Índice general

Introducción

Un lenguaje natural nace como un *dialecto* de algún lenguaje padre; después ambos divergen, pues lo largo de los años, conforme se presentan nuevas necesidades, se adquieren nuevas expresiones y palabras, y otras palabras y expresiones primitivas son relegadas al olvido, hasta que el dialecto llega a ser ininteligible para los hablantes del lenguaje padre.

Un lenguaje natural es definido por las personas que lo utilizan. Una expresión o palabra nueva, si es aceptada por un número suficiente de personas se convierte en una parte de su lenguaje. Un lenguaje natural es un medio de comunicación muy flexible, que continuamente está evolucionando para adaptarse a circunstancias cambiantes.

De forma similar, en los lenguajes de programación, si un grupo de programadores añade un nuevo constructor a un lenguaje ya establecido, podemos pensar que aparece un nuevo *dialecto* de dicho lenguaje, y nada impedirá, a ese grupo, utilizar tal construcción para comunicarse algoritmos entre ellos. Pero tal añadido no es suficiente para establecer un dialecto que a su vez produzca en nuevo lenguaje. Los ordenadores son bastante menos flexibles que las personas. El compilador del lenguaje debe modificarse para que acepte el nuevo constructor. Incluso en tal caso, existirán compiladores estándar del lenguaje que rechazarán cualquier programa que utilice el nuevo constructor. Existe una fuerte razón económica para la estandarización de los lenguajes, y es que cuando un programa se expresa en el lenguaje estándar puede ejecutarse sobre distintos entornos operativos (ordenadores, terminales, sistemas operativos) con resultados idénticos. En otras palabras, los programas escritos en un lenguaje estándar son portables. El costo del desarrollo de software de buena calidad es alto, de forma que la portabilidad del software nunca debe ser desestimada.

A su vez esta estandarización no parece deseable en los lenguajes naturales. Algunos lingüistas deploran la anarquía resultante del proceso de adquisición/desaparición de palabras y frases, e intentan especificar una versión reglada de su lenguaje. Creemos que tales intentos están condenados al fracaso, ya que van en contra de la flexibilidad del lenguaje.

Volviendo a los lenguajes de programación: la clave de la portabilidad es la es-

tandarización, aún cuando ella no es suficiente para resolver todos los problemas de la portabilidad es sin duda claramente crucial. En la práctica, sin embargo, incluso los pocos lenguajes que han sido objeto de un esfuerzo serio de estandarización no están libres de problemas de portabilidad. La causa de ello es que la descripción de un lenguaje ha tener en cuenta multitud de detalles que son difíciles de resolver satisfactoriamente si no se utilizan las herramientas adecuadas. Las especificaciones formales puede ayudar a resolver este problema: las técnicas matemáticas son especialmente efectivas cuando las circunstancias piden precisión y ausencia de ambigüedad.

La especificación de un lenguaje define tanto el conjunto de programas expresables en dicho lenguaje, como el significado de cada programa. En la vida ordinaria existen muchas cosas que queremos especificar: composiciones musicales, diseños de ingeniería, leyes científicas, leyes ordinarias. Al especificar una determinada materia habrá que elegir el método de especificación: formal o informal.

Una especificación informal se expresa en un lenguaje natural. Esto hace que todos, incluso los no especialistas, entiendan la especificación (al menos en principio). Pero la experiencia nos enseña que es muy difícil, por no decir imposible, especificar de forma precisa determinados aspectos de esta manera.

Una especificación formal se expresa en una notación especial cuyo significado es conocido de forma precisa, lo cual permite especificar determinados aspectos de una forma completa y sin ambigüedades. Su único inconveniente es hacer que las especificaciones no sean comprensibles para los no especialistas en la materia.

No parece razonable exigir que las especificaciones formales reemplacen a las descripciones en lenguaje natural de los lenguajes de programación. El lenguaje natural es una herramienta privilegiada para la comunicación entre las personas, y cualquier lenguaje debe tener un manual escrito en un lenguaje natural. Las especificaciones formales e informales juegan papeles complementarios, cuando en el documento escrito en lenguaje natural aparezcan ambigüedades o interpretaciones conflictivas, la especificación formal debe servir como último recurso. Por ello la especificación formal de los lenguajes de programación ha sido extensamente investigada y ha dado lugar a varias metodologías.

Objetivos

El objetivo de esta tesis es elaborar las herramientas necesarias para el diseño y la especificación formal de lenguajes de programación que tengan en cuenta el paradigma borroso. Hay que tener en cuenta que cuando se diseña la definición formal

de un lenguaje, existen dos requisitos fundamentales:

- La definición debe ser completa. Si no lo es, su utilidad como referencia queda muy mermada. Esta completitud solo puede alcanzarse utilizando una metodología matemáticamente bien definida

- La definición formal del lenguaje debe poder utilizarse en un entorno industrial, es decir una definición formal puede utilizarse:

    1. Como un estándar para el lenguaje, es decir para responder de una forma no ambigua a las preguntas que un programador o un implementador pueden hacerse acerca del significado de una construcción del lenguaje. La definición formal debe servir como un documento de referencia para la validación de implementaciones y como una guía para implementadores.
    2. Como un documento de referencia para justificar la validez de las optimizaciones y otras transformaciones de los programas. Las únicas optimizaciones válidas son aquellas que no alteran el significado de un programa.
    3. Como un documento de referencia para proporcionar propiedades de los programas escritos en el lenguaje. En particular, las definiciones formales permiten la derivación de reglas de inferencia que pueden utilizarse para asignar propiedades a los programas.
    4. Como entrada para un generador de compiladores

Se debe extremar el cuidado con la notación. Las consideraciones de compacidad y elegancia matemática, que son de gran importancia de un entorno científico pierden buena parte de esta importancia en un entorno ingenieril. Se debe dedicar un gran esfuerzo al establecer las definiciones y a su contenido intuitivo, para hacerlas accesibles al mayor número de lectores: implementadores, programadores, etc. Naturalmente, tal meta debe preservar el rigor matemático de la definición y debe ser el desarrollo de una notación conveniente.

Descripción global

Para especificar un lenguaje de programación se necesita un método para definirlo formalmente. Aún cuando existen varios métodos para ello: la semántica operacional, la semántica axiomática, las gramáticas atribuidas, etc. ninguno tiene una aceptación general. En nuestra tesis optamos por la semántica denotacional. Existen varias razones que nos han llevado a la elección de este método: permite la definición del lenguaje a cualquier nivel de detalle deseable, la matemática subyacente a este método ha sido extensamente investigada, el método está basado sobre una fuerte fundamentación teórica y finalmente este método es conveniente tanto para probar

la validez de las transformaciones de los programas como para probar propiedades de los mismos.[73], [75]

La finalidad de la semántica denotacional es asociar a todo programa un objeto matemático abstracto llamado su significado. Generalmente, el significado de un programa es algún objeto funcional, por ejemplo una función de su entrada en su salida. La aplicación que especifica como se asocia un significado a cualquier programa escrito en un determinado lenguaje de programación se llama la denotación semántica de dicho lenguaje de programación. Para definir apropiadamente la semántica denotacional de un lenguaje primero se debe definir un universo semántico, donde basar los significados. A continuación se describe como asociar un significado a toda componente atómica y para toda construcción del lenguaje y finalmente como derivar el significado de un fragmento compuesto a partir del significado de sus componentes. Por tanto la semántica denotacional no es nada ms que una definición de una función recursiva, eso si bastante grande, que aplica objetos sintácticos, los programas, en objetos semánticos, las funciones entrada/salida. Esta forma de definir la semántica de un lenguaje permite de manera natural asignar un significado no solo a programas completos sino también a trozos de programa.

Tenemos dos grandes concepciones: La semántica denotacional y la teoría de los conjuntos borrosos. Queremos reformular la primera para que sea capaz de dar respuesta a las necesidades que plantea la segunda. A su vez, las especificaciones denotacionales son muy parecidas a los programas escritos en un lenguaje funcional basado en el lambda cálculo. Es por tanto, el lambda cálculo el que debemos ampliar para que de cuenta del paradigma borroso.

