$

es decir, la conmutatividad del diagrama anterior nos da la siguiente igualdad,

$$[\![\theta']\!](h,s) \circ [\![\pi \vdash e'\theta \to \theta']\!] C\eta\langle\rangle = [\![\pi \vdash e'\theta \to \theta']\!] C\eta\overline{C} \circ [\![\theta]\!](h,s)$$

luego usando esta igualdad obtenemos,

$$[\![\pi \vdash e' : \theta \to \theta']\!]C\eta\overline{C}([\![\theta]\!](h,s)([\![\pi \vdash e : \theta]\!]C\eta)) =$$
$$[\![\theta']\!](h,s)([\![\pi \vdash e'\theta \to \theta']\!]C\eta\langle\rangle([\![\pi \vdash e\theta]\!]\mathbf{C}\eta)) =$$
$$[\![\theta']\!](h,s)([\![\pi \vdash ee' : \theta']\!]C\eta)$$

- Supongamos $\pi \vdash \mathbf{rec}\ e : \theta$, desarrollemos ambos lados de la igualdad y luego probemos que los lados desarrollados son iguales, por un lado entonces tenemos,

$$[\![\theta]\!](h,s)([\![\pi \vdash \mathbf{rec}\ e : \theta]\!]C\eta) =$$
$$[\![\theta]\!](h,s)(\mathbf{Y}_{[\![\theta]\!]C}([\![\pi \vdash e : \theta \to \theta]\!]C\eta)\langle\rangle) =$$
$$[\![\theta]\!](h,s)(\bigsqcup_{i=0}^{\infty}(([\![\pi \vdash e : \theta \to \theta]\!]C\eta\langle\rangle)^i\bot_{\theta,C})) =$$
$$\bigsqcup_{i=0}^{\infty}([\![\theta]\!](h,s)(([\![\pi \vdash e : \theta \to \theta]\!]C\eta\langle\rangle)^i\bot_{\theta,C}))$$

y por el otro, notar que para este lado vamos a usar la hipótesis inductiva, tenemos

$$[\![\pi \vdash \mathbf{rec}\ e : \theta]\!]C'([\![\pi]\!](h,s)\eta) =$$
$$\mathbf{Y}_{[\![\theta]\!]C'}([\![\pi \vdash e : \theta \to \theta]\!]C'([\![\pi]\!](h,s)\eta))\langle\rangle =$$
$$\mathbf{Y}_{[\![\theta]\!]C'}([\![\theta \to \theta]\!](h,s)([\![\pi \vdash e : \theta \to \theta]\!]C\eta))\langle\rangle =$$
$$\bigsqcup_{i=0}^{\infty}(([\![\theta \to \theta]\!](h,s)([\![\pi \vdash e : \theta \to \theta]\!]C\eta)\langle\rangle)^i\bot_{\theta,C'}) =$$
$$\bigsqcup_{i=0}^{\infty}(([\![\theta \to \theta]\!](h,s)([\![\pi \vdash e : \theta \to \theta]\!]C\eta)\langle\rangle)^i([\![\theta]\!](h,s)\bot_{\theta,C})) =$$
$$\bigsqcup_{i=0}^{\infty}(([\![\pi \vdash e : \theta \to \theta]\!]C\eta\overline{C})^i([\![\theta]\!](h,s)\bot_{\theta,C}))$$

sobre esta ultima ecuación es importante notar que $\bot_{\theta,C'} = [\![\theta]\!](h,s)\bot_{\theta,C}$, por ser $[\![\theta]\!](h,s)$ una función estricta. Ahora bien, para probar la igualdad de las ecuaciones desarrolladas vamos a utilizar, como ya hicimos para el caso de la aplicación, el hecho de que $[\![\pi \vdash e : \theta \to \theta]\!]C\eta : [\![\theta \to \theta]\!]C$ es una transformación natural, luego el siguiente diagrama conmuta

$$\begin{array}{ccc} [\![\theta]\!]C & \xrightarrow{[\![\pi \vdash e : \theta \to \theta]\!]C\eta\langle\rangle} & [\![\theta]\!]C \\ {\scriptstyle [\![\theta]\!](h,s)}\downarrow & & \downarrow{\scriptstyle [\![\theta]\!](h,s)} \\ [\![\theta]\!]C' & \xrightarrow{[\![\pi \vdash e : \theta \to \theta]\!]C\eta\overline{C}} & [\![\theta]\!]C' \end{array}$$

podemos concluir entonces que

$$\bigsqcup_{i=0}^{\infty}([\![\theta]\!](h,s)(([\![\pi \vdash e : \theta \to \theta]\!]C\eta\langle\rangle)^{i}\bot_{\theta,C})) =$$
$$\bigsqcup_{i=0}^{\infty}((([\![\pi \vdash e : \theta \to \theta]\!]C\eta\overline{C})^{i}([\![\theta]\!](h,s)\bot_{\theta,C}))$$

- Supongamos $\pi \vdash \textbf{skip} : \textbf{comm}$,

$$[\![\textbf{comm}]\!](h,s)([\![\pi \vdash \textbf{skip}]\!]C\eta)\sigma' = s([\![\pi \vdash \textbf{skip}]\!]C\eta)\sigma' =$$
$$s(\lambda\sigma.\ \iota_{\uparrow}\sigma)\sigma' = (\lambda\widehat{\sigma}.\ (\lambda\sigma.\ \sigma +\!\!+\ \mathtt{tail}_{\overline{C}}\widehat{\sigma})_{\bot}((\lambda\sigma.\ \iota_{\uparrow}\sigma)(h\widehat{\sigma})))\sigma' =$$
$$(\lambda\sigma.\ \sigma +\!\!+\ \mathtt{tail}_{\overline{C}}\sigma')_{\bot}((\lambda\sigma.\iota_{\uparrow}\sigma)(\mathtt{head}_{C}\sigma')) =$$
$$\iota_{\uparrow}(\mathtt{head}_{C}\sigma' +\!\!+\ \mathtt{tail}_{\overline{C}}\sigma') = \iota_{\uparrow}\sigma' = [\![\pi \vdash \textbf{skip} : \textbf{comm}]\!]C'([\![\pi]\!](h,s)\eta)\sigma'$$

- Supongamos $\pi \vdash e;e' : \textbf{comm}$,

$$[\![\textbf{comm}]\!](h,s)([\![\pi \vdash e;e' : \textbf{comm}]\!]C\eta) =$$
$$[\![\textbf{comm}]\!](h,s)(([\![\pi \vdash e' : \textbf{comm}]\!]C\eta)_{\bot\!\!\!\bot} \circ [\![\pi \vdash e : \textbf{comm}]\!]C\eta) =$$
$$s(([\![\pi \vdash e' : \textbf{comm}]\!]C\eta)_{\bot\!\!\!\bot} \circ [\![\pi \vdash e : \textbf{comm}]\!]C\eta)$$

ahora usando la proposición 3 podemos reescribir de la siguiente manera,

$$s(([\![\pi \vdash e' : \textbf{comm}]\!]C\eta)_{\bot\!\!\!\bot} \circ [\![\pi \vdash e : \textbf{comm}]\!]C\eta) =$$
$$s([\![\pi \vdash e' : \textbf{comm}]\!]C\eta)_{\bot\!\!\!\bot} \circ s([\![\pi \vdash e : \textbf{comm}]\!]C\eta) =$$
$$[\![\textbf{comm}]\!](h,s)([\![\pi \vdash e' : \textbf{comm}]\!]C\eta)_{\bot\!\!\!\bot} \circ [\![\textbf{comm}]\!](h,s)([\![\pi \vdash e : \textbf{comm}]\!]C\eta)$$

usando ahora hipótesis inductiva dos veces obtenemos,

$$[\![\textbf{comm}]\!](h,s)([\![\pi \vdash e' : \textbf{comm}]\!]C\eta)_{\bot\!\!\!\bot} \circ [\![\textbf{comm}]\!](h,s)([\![\pi \vdash e : \textbf{comm}]\!]C\eta)$$
$$=$$
$$[\![\pi \vdash e' : \textbf{comm}]\!]C'([\![\pi]\!](h,s)\eta)_{\bot\!\!\!\bot} \circ [\![\pi \vdash e : \textbf{comm}]\!]C'([\![\pi]\!](h,s)\eta) =$$
$$[\![\pi \vdash e;e' : \textbf{comm}]\!]C'([\![\pi]\!](h,s)\eta)$$

- Supongamos $\pi \vdash \textbf{while}\ b\ \textbf{do}\ e : \textbf{comm}$, para este caso recordamos que $\sigma +\!\!+\ \overline{\sigma} = \sigma'$,

$$[\![\textbf{comm}]\!](h,s)([\![\pi \vdash \textbf{while}\ b\ \textbf{do}\ e : \textbf{comm}]\!]C\eta)\sigma' =$$
$$s([\![\pi \vdash \textbf{while}\ b\ \textbf{do}\ e : \textbf{comm}]\!]C\eta)\sigma' =$$
$$(\lambda\sigma.\ \sigma +\!\!+\ \overline{\sigma})_{\bot}([\![\pi \vdash \textbf{while}\ b\ \textbf{do}\ e : \textbf{comm}]\!]C\ \eta\ \sigma)$$

olvidémonos por un momento de $(\lambda\sigma.\ \sigma +\!\!+\ \overline{\sigma})_{\bot}$ y analicemos $[\![\pi \vdash \textbf{while}\ b\ \textbf{do}\ e : \textbf{comm}]\!]C\ \eta\ \sigma$, utilizando la definición de la ecuación

$$\begin{aligned}&[\![\pi \vdash \textbf{while}\ b\ \textbf{do}\ e : \textbf{comm}]\!]C\eta\sigma =\\ &\quad \mathbf{Y}_{[\![\textbf{comm}]\!]C}(\lambda c.\lambda\sigma.(\lambda b.\mathit{if}\ b\\ &\qquad \mathit{then}\ c_{\bot\!\!\!\bot}([\![\pi \vdash e : \textbf{comm}]\!]C\eta\sigma)\\ &\qquad \mathit{else}\ \iota_{\uparrow}\sigma)_{\bot\!\!\!\bot}([\![\pi \vdash b : \textbf{boolexp}]\!]C\eta\sigma))\sigma\end{aligned}$$

podemos notar que si el comando evalúa a un $\iota_{\uparrow}\widehat{\sigma}$, es porque existieron evaluaciones de $[\![\pi \vdash e : \textbf{comm}]\!]C\eta\sigma$ con resultados $\sigma_0, \sigma_1, \ldots, \sigma_n, \widehat{\sigma}$.