La teoría de conjuntos borrosos supuso una revolución para las herramientas intelectuales que trataban de explicar y modelizar el mundo a través del cálculo. Los lenguajes de programación forman parte de tales herramientas. Algunos de los retos que ha planteado la teoría borrosa, aquellos a los que queremos dar respuesta en esta tesis, se pueden esquematizar de la siguiente forma:

- En el caso clásico la lógica subyacente en cualquier lenguaje de programación es la *Lógica clásica*. En el caso borroso existen toda una pléyade de *Lógicas borrosas*, cada una de ellas adecuada a una determinada estrategia o a algún objetivo. Deseamos que se pueda programar con facilidad en la lógica que el programador considere adecuada.

- Tratar los problemas del control de flujo cuando dicho control se lleva a cabo basándose en elementos borrosos, ello da lugar al *no determinismo*

- Definición de tipos de datos adecuados. Como se señala en [91] “una palabra en un lenguaje natural es a menudo un resumen de un concepto complejo, con múltiples aspectos, que somos incapaces de caracterizar de forma precisa.

Por esta razón la denotación de una palabra es generalmente un subconjunto borroso del universo del discurso". Queremos poder definir tipos que denoten a subconjuntos borrosos.

Por ello el plan que vamos a seguir es el siguiente:

- En el capítulo 1 expondremos de forma somera los aspectos que vamos a utilizar de los tres pilares en los que nos basamos:
  - El lambda cálculo en sus distintos aspectos
  - La semántica denotacional
  - La teoría de los conjuntos borrosos
- En el capítulo 2 ampliaremos el lambda cálculo, la herramienta fundamental de la semántica denotacional, para que tenga en cuenta el paradigma borroso. Para ello adjuntaremos, a cada construcción del lambda cálculo clásico, un elemento de un conjunto $\mathbb{D}$ sobre el que no establecemos ninguna premisa. Sin embargo. si exigimos que el cálculo resultante contenga al lambda cálculo clásico como un caso particular. Esta exigencia nos llevará a estudiar, lo que llamaremos versiones borrosas, de la sustitución y la β-reducción. El resultado será que el conjunto $\mathbb{D}$ ha de tener unas propiedades mínimas.
- En el capítulo 3 enriqueceremos del cálculo resultante, en particular introduciremos los sistemas numéricos, el resultado será una proliferación excesiva de símbolos, para manejarla y obtener un sistema adecuado a nuestras necesidades haremos uso de un interprete de "lambda cálculo puro borroso", su listado aparece en el apéndice que comienza en la página 159. Todo ello nos llevará a una caracterización más precisa de $\mathbb{D}$.
- En el capítulo 4 definiremos un lenguaje no determinista, muy simple, que maneja datos borrosos. Para ello habremos de analizar los conceptos de estado y los mecanismos que permiten transformar un estado en otro, pues mientras que en la programación clásica los estados tienen la estructura de un dominio plano, en la "programación borrosa" no sucede esto, en determinadas circunstancias un estado puede contener más o menos información que otro. Introduciremos los *dominios potencia* para manejar el no determinismo y discutiremos las diversas alternativas que existen para manejar el intervalo unidad $[0, 1]$, en el apéndice de la página 141 estudiamos con detalle las diversas ordenaciones que se pueden introducir en un dominio potencia muy simple. Finalmente daremos la semántica denotacional del lenguaje.
- El capítulo 5 es una muestra de la potencia de la herramienta que hemos construido. En él llevaremos a cabo una de las muy variadas ampliaciones que se pueden hacer al lenguaje del capítulo anterior. Introducimos funciones y procedimientos y mostraremos las formas en que los elementos borrosos, los

que llamaremos "índice" y "estrategia", se pueden transmitir entre bloques. Introduciremos los números trapezoidales como tipo básico, es decir como valores expresables. Permitiremos la definición de tipos lingüísticos. Finalmente daremos un prototipo en donde se implementarán las ideas y las propuestas que se hacen a lo largo de esta tesis, mostrando que son realizables. En el apéndice de la página 183 se lista un prototipo del lenguaje propuesto, programado en Objetive Caml (1.07) [42]. En el apéndice siguiente se muestran algunos programas ejecutables por el prototipo.

Capítulo 1

Sistemas formales y computación borrosa

1.1. Introducción

A mediados del siglo XIX Hamilton, De Morgan y Boole llevan a cabo los primeros sistemas formales lógicos. En sus trabajos hoy distinguiríamos el cálculo proposicional y el cálculo de predicados.

- El cálculo proposicional es un sistema con dos valores básicos, cierto y falso, y con operaciones básicas tales como la negación, conjunción o disyunción. Dentro de este cálculo es posible probar cuando una expresión arbitraria es o no es un teorema (siempre cierta), comenzando con axiomas (expresiones elementales que siempre son cierta), y aplicando reglas de inferencia para construir nuevos teoremas a partir de axiomas y teoremas ya existentes.

- El cálculo de predicados extiende el cálculo proposicional permitiendo expresiones que se refieran a valores no lógicos tales como números o conjuntos. Esto se consigue por medio de la introducción de predicados que generalizan las expresiones lógicas para describir propiedades de valores no lógicos y de funciones para generalizar operaciones sobre valores no lógicos. También se introduce la idea de cuantificadores para describir las propiedades de una serie de valores, así tendremos el cuantificador universal para aseverar que toda una serie de elementos tienen una propiedad y el cuantificador existencial para aseverar que uno de los elementos de una serie tiene una propiedad. Además se añaden axiomas y reglas de inferencia para expresiones cuantificadas. El cálculo de predicados puede aplicarse a distintas áreas por medio del desarrollo de predicados, funciones, axiomas y reglas de inferencia adecuados.

A finales de siglo XIX Peano desarrolló un sistema formal para la teoría de números. Se introducían los números a partir de 0 y de la función sucesor. Las demostraciones en este sistema están basadas en una forma de inducción cercana a la recursión.

A principios del siglo XX, Russell y Whitehead en su Principia Mathematica intentaron derivar la certeza matemática directamente de la certeza lógica, tratando de construir una descripción lógica de las matemáticas. A continuación Hilbert propuso un Programa para demostrar que los Principia describían la matemática totalmente. Era necesario demostrar que la descripción de la matemática en los Principia era completa y consistente. Pero Gödel demostró que cualquier sistema lo suficientemente potente como para describir la aritmética era necesariamente incompleto. Sin embargo el Programa de Hilbert promovió la investigación de un área que cristalizó en la teoría de la computabilidad, al tratar de desarrollar sistemas formales capaces de describir cualquier cálculo. En 1936, fueron propuestos tres sistemas formales para la computabilidad:

las máquinas de Turing de A.M.Turing

la teoría de funciones recursivas de S.C.Kleene

el lambda-cálculo de A.Church.

Cada uno de ellos está definido en términos de un conjunto sencillo de operaciones primitivas y un conjunto simple de reglas y además cada una tiene una teoría de la demostración.

Los tres sistemas se han mostrado equivalentes, en el sentido que:

- Un resultado en uno de los sistemas tiene equivalentes resultados en los otros sistemas. Por ejemplo Turing demostró que el problema de la parada es irresoluble. Esto también puede aplicarse al λ-cálculo y a la teoría de funciones recursivas, en efecto no hay ninguna forma de determinar si la evaluación de una λ-expresión arbitraria o una función recursiva termina.

- Cualquier sistema puede usarse para modelizar los otros sistemas.

En particular, cualquier resultado obtenido en un sistema formal es aplicable a los lenguajes de programación y cualquiera de estos sistemas puede usarse para describir los lenguajes de programación. Inversamente los lenguajes de programación pueden usarse para describir y por tanto implantar cualquiera de estos sistemas.