Además el resultado de la evaluación completa será , $(\lambda\sigma.\ \sigma + + \overline{\sigma})_{\perp}\iota_{\uparrow}\widehat{\sigma} = \iota_{\uparrow}(\widehat{\sigma} + + \overline{\sigma})$, es decir, solamente modificamos la parte del estado con forma C. Tomemos ahora el otro lado de la igualdad que queremos probar,

$$
\begin{aligned}
&[\![\pi \vdash \textbf{while}\ b\ \textbf{do}\ e : \textbf{comm}]\!]C'([\![\pi]\!](h,s)\eta)\sigma' = \\
&\quad \mathbf{Y}_{[\![\textbf{comm}]\!]C'}(\lambda c'.\lambda\sigma'.(\lambda b.\mathit{if}\ b \\
&\qquad \mathit{then}\ c'_{\perp\!\!\perp}([\![\pi \vdash e : \textbf{comm}]\!]C'([\![\pi]\!](h,s)\eta)\sigma') \\
&\qquad \mathit{else}\ \iota_{\uparrow}\sigma')_{\perp\!\!\perp}([\![\pi \vdash b : \textbf{boolexp}]\!]C'([\![\pi]\!](h,s)\eta)\sigma'))\sigma'
\end{aligned}
$$

luego aplicando hipótesis inductiva en los juicios de tipado de b y e y reescribiendo un poco las ecuaciones obtenemos,

$$
\begin{aligned}
&[\![\pi \vdash \textbf{while}\ b\ \textbf{do}\ e : \textbf{comm}]\!]C'([\![\pi]\!](h,s)\eta)\sigma' = \\
&\quad \mathbf{Y}_{[\![\textbf{comm}]\!]C'}(\lambda c'.\lambda\sigma'.(\lambda b.\mathit{if}\ b \\
&\qquad \mathit{then}\ c'_{\perp\!\!\perp}((\lambda\sigma.\ \sigma + + \overline{\sigma})_{\perp}[\![\pi \vdash e : \textbf{comm}]\!]C\eta\sigma) \\
&\qquad \mathit{else}\ \iota_{\uparrow}\sigma')_{\perp\!\!\perp}([\![\pi \vdash b : \textbf{boolexp}]\!]C\eta\sigma))\sigma'
\end{aligned}
$$

ahora vamos a suponer, como antes, que existieron evaluaciones de $[\![\pi \vdash e : \textbf{comm}]\!]C\eta\sigma$ con resultados $\sigma_0, \sigma_1, \ldots, \sigma_n, \widehat{\sigma}$, luego aplicar cada σ_i a $(\lambda\sigma.\ \sigma + + \overline{\sigma})$ nos genera la sucesión de resultados $\sigma_0 + + \overline{\sigma}, \sigma_1 + + \overline{\sigma}, \ldots, \sigma_n + + \overline{\sigma}, \widehat{\sigma} + + \overline{\sigma}$, por lo tanto el resultado de evaluar el comando completo es $\iota_{\uparrow}(\widehat{\sigma} + + \overline{\sigma})$.

- Supongamos $\pi \vdash a := e : \textbf{comm}$, empecemos desarrollando cada extremo de la igualdad que queremos probar,

$$
\begin{aligned}
&[\![\textbf{comm}]\!](h,s)([\![\pi \vdash a := e : \textbf{comm}]\!]C\eta)\sigma' = \\
&(\lambda\sigma.\ \sigma + + \overline{\sigma})_{\perp}([\![\pi \vdash a := e : \textbf{comm}]\!]C\eta\sigma) = \\
&(\lambda\sigma.\ \sigma + + \overline{\sigma})_{\perp}((\lambda v.\ [\![\pi \vdash a : \delta\textbf{acc}]\!]C\eta\ v\ \sigma)_{\perp\!\!\perp}([\![\pi \vdash e : \delta\textbf{exp}]\!]C\eta\sigma))
\end{aligned}
$$

$$
\begin{aligned}
&[\![\pi \vdash a := e : \textbf{comm}]\!]C'([\![\pi]\!](h,s)\eta)\sigma' = \\
&(\lambda v.\ [\![\pi \vdash a : \delta\textbf{acc}]\!]C'([\![\pi]\!](h,s)\eta)\ v\ \sigma')_{\perp\!\!\perp}\ ([\![\pi \vdash e : \delta\textbf{exp}]\!]C'([\![\pi]\!](h,s)\eta)\sigma')
\end{aligned}
$$

usando hipótesis inductiva dos veces obtenemos por un lado,

$$
\begin{aligned}
&[\![\pi \vdash a : \delta\textbf{acc}]\!]C'([\![\pi]\!](h,s)\eta)\ v\ \sigma' = \\
&[\![\delta\textbf{acc}]\!](h,s)([\![\pi \vdash a : \delta\textbf{acc}]\!]C\eta)v\ \sigma' = s([\![\pi \vdash a : \delta\textbf{acc}]\!]C\eta\ v)\sigma' = \\
&(\lambda\sigma.\sigma + + \overline{\sigma})_{\perp}([\![\pi \vdash a : \delta\textbf{acc}]\!]C\eta\ v\ \sigma)
\end{aligned}
$$

y por otro lado,

$$
\begin{aligned}
&[\![\pi \vdash e : \delta\textbf{exp}]\!]C'([\![\pi]\!](h,s)\eta)\sigma' = \\
&[\![\delta\textbf{exp}]\!](h,s)([\![\pi \vdash e : \delta\textbf{exp}]\!]C\eta)\sigma' = ([\![\pi \vdash e : \delta\textbf{exp}]\!]C\eta \circ h)\sigma' = \\
&[\![\pi \vdash e : \delta\textbf{exp}]\!]C\eta\sigma
\end{aligned}
$$

usando estos dos resultados que acabamos de deducir, pasando en limpio tenemos

$(\lambda \nu.\ [\![\pi \vdash a : \delta\mathbf{acc}]\!]C'([\![\pi]\!](h,s)\eta)\ \nu\ \sigma')_{\perp\!\!\perp}$
$([\![\pi \vdash e : \delta\mathbf{exp}]\!]C'([\![\pi]\!](h,s)\eta)\sigma') =$
$(\lambda \nu.\ (\lambda\sigma.\sigma +\!\!+\ \overline{\sigma})_{\perp}([\![\pi \vdash a : \delta\mathbf{acc}]\!]C\eta\ \nu\ \sigma))_{\perp\!\!\perp}\ ([\![\pi \vdash e : \delta\mathbf{exp}]\!]C\eta\sigma)$

luego podemos suponer $[\![\pi \vdash e : \delta\mathbf{exp}]\!]C\eta\sigma = \iota_{\uparrow}\nu$ distinto de $\perp$,

$(\lambda\sigma.\sigma +\!\!+\ \overline{\sigma})_{\perp}([\![\pi \vdash a : \delta\mathbf{acc}]\!]C\eta\ \nu\ \sigma) =$
$(\lambda\sigma.\sigma +\!\!+\ \overline{\sigma})_{\perp}(([\![\pi \vdash a : \delta\mathbf{acc}]\!]C\eta\ \nu\ \sigma)_{\perp\!\!\perp}([\![\pi \vdash e : \delta\mathbf{exp}]\!]C\eta\sigma)) =$
$[\![\mathbf{comm}]\!](h,s)([\![\pi \vdash a := e : \mathbf{comm}]\!]C\eta)\sigma'$

- Supongamos $\pi \vdash \lambda\iota_{\theta}.\ e : \theta \to \theta'$, tomemos $\widehat{C}$ objeto de $\mathbf{C}$ y un $z : [\![\theta]\!](C' +\!\!+\ \widehat{C})$

$[\![\theta \to \theta']\!](h,s)([\![\pi \vdash \lambda\iota_{\theta}.\ e : \theta \to \theta']\!]C\eta)\widehat{C}\ z =$
$[\![\pi \vdash \lambda\iota_{\theta}.\ e : \theta \to \theta']\!]C\eta(\overline{C} +\!\!+\ \widehat{C})\ z =$
$[\![\pi, \iota : \theta \vdash e : \theta']\!](C +\!\!+\ \overline{C} +\!\!+\ \widehat{C})[\ [\![\pi]\!](\overline{h}, \overline{s})\eta \mid \iota : z\ ] =$
$[\![\pi, \iota : \theta \vdash e : \theta']\!](C' +\!\!+\ \widehat{C})[\ [\![\pi]\!](\overline{h}, \overline{s})\eta \mid \iota : z\ ]$

ahora bien, empezamos notando que el operador $+\!\!+$ es asociativo por lo tanto $(C +\!\!+\ (\overline{C} +\!\!+\ \widehat{C})) = (C' +\!\!+\ \widehat{C})$, luego además $(\overline{h}, \overline{s}) : C \longrightarrow (C +\!\!+\ \overline{C} +\!\!+\ \widehat{C})$, y por lo tanto $(\overline{h}, \overline{s}) : C \longrightarrow (C' +\!\!+\ \widehat{C})$.

$[\![\pi \vdash \lambda\iota_{\theta}.\ e : \theta \to \theta']\!]C'([\![\pi]\!](h,s)\eta)\widehat{C}\ z =$
$[\![\pi, \iota : \theta \vdash e : \theta']\!](C' +\!\!+\ \widehat{C})[\ [\![\pi]\!](h', s')([\![\pi]\!](h,s)\eta) \mid \iota : z\ ]$

donde $(h', s') : C' \longrightarrow (C' +\!\!+\ \widehat{C})$, ahora si prestamos atención a las dos expresiones finales que nos quedaron, nos damos cuenta que ya casi estaríamos si no fuera por la forma en que acomodamos el ambiente, el plan entonces será probar que $(\overline{h}, \overline{s}) = (h \circ h', s' \circ s)$ donde por definición $(h \circ h', s' \circ s) = (h', s') \circ (h, s)$, luego usando que $[\![\pi]\!]$ es funtor podemos concluir,

$$[\![\pi]\!](\overline{h}, \overline{s}) = [\![\pi]\!](h', s') \circ [\![\pi]\!](h, s)$$

Empecemos probando que $\overline{h} = h \circ h'$, tomamos un estado $\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \widehat{\sigma}$ con forma $C' +\!\!+\ \widehat{C}$

$\overline{h} : (C' +\!\!+\ \widehat{C}) \to C$
$\overline{h}(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \widehat{\sigma}) = head_C(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \widehat{\sigma}) = \sigma$

$h : C' \to C$
$h' : (C' +\!\!+\ \widehat{C}) \to C'$
$(h \circ h')(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \widehat{\sigma}) = head_C(head_{C'}(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \widehat{\sigma})) =$
$head_C(\sigma +\!\!+\ \overline{\sigma}) = \sigma$

probemos ahora $\overline{s} = s' \circ s$, tomemos $c : C \to C_{\perp}$ y suponemos $c\sigma = \iota_{\uparrow}\tilde{\sigma}$

$\overline{s} : (C \to C_\perp) \to ((C' +\!\!+ \widehat{C}) \to (C' +\!\!+ \widehat{C})_\perp)$
$\overline{s}\ c\ (\sigma +\!\!+ \overline{\sigma} +\!\!+ \widehat{\sigma}) = (\lambda\sigma.\ \sigma +\!\!+ (\overline{\sigma} +\!\!+ \widehat{\sigma}))_\perp(c\sigma) =$
$\iota_\uparrow(\tilde{\sigma} +\!\!+ \overline{\sigma} +\!\!+ \widehat{\sigma})$

$s : (C \to C_\perp) \to (C' \to C'_\perp)$
$s' : (C' \to C'_\perp) \to ((C' +\!\!+ \widehat{C}) \to (C' +\!\!+ \widehat{C})_\perp)$
$(s' \circ s)\ c\ (\sigma +\!\!+ \overline{\sigma} +\!\!+ \widehat{\sigma}) = s'(s\ c)(\sigma +\!\!+ \overline{\sigma} +\!\!+ \widehat{\sigma}) =$
$(\lambda\sigma.\ \sigma +\!\!+ \widehat{\sigma})_\perp(s\ c\ (\sigma +\!\!+ \overline{\sigma})) =$
$(\lambda\sigma.\ \sigma +\!\!+ \widehat{\sigma})_\perp((\lambda\sigma.\ \sigma +\!\!+ \overline{\sigma})_\perp(c\sigma)) =$
$(\lambda\sigma.\ \sigma +\!\!+ \widehat{\sigma})_\perp(\iota_\uparrow(\tilde{\sigma} +\!\!+ \overline{\sigma})) = \iota_\uparrow(\tilde{\sigma} +\!\!+ \overline{\sigma} +\!\!+ \widehat{\sigma})$

por lo tanto $(\overline{h}, \overline{s}) = (h \circ h', s' \circ s)$.

- Supongamos $\pi \vdash \textbf{new}\ \delta\textbf{var}\ \iota := ei\ \textbf{in}\ c : \textbf{comm}$, la estrategia para este juicio de tipado será, resolver por completo un lado haciendo un par de suposiciones hasta llegar a un estado y luego tomar el lado restante para llegar, usando las suposiciones, a este mismo estado. Para esto entonces suponemos dos cosas relevantes,
  - $[\![\pi \vdash ei : \delta\textbf{exp}]\!]C\eta\sigma = \iota_\uparrow v$
  - $[\![\pi, \iota : \delta\textbf{var} \vdash c : \textbf{comm}]\!](C +\!\!+ \langle S_\delta\rangle)$
    $[\ [\![\pi]\!](\overline{h}, \overline{s})\eta \mid \iota : \langle a, e\rangle\ ](\sigma +\!\!+ \langle v\rangle) = \iota_\uparrow(\tilde{\sigma} +\!\!+ \langle\tilde{v}\rangle)$

$[\![\textbf{comm}]\!](h, s)([\![\pi \vdash \textbf{new}\ \delta\textbf{var}\ \iota := ei\ \textbf{in}\ c : \textbf{comm}]\!]C\eta)\sigma' =$
$s(\lambda\sigma.\ H_\perp((\lambda v.$
$[\![\pi, \iota : \delta\textbf{var} \vdash c : \textbf{comm}]\!](C +\!\!+ \langle S_\delta\rangle)$
$[\ [\![\pi]\!](\overline{h}, \overline{s})\eta \mid \iota : \langle a, e\rangle\ ](\sigma +\!\!+ \langle v\rangle))_{\perp\!\!\perp}$
$([\![\pi \vdash ei : \delta\textbf{exp}]\!]C\eta\sigma)))\sigma' =$
$(\lambda\sigma.\ \sigma +\!\!+ \overline{\sigma})_\perp(head_{C\perp}((\lambda v.$
$[\![\pi, \iota : \delta\textbf{var} \vdash c : \textbf{comm}]\!](C +\!\!+ \langle S_\delta\rangle)$
$[\ [\![\pi]\!](\overline{h}, \overline{s})\eta \mid \iota : \langle a, e\rangle\ ](\sigma +\!\!+ \langle v\rangle))_{\perp\!\!\perp}$
$([\![\pi \vdash ei : \delta\textbf{exp}]\!]C\eta\sigma))) =$
$(\lambda\sigma.\ \sigma +\!\!+ \overline{\sigma})_\perp(head_{C\perp}($
$[\![\pi, \iota : \delta\textbf{var} \vdash c : \textbf{comm}]\!](C +\!\!+ \langle S_\delta\rangle)$
$[\ [\![\pi]\!](\overline{h}, \overline{s})\eta \mid \iota : \langle a, e\rangle\ ](\sigma +\!\!+ \langle v\rangle))) =$
$(\lambda\sigma.\ \sigma +\!\!+ \overline{\sigma})_\perp(head_{C\perp}(\iota_\uparrow(\tilde{\sigma} +\!\!+ \langle\tilde{v}\rangle))) =$
$((\lambda\sigma.\ \sigma +\!\!+ \overline{\sigma}) \circ head_C)_\perp(\iota_\uparrow(\tilde{\sigma} +\!\!+ \langle\tilde{v}\rangle)) =$
$\iota_\uparrow(\tilde{\sigma} +\!\!+ \overline{\sigma})$

detallando ahora los tipos de algunos componentes que van a ser relevantes mas adelante tenemos,

$a : S_\delta \to (C +\!\!+ \langle S_\delta\rangle) \to (C +\!\!+ \langle S_\delta\rangle)_\perp$
$e : (C +\!\!+ \langle S_\delta\rangle) \to (S_\delta)_\perp$
$(\overline{h}, \overline{s}) : C \longrightarrow (C +\!\!+ \langle S_\delta\rangle)$

ahora continuemos con el otro extremo de la igualdad que queremos probar, teniendo siempre en cuenta las suposiciones que hicimos al

principio de este caso

$$\begin{array}{l}
[\![\pi \vdash \textbf{new}\ \delta\textbf{var}\ \iota := ei\ \textbf{in}\ c : \textbf{comm}]\!]C'([\![\pi]\!](h,s)\eta)\sigma' = \\
head_{C'\perp}((\lambda\nu. \\
\quad [\![\pi, \iota : \delta\textbf{var} \vdash c : \textbf{comm}]\!](C' +\!\!+ \langle S_\delta\rangle) \\
\qquad\qquad [\ [\![\pi]\!](h', s')([\![\pi]\!](h,s)\eta) \mid \iota : \langle a', e'\rangle\ ](\sigma' +\!\!+ \langle\nu\rangle))_{\perp\!\!\perp} \\
\qquad\qquad\qquad ([\![\pi \vdash ei : \delta\textbf{exp}]\!]C'([\![\pi]\!](h,s)\eta)\sigma'))
\end{array}$$

$$\begin{array}{l}
\text{donde } a' : S_\delta \to (C' +\!\!+ \langle S_\delta\rangle) \to (C' +\!\!+ \langle S_\delta\rangle)_\perp \\
e' : (C' +\!\!+ \langle S_\delta\rangle) \to (S_\delta)_\perp \\
(h', s') : C' \longrightarrow (C' +\!\!+ \langle S_\delta\rangle)
\end{array}$$

si nos enfocamos en la expresión $[\![\pi \vdash ei : \delta\textbf{exp}]\!]C'([\![\pi]\!](h,s)\eta)\sigma'$ nos damos cuenta que podemos aplicar hipótesis inductiva, de manera tal que nos quede

$$\begin{array}{l}
[\![\pi \vdash ei : \delta\textbf{exp}]\!]C'([\![\pi]\!](h,s)\eta)\sigma' = \\
[\![\delta\textbf{exp}]\!](h,s)([\![\pi \vdash ei : \delta\textbf{exp}]\!]C\eta)\sigma' = \\
([\![\pi \vdash ei : \delta\textbf{exp}]\!]C\eta \circ head_C)\sigma' = \\
[\![\pi \vdash ei : \delta\textbf{exp}]\!]C\eta\sigma = \iota_\uparrow\nu
\end{array}$$