Una diferencia importante entre las máquinas de Turing, las funciones recursivas y el lambda cálculo es que mientras que en el sistema de las máquinas de Turing se considera el cálculo como una manipulación mecanizada de símbolos basado en la asignación y en el orden de evaluación, la teoría de funciones recursivas y el lambda cálculo se basan en la aplicación de funciones y en ambos el orden de evaluación es irrelevante. Además, Church y Rosser demostraron para el lambda cálculo que si distintos órdenes de evaluación terminan sus resultados coinciden. También demostraron que un determinado orden de evaluación puede alcanzar la terminación antes

que otro. Esto tiene importantes implicaciones porque puede ser más eficiente evaluar una parte de un programa en un orden y otra parte en otro orden. Además, si la evaluación en un lenguaje es independiente del orden, entonces es posible llevar a cabo la evaluación de los distintos subprogramas de un programa de forma paralela.

1.1.1. La generación de un sistema formal

Introducir en un sistema formal, ya establecido, como es el lambda cálculo, nuevos elementos para dar cuenta de aspectos no considerados en su origen, es una tarea que requiere, sobre todo en las primeras etapas, un cuidado especial, a fin de asentar con claridad los distintos elementos y construcciones primitivas. Para ello hemos recurrido a [16], en él se dan las convenciones de lo que llamaremos *estructura primitiva*. Una vez establecida esta, aparecerán los teoremas y propiedades interesantes, y las diversas opciones que en cada momento se pueden tomar. Por ello resumimos a continuación como se construye las bases del sistema formal. Pues, aunque no de forma explícita, han sido estas reglas las que nos han servido de guía a la hora de establecer la serie de definiciones que se formulan en el capítulo 2.

Un sistema formal se define por un conjunto de convenciones que llamaremos su *estructura primitiva*. Esta tiene tres partes:

a) Un conjunto de *objetos* a los que llamaremos obs

b) Un conjunto de *enunciados elementales* referidos a los obs

c) Conjunto de enunciados verdaderos o *teoremas elementales*

La parte a) se refiere a los obs, enumera ciertos *obs primitivos* o *átomos*, y ciertas *operaciones*, cada una de las cuales es un modo de combinar secuencias finitas de obs para formar un nuevo ob. Debe entenderse que:

- los obs del sistema son precisamente los formados a partir de los átomos mediante las operaciones y según las reglas de formación.
- obs construidos mediante procesos diferentes son obs distintos.

La parte b) enumera ciertos *predicados* cada uno de los cuales es un modo de formar enunciados partiendo de una secuencia finita de obs. Puesto que las partes a) y b) tienen rasgos comunes se tratan juntas, ellas enumeran las nociones primitivas y enuncian reglas de formación. Las operaciones y los predicados se agrupan bajo el nombre de *functivas*. Cada functiva tiene un cierto número finito de argumentos que llamaremos *aridad.* Las functivas de aridad 1 se llamarán unarias, las de aridad 2 binarias, y así sucesivamente. Dada una functiva $n-aria$ se llamará *cierre* de dicha functiva al *ob* o enunciado formado con n obs por dicha functiva.

A veces es cómodo admitir entre las functivas, como predicados de aridad cero, ciertos enunciados primitivos sin analizar, análogamente podrán contemplarse átomos como operaciones de aridad cero. O bien considerar functivas de aridad variable, ya que podrían verse como un conjunto de functivas, una para cada aridad admisible.

La parte c) formula los axiomas y las reglas deductivas del sistema. Los axiomas son enunciados elementales de los que se afirma que son verdaderos. Puede haber una lista finita de axiomas o los axiomas pueden darse mediante reglas que determinen un número infinito de axiomas (por ejemplo, mediante esquemas de axiomas). Las reglas deductivas establecen como deben derivarse los teoremas partiendo de los axiomas.

Además de estos elementos que constituyen los *postulados* del sistema, son necesarios predicados y operaciones que no aparecen entre los enunciados elementales, estas nociones se llamarán *auxiliares*; entre ellas tendremos la clasificación de los obs en categorías, la definición de la operación de sustitución, etc.

Las definiciones inductivas

En el párrafo anterior nos hemos ocupado de ciertas nociones que podríamos considerar como clases, algunas de ellas son dadas como por ejemplo los átomos, las operaciones, los predicados, los axiomas, etc; mientras que otras como lo obj, los enunciados elementales y los teoremas elementales son definidos. Este último tipo debe quedar especificado por una definición, a la que llamaremos *definición inductiva* y que consta de tres pasos:

1. Se especifican ciertos elementos iniciales, lo llamaremos paso de las *especificaciones iniciales*

2. Se describen ciertos procedimientos para construir elementos nuevos partiendo de elementos dados, lo llamaremos paso de los *principios genéticos*

3. Se entiende que todos los elementos de la clase se obtienen a partir de los elementos iniciales por iteración de dichos procedimientos, si se entiende que la definición es inductiva no es necesario formular este paso.

Una clase definida de este modo se llama una *clase inductiva*. Por último se introduce una restricción importante. Dada una clase C definida inductivamente si construimos una entidad A haciendo uso de las definiciones, está claro que $A \in C$. Pero si se presenta una entidad A, puede ocurrir que no haya ningún procedimiento finito que permita decidir si $A \in C$ o no. Cuando queda excluida esta posibilidad, es decir, cuando hay un proceso prescrito tal que, dada cualquier A, ese proceso determinará, efectivamente, si pertenece o no a C, entonces se dice que C es una *clase determinada*.

En un sistema formal las nociones de átomo, operación de aridad n, predicado, etc, es decir los conjunto a) y b) han de ser determinados en el sentido antes expuesto. En particular estas condiciones se cumplirán si las clases son finitas. El caso de un número infinito de átomos no presenta ningún inconveniente si se toman como obs de algún sistema formal más fundamental. Suponemos que cada una de esas clases dadas, si es infinita, constituye una secuencia numerable.

Por lo que respecta a los teoremas elementales no se exige que constituyan una clase determinada. Si se pusiera esta condición el sistema sería *decidible* es decir trivial.

Variables

En un sistema formal lo que tenemos es ciertos obs que se llamaran "variables", no son símbolos sino obs del sistema o variables formales, los hay de tres clases

Indeterminadas. Son átomos acerca de los cuales el sistema no especifica nada, excepto que es un ob.

Variables de sustitución. Son aquellos átomos respecto de los cuales hay una regla de sustitución. Tal regla exige que se especifique una clase de obs que puedan ser sustituidos en ciertas circunstancias por obs arbitrarios o por obs de una cierta categoría; los obs de esta clase constituyen las variables de sustitución. Las variables de sustitución no son indeterminadas porque desempeñan un papel respecto de la regla de sustitución. Tienen que formularse como categoría auxiliar así como el concepto de sustitución.

Variables ligadas. Un sistema contiene *variables ligadas* cuando se tiene formulado un conjunto de variables de sustitución y, por lo menos, una operación respecto de la cual estas variables desempeñen un papel especial.

Las indeterminadas y las variables de sustitución conjuntamente se llaman *variables libres.*

Sustitución

En un sistema formal la sustitución es una operación que hay que definir de forma abstracta. Consideremos un sistema que no tiene variables ligadas, sean a y b obs y sea x un átomo, es necesario definir el ob b' obtenido por sustitución de a por x en b. El ob b está construido a partir de los átomos mediante una construcción única. Entonces b' es el objeto obtenido por el mismo proceso de construcción, pero usando a en lugar de x. Esta definición es equivalente a la siguiente definición recursiva:

i) si b es x, entonces b' es a

ii) si b es un átomo distinto de x, entonces b' es b

iii) si b se obtiene por una operación γ a partir de los argumentos $c_1, c_2, \ldots, c_n$, entonces b' se obtiene mediante γ a partir de los argumentos $c'_1, c'_2, \ldots, c'_n$

Esta es la *sustitución simple* que habrá que revisar cuando existan variables ligadas.

1.2. El lambda cálculo

1.2.1. El lambda cálculo como sistema formal

El lenguaje

El lambda cálculo como sistema formal tiene el siguiente lenguaje:

Alfabeto:

variables: a_0,a_1,...

símbolos impropios: λ,(,)

igualdad: $=$

reducción: $\preceq$

Términos: Los términos se definen inductivamente por:

1. Cualquier variable es un término
2. Si M,N son términos, entonces (MN) es un término
3. Si M es un término y x es una variable, entonces $(\lambda x M)$ es un término

Fórmulas: Si M y N son términos entonces

- $M = N$
- $M \preceq N$

son *fórmulas*

Notación sintáctica:

M, N,... denotan términos

$a, b, \ldots, x, y, \ldots$ denotan variables

$\equiv$ denota la identidad sintáctica

$M_1 M_2 \ldots M_n = (\ldots (M_1 M_2) \ldots M_n)$

$\lambda x_1 \ldots x_n.M = (\lambda x_1 (\lambda x_2 \ldots (\lambda x_n.(M)) \ldots)$

Para poder formular los axiomas se definen inductivamente:

- El *conjunto de variables libres* de un término

  · $FV(x) = \{x\}$
  · $FV(MN) = FV(M) \cup FV(N)$
  · $FV(\lambda xM) = FV(M) - \{x\}$

- El *conjunto de variables ligadas* de un término

  · $BV(x) = \emptyset$
  · $BV(MN) = BV(M) \cup BV(N)$
  · $BV(\lambda xM) = BV(M) \cup \{x\}$

- La *sustitución* de un término N en las ocurrencias libres de la variable x en M:

  · $x\,\langle N/x\rangle = N$
  · $y\,\langle N/x\rangle = y$ si $y \neq x$
  · $(M_1M_2)\,\langle N/x\rangle = (M_1\,\langle N/x\rangle)(M_2\,\langle N/x\rangle)$
  · $(\lambda xM)\,\langle N/x\rangle = \lambda xM$
  · $(\lambda yM)\,\langle N/x\rangle = \lambda y(M\,\langle N/x\rangle)$ si $y \neq x$

Esquemas de axiomas y reglas

El λ-cálculo se define por los siguientes esquemas de axiomas y reglas:

I 1. $\lambda xM \succeq \lambda y\,\langle x/y\rangle\,M$ si $y \notin FV(M) \cup BV(M)$ (α-reducción)

2. $(\lambda xM)N \succeq M\,\langle N/X\rangle$ si $BV(M) \cap FV(N) = \emptyset$ (β-reducción)

II 1. $M = M$

2. $\dfrac{M = N}{N = M}$

3. $\dfrac{M = N, N = L}{M = L}$

4. $\dfrac{M = M'}{\lambda xM = \lambda xM'}$, $\dfrac{M = M'}{MZ = M'Z}$, $\dfrac{M = M'}{ZM = ZM'}$

III 1. $M \succeq M$

2. $\dfrac{M \succeq N, N \succeq P}{M \succeq P}$

3. $\dfrac{M \succeq M'}{\lambda x M \succeq \lambda x M'}, \dfrac{M \succeq M'}{MZ \succeq M'Z}, \dfrac{M \succeq M'}{ZM \succeq ZM'}$

4. $\dfrac{M \succeq M'}{M = M'}$

Si además le añadimos el siguiente esquema de axiomas

I 3. $\lambda x(Mx) \succeq M$ si $x \notin FV(M)$ (η-reducción)

se tiene el $\lambda\eta$-cálculo o λ-cálculo + extensionabilidad

El concepto de reducción

Noción de reducción *Una noción de reducción sobre Λ es cualquier relación binaria sobre Λ.*

Si ϱ es una noción de reducción sobre Λ, entonces la notación infija $M \ \varrho \ N$ es equivalente a $(M, N) \in \varrho$. Si $M \ \varrho \ N$ entonces se dice que M es un ϱ-redex y que N es su contracto. Si ϱ_1 y ϱ_2 son nociones de reducción entonces también lo es $\varrho_1 \cup \varrho_2$

Compatible *Una relación binaria $\rightarrow$ sobre Λ se dice que es compatible (con las operaciones sintácticas) sii $M \rightarrow N$ implica $P \rightarrow Q$ cuando M es un subexpresión de P y Q se obtiene a partir de P reemplazando una ocurrencia de M en P por N.*

Relación de reducción *Una relación de reducción sobre Λ es una relación binaria que es compatible, reflexiva y transitiva.*

Relación de igualdad *Una relación de igualdad sobre Λ es una relación de equivalencia compatible.*

Sea ϱ una noción de reducción sobre Λ. Entonces ϱ induce las siguientes relaciones binarias:

$\rightarrow_\varrho$, llamada reducción en un paso, es el cierre compatible de ϱ

$\Rightarrow_\varrho$, llamada ϱ-reducción, es el cierre reflexivo y transitivo de $\rightarrow_\varrho$

$=_\varrho$, llamada ϱ-igualdad, es la relación de equivalencia generada por $\Rightarrow_\varrho$

Propiedad del diamante *Sea $\rightarrow$ una relación binaria sobre Λ. Entonces $\rightarrow$ satisface la propiedad del diamante si, para todo $T, U, V \in \Lambda$ con $T \rightarrow U$ y $T \rightarrow V$, existe algún $W \in \Lambda$ tal que $U \rightarrow W$ y $V \rightarrow W$.*

Church-Rosser *Una noción de reducción ϱ es de Church-Rosser sii $\Rightarrow_\varrho$ satisface la propiedad del diamante.*

Otras definiciones auxiliares

Radical o redex y contracto *Un término M de la forma $(\lambda x.u\ t)$ se llamará un* **radical** *y el elemento N de la forma $u\langle t/x\rangle$ tal que $M \succeq N$ se dirá que es su* **contracto**.

Normal *Un elemento $M \in \Lambda$ se dice que es* **normal** *si no contiene ningún radical. Los elementos normales son pues aquellos que se han obtenido aplicando un número finito de veces las siguientes reglas:*

1. *Si es una variable entonces es normal*
2. *Si es de la forma $\lambda x.t$ sin que t contenga a ningún radical, entonces también es normal.*
3. *Si es de la forma $(u\ t)$, donde u y t no contienen radicales y u no comienza por λ.*

Se tienen las siguientes propiedades

1. *Un elemento es de forma normal sii su parte-λ es de la forma $\lambda x_1 \ldots \lambda x_k(x)t_1 \ldots t_n$ $(k, n \geq 0)$ y*
   a) *x es una variable.*
   b) *$t_1, \ldots, t_n$ no contienen redex*
2. *Un elemento M es normal sii $M \succeq N$ es falso para todo elemento N*

Normalizable *Un elemento M se dice que es* **normalizable** *si existe un elemento normal N tal que $M \succeq N$.*

Fuertemente normalizable *Un elemento M se dice que es* **fuertemente normalizable** *si no existe ninguna sucesión infinita $M_0 = M, M_1, \ldots M_n, \ldots$ tal que $M_i\ \rho\ M_{i+1}$ para todo $i \geq 0$.*

1.2.2. Árboles

Cada término del lambda cálculo puede considerarse como un árbol considerando:

- las abstracciones, como símbolos unarios, representados por $'\lambda'$

- las aplicaciones, como símbolos binarios, representados por $'@'$
- las variables, como símbolos de aridad cero

A continuación daremos la definición usual de los árboles. En sentido estricto, no sería necesario la utilización de los árboles en esta tesis, sin embargo los utilizaremos pues dan una buena visión de las notaciones y los problemas que se nos irán presentado.

Secuencia *Llamaremos* $Sec \subset N$ *al conjunto de secuencias de números*

$$Sec = \{\langle n_1, \ldots, n_k\rangle \,|k \in N, n_1, \ldots, n_k \in N\}$$

Suponemos que tenemos definidas las siguientes funciones y relaciones en Sec:

- Si $a = \langle n_1, \ldots, n_k\rangle \in$ Sec, entonces tam$(a) =$ k
- Si $a = \langle n_1, \ldots, n_p\rangle$ y $b = \langle m_1, \ldots, m_q\rangle \in$ Sec entonces:
  - $a * b = \langle n_1, \ldots, n_p, m_1, \ldots, m_q\rangle$
  - $a \leq b$ sii $p \leq q$ y $n_i = m_i$ para $1 \leq i \leq p$

Por convención $\langle\rangle = \epsilon \in$ Sec y tam$(\epsilon) = 0$.