podemos utilizar este resultado entonces para ir reescribiendo

$$\begin{array}{l}
head_{C'\perp}((\lambda\nu. \\
\quad [\![\pi, \iota : \delta\textbf{var} \vdash c : \textbf{comm}]\!](C' +\!\!+ \langle S_\delta\rangle) \\
\qquad\qquad [\ [\![\pi]\!](h', s')([\![\pi]\!](h,s)\eta) \mid \iota : \langle a', e'\rangle\ ](\sigma' +\!\!+ \langle\nu\rangle))_{\perp\!\!\perp} \\
\qquad\qquad\qquad ([\![\pi \vdash ei : \delta\textbf{exp}]\!]C'([\![\pi]\!](h,s)\eta)\sigma')) = \\
head_{C'\perp}([\![\pi, \iota : \delta\textbf{var} \vdash c : \textbf{comm}]\!](C' +\!\!+ \langle S_\delta\rangle) \\
\qquad\qquad [\ [\![\pi]\!](h', s')([\![\pi]\!](h,s)\eta) \mid \iota : \langle a', e'\rangle\ ](\sigma' +\!\!+ \langle\nu\rangle))
\end{array}$$

llegado este punto, para continuar, lo ideal sería poder hacer algo parecido a lo que hicimos hace un instante para evaluar el juicio de tipado de ei el problema que surge es que no podemos aplicar hipótesis inductiva porque nuestro ambiente no tiene la forma correcta, es decir, nosotros quisiéramos que el ambiente fuera algo de la forma $[\![\pi, \iota : \delta\textbf{var}]\!](\widehat{h}, \widehat{s})\widehat{\eta}$, pero además no con cualquiera $\widehat{\eta}$, sino que este sea $[\ [\![\pi]\!](\overline{h}, \overline{s})\eta \mid \iota : \langle a, e\rangle\ ]$. En resumen, quisiéramos que la siguiente igualdad valga

$$\begin{array}{c}
[\![\pi, \iota : \delta\textbf{var}]\!](\widehat{h}, \widehat{s})([\ [\![\pi]\!](\overline{h}, \overline{s})\eta \mid \iota : \langle a, e\rangle\ ]) \\
= \\
[\ [\![\pi]\!](h', s')([\![\pi]\!](h,s)\eta) \mid \iota : \langle a', e'\rangle\ ]
\end{array}$$

donde $(\widehat{h}, \widehat{s}) : (C +\!\!+ \langle S_\delta\rangle) \longrightarrow (C' +\!\!+ \langle S_\delta\rangle)$
así que vamos a hacer lo siguiente, vamos a suponer que esta igualdad vale, completar la prueba y para finalizar probaremos la igualdad. Pero antes, nos falta el detalle importante sobre cómo son las funciones de la flecha $(\widehat{h}, \widehat{s})$, es decir, en general para cualquier flecha común podemos saber quiénes son las funciones viendo el

tipo de la flecha, pero en particular el tipo de esta flecha no es una simple extensión, presentemos la definición de cada una

$$
\begin{array}{l}
\widehat{h} : (C' \mathbin{+\!\!+} \langle S_\delta \rangle) \rightarrow (C \mathbin{+\!\!+} \langle S_\delta \rangle) \\
\widehat{h}\ \sigma = \mathtt{head}_C(\mathtt{head}_{C'}\sigma) \mathbin{+\!\!+} \mathtt{tail}_{\langle S_\delta \rangle}\sigma \\
\widehat{s} : ((C \mathbin{+\!\!+} \langle S_\delta \rangle) \rightarrow (C \mathbin{+\!\!+} \langle S_\delta \rangle)_\perp) \rightarrow ((C' \mathbin{+\!\!+} \langle S_\delta \rangle) \rightarrow (C' \mathbin{+\!\!+} \langle S_\delta \rangle)_\perp) \\
\widehat{s}\ f\ \sigma = (\lambda\sigma_\nu.\ (\overline{h}\sigma_\nu) \mathbin{+\!\!+} \mathtt{tail}_{\overline{C}}(\mathtt{tail}_{C'}\sigma) \mathbin{+\!\!+} \mathtt{tail}_{\langle S_\delta \rangle}\sigma_\nu)_\perp(f(\widehat{h}\sigma))
\end{array}
$$

Entonces, reescribiendo utilizando la igualdad tenemos

$$
\begin{array}{l}
\mathtt{head}_{C'\perp}(\llbracket \pi, \iota : \delta\mathbf{var} \vdash c : \mathbf{comm} \rrbracket(C' \mathbin{+\!\!+} \langle S_\delta \rangle) \\
\qquad\qquad\quad [\, \llbracket \pi \rrbracket(h', s')(\llbracket \pi \rrbracket(h, s)\eta) \mid \iota : \langle a', e' \rangle\,](\sigma' \mathbin{+\!\!+} \langle \nu \rangle)) = \\
\mathtt{head}_{C'\perp}(\llbracket \pi, \iota : \delta\mathbf{var} \vdash c : \mathbf{comm} \rrbracket(C' \mathbin{+\!\!+} \langle S_\delta \rangle) \\
\qquad\qquad (\llbracket \pi, \iota : \delta\mathbf{var} \rrbracket(\widehat{h}, \widehat{s})([\, \llbracket \pi \rrbracket(\overline{h}, \overline{s})\eta \mid \iota : \langle a, e \rangle\,]))(\sigma' \mathbin{+\!\!+} \langle \nu \rangle)) = \\
\mathtt{head}_{C'\perp}(\llbracket \mathbf{comm} \rrbracket(\widehat{h}, \widehat{s})(\llbracket \pi, \iota : \delta\mathbf{var} \vdash c : \mathbf{comm} \rrbracket(C \mathbin{+\!\!+} \langle S_\delta \rangle) \\
\qquad\qquad\quad [\, \llbracket \pi \rrbracket(\overline{h}, \overline{s})\eta \mid \iota : \langle a, e \rangle\,])(\sigma' \mathbin{+\!\!+} \langle \nu \rangle)) = \\
\mathtt{head}_{C'\perp}(\widehat{s}(\llbracket \pi, \iota : \delta\mathbf{var} \vdash c : \mathbf{comm} \rrbracket(C \mathbin{+\!\!+} \langle S_\delta \rangle) \\
\qquad\qquad\qquad\qquad [\, \llbracket \pi \rrbracket(\overline{h}, \overline{s})\eta \mid \iota : \langle a, e \rangle\,])(\sigma' \mathbin{+\!\!+} \langle \nu \rangle)) = \\
\mathtt{head}_{C'\perp}(\\
(\lambda\sigma_\nu.\ (\overline{h}\sigma_\nu) \mathbin{+\!\!+} \mathtt{tail}_{\overline{C}}(\mathtt{tail}_{C'}(\sigma' \mathbin{+\!\!+} \langle \nu \rangle)) \mathbin{+\!\!+} \mathtt{tail}_{\langle S_\delta \rangle}\sigma_\nu)_\perp \\
\qquad (\llbracket \pi, \iota : \delta\mathbf{var} \vdash c : \mathbf{comm} \rrbracket(C \mathbin{+\!\!+} \langle S_\delta \rangle) \\
\qquad\qquad\quad [\, \llbracket \pi \rrbracket(\overline{h}, \overline{s})\eta \mid \iota : \langle a, e \rangle\,])(\widehat{h}(\sigma' \mathbin{+\!\!+} \langle \nu \rangle)))) = \\
\mathtt{head}_{C'\perp}(\\
(\lambda\sigma_\nu.\ (\overline{h}\sigma_\nu) \mathbin{+\!\!+} \mathtt{tail}_{\overline{C}}(\mathtt{tail}_{C'}(\sigma' \mathbin{+\!\!+} \langle \nu \rangle)) \mathbin{+\!\!+} \mathtt{tail}_{\langle S_\delta \rangle}\sigma_\nu)_\perp \\
\qquad (\llbracket \pi, \iota : \delta\mathbf{var} \vdash c : \mathbf{comm} \rrbracket(C \mathbin{+\!\!+} \langle S_\delta \rangle) \\
\qquad\qquad\quad [\, \llbracket \pi \rrbracket(\overline{h}, \overline{s})\eta \mid \iota : \langle a, e \rangle\,])(\sigma \mathbin{+\!\!+} \langle \nu \rangle)))
\end{array}
$$

y por fin estamos en condiciones de reescribir usando la segunda suposición sobre el juicio de tipado de c

$$
\begin{array}{l}
\mathtt{head}_{C'\perp}(\\
(\lambda\sigma_\nu.\ (\overline{h}\sigma_\nu) \mathbin{+\!\!+} \mathtt{tail}_{\overline{C}}(\mathtt{tail}_{C'}(\sigma' \mathbin{+\!\!+} \langle \nu \rangle)) \mathbin{+\!\!+} \mathtt{tail}_{\langle S_\delta \rangle}\sigma_\nu)_\perp \\
\qquad (\llbracket \pi, \iota : \delta\mathbf{var} \vdash c : \mathbf{comm} \rrbracket(C \mathbin{+\!\!+} \langle S_\delta \rangle) \\
\qquad\qquad\quad [\, \llbracket \pi \rrbracket(\overline{h}, \overline{s})\eta \mid \iota : \langle a, e \rangle\,])(\sigma \mathbin{+\!\!+} \langle \nu \rangle))) = \\
\mathtt{head}_{C'\perp}(\\
(\lambda\sigma_\nu.\ (\overline{h}\sigma_\nu) \mathbin{+\!\!+} \mathtt{tail}_{\overline{C}}(\mathtt{tail}_{C'}(\sigma' \mathbin{+\!\!+} \langle \nu \rangle)) \mathbin{+\!\!+} \mathtt{tail}_{\langle S_\delta \rangle}\sigma_\nu)_\perp \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\iota_\uparrow(\tilde{\sigma} \mathbin{+\!\!+} \langle \tilde{\nu} \rangle))) = \\
\mathtt{head}_{C'\perp}(\\
(\lambda\sigma_\nu.\ (\overline{h}\sigma_\nu) \mathbin{+\!\!+} \mathtt{tail}_{\overline{C}}(\mathtt{tail}_{C'}(\sigma' \mathbin{+\!\!+} \langle \nu \rangle)) \mathbin{+\!\!+} \mathtt{tail}_{\langle S_\delta \rangle}\sigma_\nu)_\perp \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\iota_\uparrow(\tilde{\sigma} \mathbin{+\!\!+} \langle \tilde{\nu} \rangle))) = \\
\mathtt{head}_{C'\perp}(\ (\lambda\sigma_\nu.\ (\overline{h}\sigma_\nu) \mathbin{+\!\!+} \overline{\sigma} \mathbin{+\!\!+} \mathtt{tail}_{\langle S_\delta \rangle}\sigma_\nu)_\perp\ (\iota_\uparrow(\tilde{\sigma} \mathbin{+\!\!+} \langle \tilde{\nu} \rangle))) = \\
(\mathtt{head}_{C'} \circ (\lambda\sigma_\nu.\ (\overline{h}\sigma_\nu) \mathbin{+\!\!+} \overline{\sigma} \mathbin{+\!\!+} \mathtt{tail}_{\langle S_\delta \rangle}\sigma_\nu))_\perp\ (\iota_\uparrow(\tilde{\sigma} \mathbin{+\!\!+} \langle \tilde{\nu} \rangle)) = \\
\iota_\uparrow(\mathtt{head}_{C'}(\tilde{\sigma} \mathbin{+\!\!+} \overline{\sigma} \mathbin{+\!\!+} \langle \tilde{\nu} \rangle)) = \iota_\uparrow(\tilde{\sigma} \mathbin{+\!\!+} \overline{\sigma})
\end{array}
$$