Escribiremos $a < b$ si $a \leq b$ y no se cumple $b \leq a$.

Árbol *Un* **árbol** *A es un conjunto de secuencias de números tales que:*

1. *Si* $a \in A$ *y* $b \leq a$ *entonces* $b \in A$
2. *Si* $1 \leq i \leq j$ *y* $a * \langle j\rangle \in A$ *entonces* $a * \langle i\rangle \in A$

Los elementos de A son los **nodos** del árbol.

Subárbol *El* **subárbol** *de A en el nodo a (notación A/a) es el árbol* $\{b|a * b \in A\}$

Signatura *Una* **signatura** *es un conjunto* Σ *junto con una aplicación ar:* $\Sigma \rightarrow N$. *Si* $f \in \Sigma$ *y ar(f)* $= n$ *se dice que f tiene aridad n. Si* $n \geq 1$ *se dice que f es un símbolo funcional y si* $n = 0$ *se dice que es una constante.*

Árbol etiquetado *Sea* Σ *una signatura. Un* **árbol** Σ**-etiquetado** *es una aplicación parcial* $\varphi : Sec \rightarrow \Sigma$ *tal que el conjunto*

$$T = \{a \in Sec \,|\varphi(a) \text{ está definida}\}$$

es un árbol y además si ar(a) $= n$, *entonces*

$$a * \langle 1 \rangle, \ldots, a * \langle n \rangle \in T \ y \ a * \langle n+1 \rangle \notin T$$

T es llamado el árbol subyacente

La etiqueta en el nodo $a \in T$ es $\varphi(a)$.

1.2.3. Representación por árboles

La representación de los $\lambda-$términos como árboles se lleva a cabo considerando cada λx como un símbolo unario, las aplicaciones como símbolos binarios escritos @ y cada variable como un símbolo de aridad cero. Y ya que, en este caso, las signaturas están determinadas, se utiliza un alfabeto especializado para designar las ocurrencias de símbolos en un árbol: $A = \{0,1,2\}$. Si M es un término representado por un árbol, numeramos las flechas que salen de cada nodo de la siguiente forma:

- 0 para la flecha que parte de un λ
- 1 y 2 par las flechas izquierda y derecha de una aplicación @

Cualquier símbolo de M se designa por la palabra obtenida por concatenar el número de las flechas del camino que lleva hasta él partiendo de la raíz

La adaptación a los $\lambda-$términos de la anterior nomenclatura es:

1. Para las variables
$\mathcal{A}(x) = \{\epsilon\}$
$x(\epsilon) = x$
$x/\epsilon = x$

2. Para las abstracciones
$\mathcal{A}(\lambda x.M) = \{\epsilon\} \cup 0.\mathcal{A}(\mathcal{M})$
$(\lambda x.M)(\epsilon) = \lambda x$
$(\lambda x.M)/\epsilon = \lambda x.M$
$(\lambda x.M)(0.u) = M(u)$
$(\lambda x.M)/0.u = M/u$

3. Para las aplicaciones
$\mathcal{A}(MN) = \{\epsilon\} \cup 1.\mathcal{A}(\mathcal{M}) \cup .\mathcal{A}(\mathcal{N})$
$(MN)(\epsilon) = @$
$(MN)/\epsilon = MN$
$(MN)(1.u) = M(u)$

$(MN)/1.u = M/u$
$(MN)(2.u) = N(u)$
$(MN)/2.u = N/u$

1.2.4. El lambda cálculo modelo de los lenguajes de programación

Los lenguajes de programación pueden considerarse también como modelos generales de la calculabilidad, por ello el lambda cálculo puede considerarse como un modelo de los lenguajes de programación. Para ello será necesario utilizar un lambda cálculo con constantes y seleccionar un subconjunto de λ-expresiones para representar a los enteros de forma apropiada. Entonces la regla de β-conversión será muy parecida al mecanismo de llamada de un procedimiento en un lenguaje de programación y aparecerán problemas similares al paso de parámetros.

El modelo así construido es demasiado general, ya que los lenguajes de programación, por diversas razones: seguridad, eficacia, velocidad, ... de la compilación, imponen restricciones de tipos, por lo que un lambda cálculo con tipos es más apropiado.

1.2.5. El lambda cálculo con tipos

En el lambda cálculo sin tipos pueden aparecer expresiones que tienen una difícil interpretación. Por ello se selecciona un subconjunto estricto de la λ-expresiones, el conjunto de las λ-expresiones tipadas que no darán problemas de interpretación.

Un tipo es una expresión que representa a un conjunto de objetos. Así si α y β son dos tipos que representan a los conjuntos D_α y D_β, representaremos por $\alpha \longrightarrow \beta$ al conjunto $D_\alpha \longrightarrow D_\beta$ de las funciones de D_α en D_β. Toda λ-expresión tipada M tiene un tipo que representaremos por $\tau(M)$. Por ello se puede definir el conjunto de las λ-expresiones tipadas como el mínimo conjunto L_t que contiene a un conjunto V_t de variables tipadas y un conjunto C_t de constantes tipadas y que es cerrado respecto de:

- Aplicación : Si M y $N \in L_t$ y $\tau(M) = \alpha \longrightarrow \beta$ y $\tau(N) = \alpha$, entonces $(MN) \in L_t$ y $\tau((MN)) = \beta$
- Abstracción: Si $M \in L_t$ y $x \in V_t$, entonces $(\lambda x.M) \in L_t$. Y además si $\tau(x) = \alpha$ y $\tau(M) = \beta$, entonces $\tau((\lambda x.M)) = \alpha \longrightarrow \beta$

El lambda cálculo tipado tiene la propiedad de Church Rosser y una interpretación intuitiva fundamentada sobre la jerarquía de los funcionales asociados a las expresiones de tipo. Sin embargo, no es un modelo general de calculabilidad, ni es un buen

modelo de los lenguajes de programación, ya que las restricciones de tipo hacen que todas las reducciones acaben en una forma normal.

1.3. Semántica denotacional

La teoría matemática subyacente a la semántica denotacional fue desarrollada por Dana Scott [73]. La teoría que vamos a utilizar difiere ligeramente de la original propuesta por Scott; suponemos que trabajamos sobre ordenes parciales completos (cpo) en lugar de hacerlo sobre retículos.

A continuación resumimos los conceptos y definiciones más importantes de dicha teoría. Para un tratamiento más completo ver [73], [75], [68]

1.3.1. Funciones

Sean A y B dos conjuntos arbitrarios; $A \to B$ denota el conjunto de todas las funciones de A en B. "$\to$" es asociativa por la derecha. Se supone que todas las funciones o bien son constantes o tienen un argumento. Esto permite escribir la aplicación de funciones de forma "currificada" como en $f\ x$. Esta yuxtaposición es asociativa por la izquierda. Funciones con varios argumentos como $f \in A \times B \to C$ se aplican a tuplas como en $f(x, y)$. Alternativamente, f puede considerarse definida en el dominio $A \to (B \to C)$ en cuyo caso se escribe $f\ x\ y$.

Se pueden construir nuevas funciones a partir de unas funciones dadas utilizando la composición el condicional y la λ-abstracción como definimos a continuación. Este lenguaje es esencialmente el λ-cálculo de Church.

λ-abstracción Si $T(x)$ es un término donde x es posiblemente un variable libre, entonces $\lambda x.T(x)$ denota a una función f tal que $f\ y = T(y)$. Las notaciones "**let** $x = T_1$ **in** T_2" y "T_2 **where** $x = T_1$" son equivalentes a $(\lambda x.T_2)T_1$.

Redefinición Si $f \in A \to B$, entonces $[x \mapsto y]f$ denota a la función $\lambda a.$ **if** $a = x \Rightarrow y \ \square\ f\ a$

1.3.2. Definiciones básicas

Orden parcial Un orden parcial es un conjunto A junto con una relación binaria $\sqsubseteq$ que es reflexiva antisimétrica y transitiva.

Acotado Un subconjunto $P \subseteq A$ de un orden parcial A se dice que está acotado si existe un $x \in A$ tal que $y \sqsubseteq x$ para cada $y \in P$ en cuyo caso se dice que x es una cota superior de P.

Menor cota La menor de las cotas superiores de un subconjunto $P \subseteq A$ es una cota superior $\sqcup P$ tal que $\sqcup P \sqsubseteq x$ para cualquier otra cota superior x de P. La menor de las cotas superiores si existe es única.

Dirigido Un subconjunto $P \subseteq A$ es dirigido si todo conjunto finito $u \subseteq P$ tiene una cota superior $x \in P$. Un conjunto dirigido P nunca es vacio.

Cpo Un orden parcial A es completo *(cpo)* si todo subconjunto dirigido $P \subseteq A$ tiene la menor de los cotas superiores.

Puntiagudo Si un orden parcial A tiene un menor elemento se dice que es puntiagudo.

Compacto Sea A un cpo. Un elemento $x \in A$ es compacto si, para toda colección dirigida M tal que $x \sqsubseteq \sqcup M$ existe algún $y \in M$ tal que $x \sqsubseteq y$. Se define $\mathsf{K}(A)$ como el conjunto de los elementos compactos de A.

Algebraico Un cpo A es algebraico si, para todo $x \in A$, el conjunto $M = \{a \in \mathsf{K}(A) \mid a \sqsubseteq x\}$ es dirigido y $\sqcup M = x$.

Base Dado un cpo A, un conjunto A_0 de elementos compactos de A forman una base si, para todo $x \in A$, el conjunto $M = \{a \in A_0 \mid a \sqsubseteq x\}$ es dirigido y $\sqcup M = x$.

Pre-orden Un pre-orden es un conjunto A dotado de una relación binaria $\widetilde{\sqsubseteq}_A$ que es reflexiva y transitiva.

Ideal Un ideal sobre un pre-orden $(A, \widetilde{\sqsubseteq})$ es un subconjunto $x \in A$ tal que

1. Si $u \subseteq x$ es finito, entonces existe un $a \in x$ tal que $b \mathrel{\widetilde{\sqsubseteq}} a$ para cada $b \in u$
2. Si $a \in x$, entonces $\downarrow a = \{b \mid b \mathrel{\widetilde{\sqsubseteq}} a\} \subseteq x$

El conjunto de los ideales sobre el pre-orden $(A, \widetilde{\sqsubseteq})$ se escribe $\mathsf{Idl}(A, \widetilde{\sqsubseteq})$

Ideal principal El conjunto $\downarrow a = \{b \mid b \mathrel{\widetilde{\sqsubseteq}} a\}$ es llamado el ideal principal generado por a. Se dice que el conjunto parcialmente ordenado *inducido por* A es el conjunto de ideales principales $\downarrow a$ ordenados por inclusión.

1.3.3. Dominios

Dominios En principio consideraremos que un dominio es un cpo; escribiremos D para un dominio y omitiremos el orden $\sqsubseteq$ si este es claro. Algunos dominios tienen un conjunto de elementos de error E; es decir serán de la forma $D + E$ aún cuando a menudo omitamos la E en la definición.

Dominios planos Dado un conjunto arbitrario S se llama dominio plano al cpo $(S_\perp, \sqsubseteq)$ con $S_\perp = S \bigcup \{\perp\}$ y $a \sqsubseteq b$ ssi $a = \perp$. A los elementos de S se les llama propios.

Suma Si D_1 y D_2 son dos dominios, entonces $D_1 + D_2$ es el dominio suma de D_1 y D_2. Los elementos de $D_1 + D_2$ son de la forma $\{(1, x) | x \in D_1\} \bigcup \{(2, x) | x \in D_2\}$ con $(1, \perp_{D_1}) = (2, \perp_{D_2}) = \perp_{D_1+D_2}$. Para extender $\sqsubseteq$ a la suma acordamos que $(i, x) \sqsubseteq (j, y)$ sii $i = j \wedge x \sqsubseteq y$. Si $D = D_1 + D_2$ y $x \in D_i$, entonces $inD(x)$ denota a (i, x) la inyección de x en D. Si $z = (i, y) \in D$ el predicado $z \in D_j$ es cierto ssi $i = j$.

Producto $D_1 \times D_2$ es el dominio producto de D_1 y D_2 decimos que $(x_1, x_2) \sqsubseteq (y_1, y_2)$ ssi $x_1 \sqsubseteq y_1$ y $x_2 \sqsubseteq y_2$. Si $x \in D_1 \times D_2$ entonces $x = (fst(x), snd(x))$.

Continuidad Una función $f \in D_1 \to D_2$ es continua si para todo subconjunto dirigido $S \subseteq D_1$ se cumple que $f(\bigsqcup S) = \bigsqcup f(S)$, donde $f(S) = \{f(s) | s \in S\}$

Monotonía Una función f es monótona si cuando $x \sqsubseteq y$ entonces $f(x) \sqsubseteq f(y)$. Toda función continua es monótona. Además si D_1 es un dominio plano, entonces cualquier función monótona $f \in D_1 \to D_2$ es continua.

Estricta Una función $f \in D_1 \to D_2$ es estricta si $f(\perp_{D_1}) = \perp_{D_2}$. Cualquier función estricta definida sobre un dominio plano es monótona y continua

Producto estricto Sea R una relación definida sobre $D_1 \times D_2$ tal que

$$(x_1, x_2) R (y_1, y_2) \text{ sii } (x_1 = \perp_{D_1} \vee x_2 = \perp_{D_2}) \wedge (y_1 = \perp_{D_1} \vee y_2 = \perp_{D_2})$$

El dominio $D_1 \times_s D_2 = (D_1 \times D_2)/R$ es producto estricto de D_1 y D_2

Utilizando los operadores $+, \times, \to$ se puede formar expresiones sobre dominios, definiendo de esta manera nuevos dominios. Se usa la convención que $\to$ es asociativo por la derecha, y que tanto $+$ como $\times$ son asociativos.

Dominios reflexivos Las ecuaciones recursivas de dominios dan lugar a la definición de dominios reflexivos. Estos, como por ejemplo las listas. dan lugar a cadenas cuyos límites son cadenas infinitas. Utilizando entonces el producto estricto se puede evitar estas situaciones en general no deseadas.

1.3.4. Notaciones

Condicionales El condicional se define estricto en la condición

$$\textbf{if } E_0 \Rightarrow E_1 \,\square\, E_2 = \begin{cases} \perp & \text{si } E_0 = \perp \\ E_1 & \text{si } E_0 = true \\ E_2 & \text{si } E_0 = false \end{cases}$$

El condicional estricto con una sola rama se define como

$$\textbf{if } E_0 \Rightarrow E_1 = \begin{cases} \bot & \text{si } E_0 = \bot \\ E_1 & \text{si } E_0 = true \\ \bot & \text{si } E_0 = false \end{cases}$$

Por lo que ambas versiones del condicional son continuas

Listas Ya que listas (o secuencias) sobre dominios aparecen con frecuencia se introduce un operador especial. Si D es un dominio entonces D^* se define como $D^* = D + D \times_s D^*$. Se escribe $\langle\rangle$ para la secuencia vacía, $\langle x_1, \ldots, x_n\rangle$ es la secuencia de elementos $x_1, \ldots, x_n$. Las funciones hd y tl producen la cabeza y la cola de una secuencia y $\bot$ si se aplican a $\langle\rangle$.

Punto fijo Si $f \in D \to D$, entonces $fix\ f$ denota al menor punto fijo de f, tal que es el menor elemento de $\{g \mid g = f(g)\}$. Si f es monótona el menor punto fijo existe; y si además f es continua el menor punto fijo fijo viene dado por $fix\ f = \bigsqcup\{f_i\}$ donde $f_0 = \bot$, $f_{i+1} = f(f_i)$.

1.3.5. Dominios potencia

Conjunto de los subconjuntos Para cualquier conjunto S, sea $\mathcal{P}^*_f(S)$ el conjunto de los subconjuntos finitos no vacíos de S.