por lo tanto, hemos probado la igualdad para el caso del **new**δ**var**. Nos resta probar entonces la suposición que hicimos sobre los ambientes, recordando deberíamos probar

$$[\![\pi,\iota:\delta\mathbf{var}]\!](\widehat{h},\widehat{s})([\ [\![\pi]\!](\overline{h},\overline{s})\eta \mid \iota:\langle a,e\rangle\ ])$$
$$=$$
$$[\ [\![\pi]\!](h',s')([\![\pi]\!](h,s)\eta) \mid \iota:\langle a',e'\rangle\ ]$$

Supongamos un κ identificador y dos valores ν y x en S_δ, cualesquiera

- Sí $\kappa = \iota$,

  $([\![\pi,\iota:\delta\mathbf{var}]\!](\widehat{h},\widehat{s})([\ [\![\pi]\!](\overline{h},\overline{s})\eta \mid \iota:\langle a,e\rangle\ ]))\kappa =$
  $[\![(\pi,\iota:\delta\mathbf{var})\kappa]\!](\widehat{h},\widehat{s})\langle a,e\rangle =$
  $\langle[\![\delta\mathbf{var}]\!](\widehat{h},\widehat{s})a, [\![\delta\mathbf{var}]\!](\widehat{h},\widehat{s})e\rangle$

  $[\ [\![\pi]\!](h',s')([\![\pi]\!](h,s)\eta) \mid \iota:\langle a',e'\rangle\ ]\kappa = \langle a',e'\rangle$

  probemos ahora que $a' = [\![\delta\mathbf{var}]\!](\widehat{h},\widehat{s})a$ y $e' = [\![\delta\mathbf{var}]\!](\widehat{h},\widehat{s})e$

  $[\![\delta\mathbf{var}]\!](\widehat{h},\widehat{s})\ e\ (\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle) = e\ (\widehat{h}(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle)) =$
  $\iota_\uparrow(last_{S_\delta}(\sigma +\!\!+\ \langle\nu\rangle)) = \iota_\uparrow\ \nu = \iota_\uparrow(last_{S_\delta}(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle)) =$
  $e'(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle)$

  $[\![\delta\mathbf{var}]\!](\widehat{h},\widehat{s})\ a\ x\ (\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle) = \widehat{s}(a\ x)(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle) =$
  $(\lambda\sigma_x.\ \iota_\uparrow((\overline{h}\sigma_x) +\!\!+\ tail_{\overline{C}}(tail_{C'}(\sigma' +\!\!+\ \langle x\rangle)) +\!\!+\ tail_{\langle S_\delta\rangle}\sigma_x))_{\perp\!\!\perp}$
  $((a\ x)(\widehat{h}(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle))) =$
  $(\lambda\sigma_x.\ \iota_\uparrow((\overline{h}\sigma_x) +\!\!+\ tail_{\overline{C}}(tail_{C'}(\sigma' +\!\!+\ \langle x\rangle)) +\!\!+\ tail_{\langle S_\delta\rangle}\sigma_x))_{\perp\!\!\perp}$
  $(\lambda\sigma.\ \iota_\uparrow\ head_C\sigma +\!\!+\ \langle x\rangle)(\sigma +\!\!+\ \langle\nu\rangle) =$
  $(\lambda\sigma_x.\ \iota_\uparrow((\overline{h}\sigma_x) +\!\!+\ tail_{\overline{C}}(tail_{C'}(\sigma' +\!\!+\ \langle x\rangle)) +\!\!+\ tail_{\langle S_\delta\rangle}\sigma_x))_{\perp\!\!\perp}$
  $(\iota_\uparrow(\sigma +\!\!+\ \langle x\rangle)) =$
  $\iota_\uparrow(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle x\rangle) =$
  $\iota_\uparrow(head_{C'}(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle) +\!\!+\ \langle x\rangle) = a'\ x\ (\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle)$

- Sí $\kappa \neq \iota$,

  $([\![\pi,\iota:\delta\mathbf{var}]\!](\widehat{h},\widehat{s})([\ [\![\pi]\!](\overline{h},\overline{s})\eta \mid \iota:\langle a,e\rangle\ ]))\kappa = [\![\pi,\iota:\delta\mathbf{var}]\!](\widehat{h},\widehat{s})([\![\pi]\!](\overline{h},\overline{s})\eta)\kappa$

  $[\ [\![\pi]\!](h',s')([\![\pi]\!](h,s)\eta) \mid \iota:\langle a',e'\rangle\ ]\kappa = [\![\pi]\!](h',s')([\![\pi]\!](h,s)\eta)\kappa$

  para probar esto vamos a utilizar que $[\![\pi]\!]$ es funtor como ya hemos hecho para el caso de la abstracción lambda. Es decir, deberíamos probar que $(\overline{h}\circ\widehat{h},\widehat{s}\circ\overline{s}) = (h\circ h', s'\circ s)$. Supongamos $f : C \to C_\perp$

  $(\overline{h}\circ\widehat{h})(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle) = \overline{h}(\widehat{h}(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle)) = \overline{h}(\sigma +\!\!+\ \langle\nu\rangle) = \sigma$

  $(h\circ h')(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle) = h(h'(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle)) = h(\sigma +\!\!+\ \overline{\sigma}) = \sigma$

  para la segunda función de la flecha, supongamos además $f(\overline{h}(\widehat{h}(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle))) = \iota\tilde{\sigma}$ y empecemos notando que
  $f(\overline{h}(\widehat{h}(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle))) = f(h(h'(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle\nu\rangle))) = f(h(\sigma +\!\!+\ \overline{\sigma}))$

$$
\begin{aligned}
&(\widehat{s} \circ \overline{s})\ f\ (\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle v \rangle) = \widehat{s}(\overline{s}\ f)\ (\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle v \rangle) = \\
&(\lambda\sigma_v.\ (\overline{h}\sigma_v) +\!\!+\ \mathtt{tail}_{\overline{C}}(\mathtt{tail}_{C'}(\sigma' +\!\!+\ \langle v \rangle)) +\!\!+\ \mathtt{tail}_{\langle S_\delta \rangle}\sigma_v)_\perp \\
&\qquad\qquad (\overline{s}\ f(\widehat{h}(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle v \rangle))) = \\
&(\lambda\sigma_v.\ (\overline{h}\sigma_v) +\!\!+\ \mathtt{tail}_{\overline{C}}(\mathtt{tail}_{C'}(\sigma' +\!\!+\ \langle v \rangle)) +\!\!+\ \mathtt{tail}_{\langle S_\delta \rangle}\sigma_v)_\perp \\
&\qquad\qquad ((\lambda\sigma.\ \sigma +\!\!+\ \langle v \rangle)_\perp(f(\overline{h}(\widehat{h}(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle v \rangle))))) = \\
&(\lambda\sigma_v.\ (\overline{h}\sigma_v) +\!\!+\ \mathtt{tail}_{\overline{C}}(\mathtt{tail}_{C'}(\sigma' +\!\!+\ \langle v \rangle)) +\!\!+\ \mathtt{tail}_{\langle S_\delta \rangle}\sigma_v)_\perp \\
&\qquad\qquad ((\lambda\sigma.\ \sigma +\!\!+\ \langle v \rangle)_\perp(\iota_\uparrow\tilde{\sigma})) = \\
&(\lambda\sigma_v.\ (\overline{h}\sigma_v) +\!\!+\ \mathtt{tail}_{\overline{C}}(\mathtt{tail}_{C'}(\sigma' +\!\!+\ \langle v \rangle)) +\!\!+\ \mathtt{tail}_{\langle S_\delta \rangle}\sigma_v)_\perp \\
&\qquad\qquad (\iota_\uparrow(\tilde{\sigma} +\!\!+\ \langle v \rangle)) = \\
&\iota_\uparrow(\tilde{\sigma} +\!\!+\ \overline{\sigma} +\!\!+\ \langle v \rangle)
\end{aligned}
$$

$$
\begin{aligned}
&(s' \circ s)\ f\ (\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle v \rangle) = (s'(s\ f)\ (\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle v \rangle) = \\
&(\lambda\sigma.\ \sigma +\!\!+\ \langle v \rangle)_\perp((s\ f)(h'(\sigma +\!\!+\ \overline{\sigma} +\!\!+\ \langle v \rangle))) = \\
&(\lambda\sigma.\ \sigma +\!\!+\ \langle v \rangle)_\perp((\lambda\sigma.\ \sigma +\!\!+\ \overline{\sigma})_\perp(f\ (h(\sigma +\!\!+\ \overline{\sigma})))) = \\
&(\lambda\sigma.\ \sigma +\!\!+\ \langle v \rangle)_\perp(\iota_\uparrow(\tilde{\sigma} +\!\!+\ \overline{\sigma})) = \iota_\uparrow(\tilde{\sigma} +\!\!+\ \overline{\sigma} +\!\!+\ \langle v \rangle)
\end{aligned}
$$

Finalmente, con la igualdad entre estas flechas que acabamos de probar, hemos completado la prueba de naturalidad para el comando **newδvar** y además hemos completado la prueba de naturalidad para nuestro lenguaje λ^{like}.

□

4.5. Implementación en Idris

La implementación del lenguaje se encuentra en:
`https://github.com/alexgadea/thesis/tree/master/Prototypes/Idris/LambdaLike`

Sintaxis de los tipos de dato básicos de λ^{like}

```
data DataType = IntDT | RealDT | BoolDT
```

Semántica de los tipos de dato básicos de λ^{like}

```
evalDTy : DataType -> Type
evalDTy IntDT  = Int
evalDTy RealDT = Float
evalDTy BoolDT = Bool
```

Sintaxis de los tipos de las frases de λ^{like}

```
data PhraseType = IntExp | RealExp | BoolExp
                | IntAcc | RealAcc | BoolAcc
                | IntVar | RealVar | BoolVar
                | PhraseType :-> PhraseType
                | Comm

-- Conversión entre tipos
dtTOacc : DataType -> PhraseType
dtTOacc IntDT  = IntAcc
dtTOacc RealDT = RealAcc
dtTOacc BoolDT = BoolAcc
```

```
dtTOexp : DataType -> PhraseType
dtTOexp IntDT  = IntExp
dtTOexp RealDT = RealExp
dtTOexp BoolDT = BoolExp

dtTOvar : DataType -> PhraseType
dtTOvar IntDT  = IntVar
dtTOvar RealDT = RealVar
dtTOvar BoolDT = BoolVar

evalTyArgs : PhraseType -> Type
evalTyArgs IntExp  = Int
evalTyArgs RealExp = Float
evalTyArgs BoolExp = Bool

ptToDt : PhraseType -> DataType
ptToDt IntExp  = IntDT
ptToDt RealExp = RealDT
ptToDt BoolExp = BoolDT
```

Semántica de los tipos de las frases de λ^{like}

```
-- Semántica para los phrase types aplicada a objetos con forma.
evalTyO : PhraseType -> Shp -> Type
evalTyO IntExp            C = shapes C -> Int
evalTyO RealExp           C = shapes C -> Float
evalTyO BoolExp           C = shapes C -> Bool
evalTyO IntAcc            C = Int -> evalTyO Comm C
evalTyO RealAcc           C = Float -> evalTyO Comm C
evalTyO BoolAcc           C = Bool -> evalTyO Comm C
evalTyO IntVar            C = (evalTyO IntAcc  C, evalTyO IntExp C)
evalTyO RealVar           C = (evalTyO RealAcc C, evalTyO RealExp C)
evalTyO BoolVar           C = (evalTyO BoolAcc C, evalTyO BoolExp C)
evalTyO Comm              C = shapes C -> shapes C
evalTyO (Theta :-> Theta') C = (C':Shp) -> evalTyO Theta (C ++ C') -> evalTyO Theta' (C ++ C')

-- Transforma la semántica de un tipo en un cierto objeto c, en la semántica
-- de un tipo en otro objeto c', pero mientras c=c' .
convEvTyCtx : {Pt : PhraseType} -> (C : Shp) -> (C' : Shp) -> C = C' ->
              evalTyO Pt C -> evalTyO Pt C'
convEvTyCtx c c refl eval = eval

-- Semántica para los phrase types aplicada a morfismos entre objetos.
evalTyM : {C:Shp} -> {C':Shp} -> (t:PhraseType) -> C <= C' -> evalTyO t C -> evalTyO t C'
evalTyM IntExp            (morp (h,s,_)) e        = e . h
evalTyM RealExp           (morp (h,s,_)) e        = e . h
evalTyM BoolExp           (morp (h,s,_)) e        = e . h
evalTyM IntAcc            (morp (h,s,_)) a        = s . a
evalTyM RealAcc           (morp (h,s,_)) a        = s . a
evalTyM BoolAcc           (morp (h,s,_)) a        = s . a
evalTyM IntVar            (morp (h,s,_)) (a,e)    = (s . a, e . h)
evalTyM BoolVar           (morp (h,s,_)) (a,e)    = (s . a, e . h)
evalTyM RealVar           (morp (h,s,_)) (a,e)    = (s . a, e . h)
evalTyM Comm              (morp (h,s,_)) c        = s c
evalTyM {C=c} {C'=c'} (Theta :-> Theta') (morp (h,s,(c1 ** p))) f =
                    \c'' => \v => contract c'' (f (c1++c'') (expand c'' v))
    where
        concatProof : (c'':Shp) -> c' ++ c'' = (c++c1) ++ c''
        concatProof c'' = eqConcat c' (c++c1) c'' p

        assocRProof : (c'':Shp) -> (c++c1)++c'' = c++(c1++c'')
        assocRProof c'' = assocR c c1 c''
```

```
        transProof : (c'':Shp) -> c' ++ c'' = c ++ (c1 ++ c'')
        transProof c'' = trans (concatProof c'') (assocRProof c'')

        symmProof : (c'':Shp) -> c ++ (c1 ++ c'') = c' ++ c''
        symmProof c'' = symmShp (transProof c'')

        expand : (c'':Shp) -> evalTy0 Theta (c'++c'') -> evalTy0 Theta (c++(c1++c''))
        expand c'' v = convEvTyCtx (c'++c'') (c ++ (c1 ++ c'')) (transProof c'') v

           contract : (c'':Shp) -> evalTy0 Theta' (c++(c1++c'')) -> evalTy0 Theta' (c'++c'')
        contract c'' v = convEvTyCtx (c ++ (c1 ++ c'')) (c'++c'') (symmProof c'') v

-- Propiedad de que la semántica de un tipo en un objeto c, es la semántica
-- de ese tipo en el objeto c++ShpUnit.
convL : {Pt : PhraseType} -> {C : Shp} -> evalTy0 Pt C -> evalTy0 Pt (C ++ ShpUnit)
convL {C=c} eval = convEvTyCtx c (c++ShpUnit) (neutDShp c) eval

-- Propiedad de que la semántica de un tipo en un objeto c++ShpUnit, es la semántica
-- de ese tipo en el objeto c.
convR : {Pt : PhraseType} -> {C : Shp} -> evalTy0 Pt (C ++ ShpUnit) -> evalTy0 Pt C
convR {C=c} eval = convEvTyCtx (c++ShpUnit) c (neutLShp c) eval

toAcc : {c:Shp} -> {pt:PhraseType} ->
        (d:DataType) -> evalTy0 pt c -> evalDTy d -> shapes c -> shapes c
toAcc {pt=IntAcc}  IntDT  p = p
toAcc {pt=RealAcc} RealDT p = p
toAcc {pt=BoolAcc} BoolDT p = p

toExp : {c:Shp} -> {pt:PhraseType} ->
        (d:DataType) -> evalTy0 pt c -> shapes c -> evalDTy d
toExp {pt=IntExp}  IntDT  p = p
toExp {pt=RealExp} RealDT p = p
toExp {pt=BoolExp} BoolDT p = p

toComm : {c:Shp} -> {pt:PhraseType} -> evalTy0 pt c -> shapes c -> shapes c
toComm {pt=Comm} p = p

fromAcc : {c:Shp} -> {pt:PhraseType} ->
          (d:DataType) -> (evalDTy d -> shapes c -> shapes c) -> evalTy0 pt c
fromAcc {pt=IntAcc}  IntDT  p = p
fromAcc {pt=RealAcc} RealDT p = p
fromAcc {pt=BoolAcc} BoolDT p = p

fromExp : {c:Shp} -> {pt:PhraseType} ->
        (d:DataType) -> (shapes c -> evalDTy d) -> evalTy0 pt c
fromExp {pt=IntExp}  IntDT  p = p
fromExp {pt=RealExp} RealDT p = p
fromExp {pt=BoolExp} BoolDT p = p

fromFun : {c:Shp} -> (pt,pt',pt'':PhraseType) ->
          ((c':Shp) -> evalTy0 pt (c++c') -> evalTy0 pt' (c++c')) -> evalTy0 pt'' c
fromFun pt pt' (pt:->pt') p = p

fromVar : {c:Shp} -> {pt:PhraseType} ->
        (d:DataType) -> ( evalDTy d -> shapes c -> shapes c
                        , shapes c -> evalDTy d
                        ) -> evalTy0 pt c
fromVar {pt=IntVar}  IntDT  p = p
fromVar {pt=RealVar} RealDT p = p
fromVar {pt=BoolVar} BoolDT p = p
```