Pre-ordenes en A Dado un conjunto parcialmente ordenado $(A, \subseteq)$, definimos un pre-orden sobre $\mathcal{P}^*_f(A)$ como sigue:

1. $u \sqsubseteq_1 v$ si y sólo si $(\forall x \in u)(\exists y \in v)$ tal que $x \sqsubseteq y$
2. $u \sqsubseteq_2 v$ si y sólo si $(\forall y \in v)(\exists x \in u)$ tal que $x \sqsubseteq y$
3. $u \sqsubseteq_3 v$ si y sólo si $u \sqsubseteq_1 v$ y $u \sqsubseteq_2 v$

Dominios potencia Si D es un dominio, entonces sea $\mathsf{C}(D)$ es el dominio de ideales sobre $(\mathcal{P}^*_f(\mathsf{K}(D)), \sqsubseteq_3)$. Llamaremos a $\mathsf{C}(D)$ el dominio potencia (*powerdomain*) convexo de D. Similarmente se define el dominio potencia superior $\mathsf{U}(D)$ y el dominio potencia inferior $\mathsf{L}(D)$

1.4. Teoría de conjuntos borrosos

Los conjuntos borrosos fueron introducidos por Zadeh a mediados de los 60, [85], como una generalización del concepto clásico de subconjunto, cuando, por causas diversas, existen elementos, cuya pertenencia a un conjunto no está clara.

1.4.1. Definiciones básicas

Conjunto borroso Un conjunto borroso (o difuso) $\widetilde{A}$ sobre un universo del discurso Ω es un conjunto de pares

$$\widetilde{A} = \{\mu_A(x)/x \mid x \in \Omega, \mu_A(x) \in [0,1]\}$$

donde $\mu_A(x)$ es llamado el grado de pertenencia del elemento x al conjunto borroso $\widetilde{A}$.

Si el universo del discurso Ω es finito, un conjunto borroso $\widetilde{A}$ se puede representar por extensión como

$$\widetilde{A} = \mu_1/x_1 + \mu_2/x_2 + \ldots + \mu_n/x_n$$

donde μ_i representa el grado de pertenencia del elemento x_i, con $i = 1, 2, \ldots, n$ y donde el signo "+" se toma en el sentido de la agregación.

Si el universo del discurso Ω es infinito, un conjunto borroso $\widetilde{A}$ se puede representar por compresión como

$$\widetilde{A} = \int \mu_{\widetilde{A}}(x)/x$$

donde $\mu_{\widetilde{A}}(x)$ es llamada también función característica

Igualdad Dos conjuntos borrosos $\widetilde{A}$ y $\widetilde{B}$ sobre Ω se dicen que son iguales y se representa por $\widetilde{A} = \widetilde{B}$ sii cumplen:

$$\forall x \in \Omega, \mu_{\widetilde{A}}(x) = \mu_{\widetilde{B}}(x)$$

Inclusión Dados dos conjuntos borrosos $\widetilde{A}$ y $\widetilde{B}$ sobre Ω se dicen que $\widetilde{A}$ está incluido en $\widetilde{b}$ y se representa por $\widetilde{A} \subseteq \widetilde{B}$ sii se cumple que:

$$\forall x \in \Omega, \mu_{\widetilde{A}}(x) \leq \mu_{\widetilde{B}}(x)$$

Altura La altura (height) de un conjunto borroso $\widetilde{A}$ definido sobre Ω se define como:

$$hgt(\widetilde{A}) = \sup_{x \in \Omega} \mu_{\widetilde{A}}(x)$$

Normalización Un conjunto borroso $\widetilde{A}$ definido sobre Ω se dice normalizado sii:

$$\exists x \in \Omega, \mu_{\widetilde{A}}(x) = hgt(\widetilde{A}) = 1$$

Soporte El soporte (support) de un conjunto borroso $\widetilde{A}$ definido sobre Ω es un subconjunto de dicho universo que verifica:

$$supp(\widetilde{A}) = \{x \in \Omega, \mu_{\widetilde{A}}(x) > 0\}$$

Núcleo El núcleo (kernel) de un conjunto borroso $\widetilde{A}$ definido sobre Ω es un subconjunto de dicho universo que verifica:

$$kern(\widetilde{A}) = \{x \in \Omega, \mu_{\widetilde{A}}(x) = 1\}$$

1.4.2. Operaciones sobre conjuntos borrosos

Unión Si $\widetilde{A}$ y $\widetilde{B}$ son dos conjuntos borrosos sobre el universo del discurso Ω, la función de pertenencia de la unión de ambos conjuntos, $\widetilde{A} \cup \widetilde{B}$, viene dada por:

$$\mu_{\widetilde{A} \cup \widetilde{B}} = f(\mu_{\widetilde{A}}(x), \mu_{\widetilde{B}}(x)) \forall x \in \Omega$$

donde f es una t-conorma

Intersección Si $\widetilde{A}$ y $\widetilde{B}$ son dos conjuntos borrosos sobre el universo del discurso Ω, la función de pertenencia de la intersección de ambos conjuntos, $\widetilde{A} \cap \widetilde{B}$, viene dada por:

$$\mu_{\widetilde{A} \cap \widetilde{B}} = g(\mu_{\widetilde{A}}(x), \mu_{\widetilde{B}}(x)) \forall x \in \Omega$$

donde g es una t-norma

Existen varios operadores que satisfacen los conceptos de t-norma y t-conorma. Algunos de los más interesantes son:

Idempotentes . El máximo y mínimo para la unión e intersección respectivamente. Son distributivas y estrictamente crecientes.

Arquimedianos . La suma probabilística $(x+y-x*y)$, y el producto, $(x*y)$, para la unión e intersección. No son idempotentes ni distributivos.

Acotados . $\min(1, x+y)$ y $\max(0, x+y-1)$, para la unión y la intersección. No son distributivos

Complementación Una función C de $[0,1]$ en $[0,1]$ es una negación fuerte si satisface las siguientes condiciones:

1. $C(0) = 1$
2. $C(C(x)) = x$ (involución)
3. C es estrictamente decreciente
4. C es continua

[81], [32], [92], [3]

1.4.3. Estructura del conjunto de los subconjuntos borrosos de X

Sea $\mathcal{P}(X)$ el conjunto de los subconjuntos nítidos de X. $\mathcal{P}(X)$ es un retículo booleano para $\cup$ y $\cap$. Su estructura puede verse como inducida de la de {0,1}, que es un caso trivial de retículo booleano.

Sea $\widetilde{\mathcal{P}}\ (X)$ el conjunto de subconjuntos borrosos de X. Su estructura puede inducirse de la de el intervalo real $[0,1]$. $[0,1]$ es un retículo pseudo-complementado distributivo. $\widetilde{\mathcal{P}}\ (X)$, puede considerarse como el conjunto de aplicaciones de X en $[0,1]$. Se tienen las siguiente propiedades para $\cup$, $\cap$, $^-$:

1. Conmutatividad: $A \cup B = B \cup A$; $A \cap B = B \cap A$
2. Asociatividad: $A \cup (B \cup C) = (A \cup B) \cup C$; $A \cap (B \cap C) = (A \cap B) \cap C$
3. Idempotencia: $A \cup A = A$; $A \cap A = A$
4. Distributividad: $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$; $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
5. $A \cap \emptyset = \emptyset$; $A \cup X = X$
6. Identidad: $A \cup \emptyset = A$; $A \cap X = A$
7. Absorción: $A \cup (A \cap B) = A$; $A \cap (A \cup B) = A$
8. Leyes de Morgan: $\overline{(A \cap B)} = \overline{A} \cup \overline{B}$; $\overline{(A \cup B)} = \overline{A} \cap \overline{B}$
9. Involución: $\overline{\overline{A}} = A$
10. Equivalencia: $(\overline{A} \cup B) \cap (A \cup \overline{B}) = (\overline{A} \cap \overline{B}) \cup (A \cap B)$
11. Diferencia simétrica: $(\overline{A} \cap B) \cup (A \cap \overline{B}) = (\overline{A} \cup \overline{B}) \cap (A \cup B)$