Sintaxis y semántica de los contexto de λ^{like}

```
-- ################## Versión sintácticas ##################
mutual
    -- Representación de un contexto sintáctico.
    data Ctx : Type where
        CtxUnit : Ctx
        Prepend : (p:Ctx) -> (i:Identifier) -> (pt:PhraseType) ->
                  Fresh p i -> Ctx
    -- Representa si un identificador es fresco para un contexto.
    data Fresh : Ctx -> Identifier -> Type where
        FUnit : (i:Identifier) -> Fresh CtxUnit i
        FCons : (i:Identifier) -> (pt':PhraseType) -> (i':Identifier) ->
                (p:Ctx) -> (fi':Fresh p i') -> so (i/=i') -> (Fresh p i) ->
                Fresh (Prepend p i' pt' fi') i

-- Representa la pertenencia de un identificador en un contexto.
data InCtx : Ctx -> Identifier -> Type where
    InHead : (p:Ctx) -> (i:Identifier) -> (pt:PhraseType) ->
             (fi:Fresh p i) -> InCtx (Prepend p i pt fi) i
    InTail : (p:Ctx) -> (i:Identifier) -> (pt:PhraseType) ->
             (j:Identifier) -> (fj:Fresh p j) ->
             InCtx p i -> InCtx (Prepend p j pt fj) i

-- ################## Versión semánticas ##################

-- Semántica de un contexto en un objeto con forma.
-- Representación de un contexto semántico.
evalCtxO : Ctx -> Shp -> Type
evalCtxO CtxUnit C = ()
evalCtxO (Prepend p _ pt _) C = (evalCtxO p C, evalTyO pt C)

-- Semántica de un contexto aplicado a morfismos entre objetos.
evalCtxM : {C:Shp} -> {C':Shp} ->
           (p:Ctx) -> C <= C' -> evalCtxO p C -> evalCtxO p C'
evalCtxM CtxUnit m _ = ()
evalCtxM (Prepend p i pt _) m (eta,etai) = (evalCtxM p m eta, evalTyM pt m etai)

-- Transforma un environment con forma C, en uno con forma C ~: Dt
liftEta : (c:Shp) -> (dt:DataType) -> (p:Ctx) ->
          evalCtxO p c -> evalCtxO p (c ~: dt)
liftEta c dt p eta = evalCtxM p (c >>> (ShpUnit ~: dt)) eta

-- Transforma un environment con forma C, en uno con forma C++C'
liftEta' : (c:Shp) -> (c':Shp) -> (p:Ctx) -> evalCtxO p c ->
           evalCtxO p (c ++ c')
liftEta' c c' p eta = evalCtxM p (c >>> c') eta

-- Buscar el valor de un identificador en un contexto semántico.
search : (c:Shp) -> (p:Ctx) -> (i:Identifier) -> (pt:PhraseType) ->
         InCtx p i -> evalCtxO p c -> evalTyO pt c
search _ (Prepend _ i pt _) i pt (InHead _ i pt fi) (eta,v) = v
search c (Prepend ctx j pt _) i pt (InTail _ _ pt j _ inc) (eta,_) = search c ctx i pt inc eta

-- Actualizar el valor de un identificador.
update : (c:Shp) -> (p:Ctx) -> evalCtxO p c -> (i:Identifier) ->
         (pt:PhraseType) -> evalTyO pt c -> (fi:Fresh p i) -> evalCtxO (Prepend p i pt fi) c
update _ p eta i pt z fi = (eta,z)
```

Representación de la categoría de formas

```
-- Constructor de morfismo entre dos objetos con forma.
infixr 10 >>>
-- Pegar por detras para objetos con forma.
infixr 10 ~:
```

```
-- Concatenar objetos con forma.
infixl 8 ++
-- Concatenar shapes.
infixl 8 <++>

-- Representa la forma del estado de la parte imperativa.
data Shp = ShpUnit | (~:) Shp DataType

instance Eq Shp where
    ShpUnit == ShpUnit = True
    (c~:dt) == (c'~:dt') = dt == dt' && c == c'
    _ == _ = False

-- Transformación de un objeto con forma a la representación
-- de un conjunto de estados con esa forma.
shapes : Shp -> Type
shapes ShpUnit = ()
shapes (c~:dt) = (shapes c,evalDTy dt)

-- Concatena objetos con forma.
(++) : Shp -> Shp -> Shp
C' ++ ShpUnit = C'
C' ++ (c ~: dt) = (C' ++ c) ~: dt

-- Establece la igualdad cuando pegamos por delante a objetos con forma.
cong : (C:Shp) -> (C':Shp) -> (C=C') -> (C ~: dt = C' ~: dt)
cong c c refl = refl

-- Propiedad de simetria para los objetos con forma.
symmShp : {C : Shp} -> {C' : Shp} -> C = C' -> C' = C
symmShp cEqc' = sym cEqc'

eqConcat : (C : Shp) -> (C' : Shp) -> (C'':Shp) -> C = C' -> C++C'' = C'++C''
eqConcat c c c'' refl = refl

trans : {C : Shp} -> {C' : Shp} -> {C'' : Shp} -> C = C' -> C' = C'' -> C = C''
trans refl refl = refl

assocL : (C : Shp) -> (C' : Shp) -> (C'' : Shp) -> C ++ (C' ++ C'') = (C ++ C') ++ C''
assocL c c' ShpUnit = refl
assocL c c' (c'' ~: dt) = cong (c++(c'++c'')) ((c++c')++c'') (assocL c c' c'')

assocR : (C : Shp) -> (C' : Shp) -> (C'' : Shp) -> (C ++ C') ++ C'' = C ++ (C' ++ C'')
assocR c c' ShpUnit = refl
assocR c c' (c'' ~: dt) = cong ((c++c')++c'') (c++(c'++c'')) (assocR c c' c'')

-- Propiedad de neutro a derecha.
neutDShp : (C:Shp) -> C = (C ++ ShpUnit)
neutDShp ShpUnit = refl
neutDShp (c ~: dt) = cong c (c ++ ShpUnit) (neutDShp c)

-- Propiedad de neutro a izquierda.
neutLShp : (C:Shp) -> (C ++ ShpUnit) = C
neutLShp ShpUnit = refl
neutLShp (c ~: dt) = cong (c ++ ShpUnit) c (neutLShp c)

-- Toma el tramo inicial con forma C, de un estado con forma C++C'
head : (C : Shp) -> (C' : Shp) -> shapes (C ++ C') -> shapes C
head c ShpUnit shp = shp
head c (c' ~: dt) (s,d') = head c c' s

-- Toma el tramo final con forma C', de un estado con forma C++C'
```

```
tail : (C : Shp) -> (C' : Shp) -> shapes (C ++ C') -> shapes C'
tail c ShpUnit shp = ()
tail c (c' ~: dt) (s,d')  = (tail c c' s,d')

-- Toma el ultimo elemento de tipo Dt del estado con forma C.
last : (C : Shp) -> (Dt:DataType) -> shapes C -> evalDTy Dt
last (c ~: dt) dt (s,v) = v

-- Concatena shapes.
(<++>) : {C:Shp} -> {C' : Shp} -> shapes C -> shapes C' -> shapes (C ++ C')
(<++>) {C'=ShpUnit} s _ = s
(<++>) {C'=c~:dt} s (s',d) = (s <++> s',d)

-- Agrega un valor por detras al estado.
prependShp : {c:Shp} -> {dt:DataType} ->
             shapes c -> evalDTy dt -> shapes (c ~: dt)
prependShp s z = (s,z)

eqShape : Shp -> Shp -> Shp -> Type
eqShape c c' c'' = c' = c ++ c''

-- Representa un morfismo entre dos objetos con forma.
data (<=) : Shp -> Shp -> Type where
        morp : {C:Shp} -> {C':Shp} ->
               ( shapes C' -> shapes C
               , (shapes C -> shapes C) -> (shapes C' -> shapes C')
               , ( c'' : Shp ** eqShape C C' c'')
               ) -> C <= C'

-- Constructor de un morfismo entre dos objetos.
(>>>) : (C : Shp) -> (C' : Shp) -> C <= (C ++ C')
c >>> c' = morp (head c c', sim, (c' ** refl))
    where
        sim : (shapes c -> shapes c) -> (shapes (c ++ c') -> shapes (c ++ c'))
        sim f sigma' = f (head c c' sigma') <++> tail c c' sigma'
```

Sintaxis del subtipado.

```
data (<~) : PhraseType -> PhraseType -> Type where
    VarToAcc : BoolVar <~ BoolAcc
    VarToExp : BoolVar <~ BoolExp

    IntExpToRealExp : IntExp <~ RealExp
    RealAccToIntAcc : RealAcc <~ IntAcc

    Reflx : (t:PhraseType) -> t <~ t
    Trans : {t:PhraseType} -> {t':PhraseType} -> {t'':PhraseType} ->
            t <~ t' -> t' <~ t'' -> t <~ t''

    SubsFun : {t0:PhraseType} -> {t0':PhraseType} ->
              {t1:PhraseType} -> {t1':PhraseType} ->
              t0 <~ t0' -> t1 <~ t1' -> (t0' :-> t1) <~ (t0 :-> t1')
```

Semántica del subtipado

```
evalLeq : t <~ t' -> (C:Shp) -> evalTy0 t C -> evalTy0 t' C
evalLeq VarToAcc c (a,e)    = a
evalLeq VarToExp c (a,e)    = e
evalLeq IntExpToRealExp c f = prim__intToFloat . f
evalLeq {t'=t} (Reflx t) c f = f
evalLeq (Trans leq leq') c f = evalLeq leq' c $ evalLeq leq c f
evalLeq (SubsFun leq leq') c f = \c' => evalLeq leq' (c ++ c') . (f c') . evalLeq leq (c ++ c')
```

Juicios de tipado del lenguaje λ^{like}

```
using (Pi:Ctx,Theta:PhraseType,Theta':PhraseType)
    data Phrase : Ctx -> PhraseType -> Type where
        Skip    : Phrase Pi Comm
        Seq     : Phrase Pi Comm -> Phrase Pi Comm -> Phrase Pi Comm
        While   : Phrase Pi BoolExp -> Phrase Pi Comm -> Phrase Pi Comm
        If      : {pt :PhraseType} -> Phrase Pi BoolExp ->
                  Phrase Pi pt -> Phrase Pi pt -> Phrase Pi pt

        I       : (i:Identifier) -> InCtx Pi i -> Phrase Pi Theta
        Assig   : (d:DataType) -> Phrase Pi (dtTOacc d) ->
                  Phrase Pi (dtTOexp d) -> Phrase Pi Comm
        NewVar : (d:DataType) -> (i:Identifier) -> Phrase Pi (dtTOexp d) ->
                 (fi:Fresh Pi i) -> Phrase (Prepend Pi i (dtTOvar d) fi) Comm ->
                 Phrase Pi Comm

        CInt    : Int   -> Phrase Pi IntExp
        CFloat  : Float -> Phrase Pi RealExp
        CBool   : Bool  -> Phrase Pi BoolExp

        BinOp : {a : DataType} -> {b : DataType} -> {d : DataType} ->
                (evalDTy a -> evalDTy b -> evalDTy d) ->
                Phrase Pi (dtTOexp a) -> Phrase Pi (dtTOexp b) -> Phrase Pi (dtTOexp d)
        UnOp  : {a:DataType} -> {b:DataType} ->
                (evalDTy a -> evalDTy b) ->
                Phrase Pi (dtTOexp a)  -> Phrase Pi (dtTOexp b)

        Lam   : (i:Identifier) -> (pt:PhraseType) -> (fi:Fresh Pi i) ->
                Phrase (Prepend Pi i pt fi) Theta' ->
                Phrase Pi (pt :-> Theta')
        App    : Phrase Pi (Theta :-> Theta') -> Phrase Pi Theta ->
                 Phrase Pi Theta'
        Rec    : Phrase Pi (Theta :-> Theta) -> Phrase Pi Theta

        Subs   : t <~ t' -> Phrase Pi t -> Phrase Pi t'
```

Semántica de las frases de λ^{like}

```
evalPhrase : {Pi:Ctx} -> {Theta:PhraseType} ->
             Phrase Pi Theta -> (C:Shp) -> evalCtxO Pi C -> evalTyO Theta C
evalPhrase (Subs leq var) c eta = evalLeq leq c (evalPhrase var c eta)
-- Semántica para los comandos.
evalPhrase (Assig d a e) c eta = \sigma => (\x => evalAcc x sigma) (evalExp sigma)
    where
        evalAcc : evalDTy d -> shapes c -> shapes c
        evalAcc = toAcc d $ evalPhrase a c eta
        evalExp : shapes c -> evalDTy d
        evalExp = toExp d $ evalPhrase e c eta
evalPhrase Skip c eta = \sigma => sigma
evalPhrase (Seq comm comm') c eta = \sigma => evalPhrase comm' c eta (evalPhrase comm c eta sigma)
evalPhrase (While b comm) c eta = fix (\f => \sigma =>
                                      if evalPhrase b c eta sigma
                                          then f (evalPhrase comm c eta sigma)
                                          else sigma)
evalPhrase {Pi=p} {Theta=pt} (I i iIn) c eta = search c p i pt iIn eta
-- Semántica para los valores constantes.
evalPhrase (CInt i)   c eta = \sigma => i
evalPhrase (CFloat r) c eta = \sigma => r
evalPhrase (CBool b)  c eta = \sigma => b
evalPhrase (BinOp {a} {b} {d} op x y) c eta = fromExp d (\sigma => (op (z sigma) (z' sigma)))
    where z : shapes c -> evalDTy a
          z = toExp a $ evalPhrase x c eta
```

```
          z' : shapes c -> evalDTy b
          z' = toExp b $ evalPhrase y c eta
evalPhrase (UnOp {a} {b} op x) c eta = fromExp b (\sigma => op (z sigma))
    where
        z : shapes c -> evalDTy a
        z = toExp a $ evalPhrase x c eta
evalPhrase {Theta=(pt :-> t')} {Pi=p} (Lam i pt fi b) c eta = evalLambda
    where
        newLeta : (C':Shp) -> evalTyO pt (c ++ C') -> evalCtxO (Prepend p i pt fi) (c ++ C')
        newLeta c' z = update (c++c') p (liftEta' c c' p eta) i pt z fi

        evalLambda : (C':Shp) -> evalTyO pt (c ++ C') -> evalTyO t' (c ++ C')
        evalLambda c' z = evalPhrase b (c++c') (newLeta c' z)
evalPhrase (App e e') c eta = convR $ (evalPhrase e c eta ShpUnit) (convL $ evalPhrase e' c eta)
evalPhrase (Rec e) c eta = fix $ convR . (evalPhrase e c eta ShpUnit) . convL
evalPhrase {Theta=pt} (If {pt} b e e') c eta =
            case pt of
                Comm        => makeComm Comm
                IntExp      => makeExp IntDT
                RealExp     => makeExp RealDT
                BoolExp     => makeExp BoolDT
                IntAcc      => makeAcc IntDT
                RealAcc     => makeAcc RealDT
                BoolAcc     => makeAcc BoolDT
    where
        makeComm : (pt:PhraseType) -> evalTyO pt c
        makeComm Comm = \sigma => if evalPhrase b c eta sigma
                                  then toComm (evalPhrase e c eta) sigma
                                  else toComm (evalPhrase e' c eta) sigma
        makeAcc : DataType -> evalTyO pt c
        makeAcc dt = fromAcc dt (\z => \sigma => if evalPhrase b c eta sigma
                                                  then toAcc dt (evalPhrase e c eta) z sigma
                                                  else toAcc dt (evalPhrase e' c eta) z sigma)
        makeExp : DataType -> evalTyO pt c
        makeExp dt = fromExp dt (\sigma => if evalPhrase b c eta sigma
                                              then toExp dt (evalPhrase e c eta) sigma
                                              else toExp dt (evalPhrase e' c eta) sigma)
evalPhrase {Pi=p} (NewVar d i vInit fi comm) c eta =
        \sigma => head c shpDT (evalComm (prependShp {dt=d} sigma (evalInit sigma)))
    where
        shpDT : Shp
        shpDT = ShpUnit ~: d

        a : evalDTy d -> shapes (c ~: d) -> shapes (c ~: d)
        a = \x => \sigma' => prependShp (head c shpDT sigma') x
        e : shapes (c ~: d) -> evalDTy d
        e = last (c ~: d) d

        evalInit : shapes c -> evalDTy d
        evalInit = toExp d $ evalPhrase vInit c eta

        zvar : evalTyO (dtTOvar d) (c ~: d)
        zvar = fromVar d (a,e)

        newAeta : evalCtxO (Prepend p i (dtTOvar d) fi) (c ~: d)
        newAeta = update (c ~: d) p (liftEta c d p eta) i (dtTOvar d) zvar fi

        evalComm : evalTyO Comm (c ~: d)
        evalComm = evalPhrase comm (c ~: d) newAeta
```

5 Conclusión

La tesis tenía como objetivo definir las ecuaciones semánticas y dar una implementación del lenguaje Forsythe. El plan entonces fue comenzar con un lenguaje simple e ir complejizando su definición hasta llegar a Forsythe; empezamos con $\lambda^{\rightarrow}$ que es el cálculo lambda simplemente tipado, luego agregábamos subtipado en $\lambda^{\leqslant}$, el siguiente paso fue agregar aspectos imperativos y obtener λ^{like}, este ya cerca de Forsythe. Y una vez con λ^{like} definido dar el salto hacia Forsythe.

Lo que se realizo en la tesis fue la definición e implementación de los lenguajes $\lambda^{\rightarrow}$, $\lambda^{\leqslant}$ y λ^{like}. Para cada uno además probamos distintas propiedades como continuidad, naturalidad o coherencia de la ecuaciones semánticas.

En cuanto a $\lambda^{\rightarrow}$ y $\lambda^{\leqslant}$ podemos mencionar que fueron especialmente prácticos al momento de estudiar los sistemas de tipado y sobre todo para comenzar con la implementación en un lenguaje de programación con tipos dependientes. Además desde el punto de vista de las propiedades fue muy interesante pensar el significado de coherencia para $\lambda^{\leqslant}$ y su contraste con [3, Prop 4].

Para λ^{like} surgió la idea de dar una semántica que no contemple stack discipline, además de la semántica general con stack discipline que caracteriza a este tipo de lenguajes. La semántica entonces que se definió intento ser lo mas general posible posible utilizando categorías de manera de poder, cambiando lo mínimo posible, seleccionar entre una semántica u otra. El secreto parecía estar en tener una categoría de estados con algunas pequeñas restricciones y luego tener dos versiones de esta categoría.
La implementación de λ^{like} termino difiriendo un poco de la semántica denotacional que dimos y de la cual probamos ciertas propiedades, esto debido principalmente por la utilización de categorías. En contraste con las implementaciones de $\lambda^{\rightarrow}$ y $\lambda^{\leqslant}$ que respetan bien la estructura de la semántica denotacional. En particular la definición de la categoría de estados que hacía falta para la semántica denotacional no aparece en la implementación sino que se utilizo una idea análoga, no categórica, que se puede encontrar en [5, Cap 19].

Las cosas que quedaron por hacer y pueden ser parte de trabajos futuros, fueron definir e implementar la semántica de Forsythe, para lo cual se propone definir un lenguaje intermedio entre λ^{like} y Forsythe, $\lambda^{\&}$. El cual es básicamente λ^{like} pero tal que el sistema de tipos soporta intersección, una prueba muy interesante para $\lambda^{\&}$ sería coherencia.
Además por el lado de la implementación sería bueno implementar categorías para después dar una implementación que se corresponda con la semántica denotacional que se dio para λ^{like}. Luego dar una implementación de $\lambda^{\&}$ parece ser bastante directo en base a la de λ^{like}.

Bibliografía

[1] Edwin Brady. *Programming in IDRIS: a tutorial*. 2012 (vid. pág. 1).

[2] Frank J. Oles. *Functor Categories and Store Shapes*. Birkhauser Boston Inc., 1997, págs. 3-12. ISBN: 978-1-4757-3851-3 (vid. págs. 1, 43).

[3] John C. Reynolds. "The Coherence of Languages with Intersection Types". En: *TACS*. Ed. por Takayasu Ito y Albert R. Meyer. Vol. 526. Lecture Notes in Computer Science. Springer, 1991, págs. 675-700. ISBN: 3-540-54415-1 (vid. págs. 32, 75).

[4] John C. Reynolds. *The Essence of Algol*. Birkhauser Boston Inc., 1997. Cap. The essence of ALGOL, págs. 67-88. ISBN: 0-8176-3880-6 (vid. pág. 35).

[5] John C. Reynolds. *Theories of Programming Languages*. Paperback re-issue. Cambridge University Press, 2009 (vid. págs. 1, 7, 12, 42, 43, 75).