Las únicas leyes que no se verifican son las del tercero excluido:

$$A \cap \overline{A} \neq \emptyset;\ A \cup \overline{A} \neq X$$

Ya que el conjunto borroso A no tiene cotas definidas y tampoco las tiene $\overline{A}$, puede considerarse natural que A y $\overline{A}$ se solapen. Sin embargo el solapamiento siempre está limitado ya que

$$\forall A, \forall x, \min(\mu_A(x), \mu_{\overline{A}}(x)) \leq \frac{1}{2}$$

Por la misma razón. $A \cup \overline{A}$ no cubre a X; sin embargo

$$\forall A, \forall x, \max(\mu_A(x), \mu_{\overline{A}}(x)) \geq \frac{1}{2}$$

1.4.4. Aritmética borrosa

Existen múltiples definiciones de número borroso. Una de las más extendidas, es considerar como tal a cualquier conjunto borroso de la recta real que sea normal acotado y convexo. La aritmética sobre tales números está basada en el

Principio de de extensión de Zadeh Si A y B son números borrosos y C depende funcionalmente de ellos por medio de f, entonces

$$\mu_C(x) = \sup_{z=f(x,y)} min(\mu_A(x), \mu_B(y))$$

pero su aplicación es costosa y problemática [10], [36], [29], [44]. Por ello en esta tesis haremos uso de un caso particular de los nḿeros LR [23], por dos razones:

- los números triangulares y trapezoidales son suficientes para modelar la mayoría de los casos en que hay que tratar la imprecisión
- su eficiencia computacional

Para un tratamiento más completo ver [53].

Número borroso Sea A un subconjunto borroso de Ω y μ_A su función de pertenencia tales que:

1. $\forall x, y \in \Omega, \mu_A(t) \geq \min\{\mu_A(x), \mu_A(y)\}$
2. μ_A es semi-continua superiormente
3. el soporte de A es un conjunto acotado

entonces diremos que A es un número borroso

La forma general de la función de pertenencia de un número borroso A normalizado, es la siguiente:

$$\mu_A(x) = \begin{cases} r_A(x) & \text{si } x \in [a_1, a_2) \\ 1 & \text{si } x \in [a_2, a_3] \\ s_A(x) & \text{si } x \in (a_3, a_4] \\ 0 & \text{en otro caso} \end{cases}$$

donde $r_A, s_A : \Omega \to [0, 1]$, siendo r_A no decreciente y s_A no creciente.

Números triangulares y trapezoidales Un caso, especialmente interesante, es considerar r_A y s_A como funciones lineales, se obtienen así los números trian-

gulares y trapezoidales, cuya función de pertenencia es:

$$\mu_A(x) = \begin{cases} \dfrac{x - a_1}{a_2 - a_1} & \text{si } x \in [a_1, a_2) \\ 1 & \text{si } x \in [a_2, a_3] \\ \dfrac{a_4 - x}{a_4 - a_3} & \text{si } x \in (a_3, a_4] \\ 0 & \text{en otro caso} \end{cases}$$

Operaciones Dados dos números trapezoidales $A = (a_1, a_2, a_3, a_4)$ y $B = (b_1, b_2, b_3, b_4)$ un resultado aproximado será:

Suma $= (a_1 + b_1, a_2 + b_2, a_3 + b_3, a_4 + b_4)$

Resta $= (a_1 - b_4, a_2 - b_3, a_3 - b_2, a_4 + b_1)$

Multiplicación $= (a_1 * b_1, a_2 * b_2, a_3 * b_3, a_4 * b_4)$

División $= (a_4/b_1, a_3/b_2, a_2/b_3, a_1/b_4)$

No se contempla la comparación de números borrosos.

1.4.5. Relaciones y producto cartesiano de conjuntos borrosos

Relación borrosa Sean Ω_1 y Ω_2 dos referenciales, se llama relación borrosa a un conjunto borroso de $\Omega_1 \times \Omega_2$. La relación borrosa R tiene por función de pertenencia $\mu_R = \pi$, la cual tiene por argumentos $\omega_1 \in \Omega_1$, y $\omega_2 \in \Omega_2$.

Distribución marginal de posibilidad Es la proyección de R sobre Ω_i con $i = 1, 2$. En [90] viene definida por:

$$\pi_i(\omega_i) = \sup_{\omega_j} \pi(\omega_1, \omega_2)$$

Extensiones Sean F_1 y F_2 dos conjuntos borrosos sobre los referenciales Ω_1 y Ω_2 respectivamente, se puede extender μ_{F_i} a $\Omega_1 \times \Omega_2$ por:

$$\forall \omega_1, \omega_2, \mu_{C_j(F_i)}(\omega_1, \omega_2) = \mu_{F_i}(\omega_i)$$

$C_j(F_i)$ es la extensión cilíndrica de F_i a Ω_j

Producto y Coproducto Sean F_1 y F_2 son dos conjuntos borrosos sobre Ω_1 y Ω_2 respectivamente. Se puede extender la noción de producto y coproducto cartesiano a F_1 y F_2, definiéndolos como la intersección (resp: la unión) de sus extensiones cilíndricas:

$$\mu_{F_1 \times F_2}(\omega_1, \omega_2) = \min(\mu_{F_1}(\omega_1), \mu_{F_2}(\omega_2))$$

$$\mu_{F_1 + F_2}(\omega_1, \omega_2) = \sup(\mu_{F_1}(\omega_1), \mu_{F_2}(\omega_2))$$

Proyecciones Si $Proy_i(R)$ es la proyección de R sobre Ω_i entonces

$$R \subseteq Proy_1(R) \times Proy_2(R)$$

Separabilidad Una relación borrosa R se dice que es separable si y solamente si

$$R = Proy_1(R) \times Proy_2(R)$$

Si F_1 y F_2 son dos conjuntos borrosos normalizados sobre Ω_1 y Ω_2 respectivamente, entonces la mayor relación borrosa R tal que $F_i = Proy_i(R)$ es $F_1 \times F_2$.

Conjuntos borroso no interactivos Sean X_1 y X_2 variables tales que el dominio de variación de (X_1, X_2) esté delimitado por R, y sea F_i es el conjunto borroso de valores posibles (sobre Ω_i) de X_i, si

$$\pi_{X_1,X_2} = \min(\mu_{F_1}, \mu_{F_2})$$

se dice que los conjuntos borrosos F_1 y F_2 son no interactivos y que las variables X_1 y X_2 no están ligadas

1.5. Trabajos previos

Muchos investigadores han dedicado sus trabajos a estudiar las estructuras formales borrosas. Generalmente han partido de estructuras no borrosas ya existentes, modificándolas para dar cuenta del paradigma borroso. Presentamos algunas de ellas, prestando especial atención a los trabajos fundacionales, es decir a aquellos que han iniciado una vía que posteriormente ha sido desarrollada en multitud de variantes y refinamientos.

1.5.1. Lógica borrosa y el principio de resolución (Lee 1972)

La lógica borrosa puede definirse como un sistema algebraico $\langle [0,1], \bigwedge, \bigvee, \breve{} \rangle$, donde el intervalo cerrado $[0,1]$ es el conjunto de valores de verdad y donde las operaciones lógicas $AND(\bigwedge)$, $OR(\bigvee)$ y $NOT(\breve{})$ se definen de la siguiente forma:

- A $\bigwedge$ B = min(A,B)
- A $\bigvee$ B = max(A,B)
- $\breve{\text{A}}$ (o not(A)) = 1 - A

con A,B$\in [0,1]$

Definición 1.1 *Una variable* x_i $(i = 1, \ldots, n)$ *o su negación* $\breve{x}_i$*, es un* literal*, y el par* x *y* $\breve{x}_i$ *se dice que son complementarios o que forman un par de variables complementarias. Una* cláusula *es una colección de literales (implícitamente unidos por disyunciones). Una sentencia en* forma clausulada *es una colección de cláusulas (implícitamente unidas por conjunciones). Una cláusula con un único literal se la llama cláusula unaria. Una cláusula sin literales de representa por* [ ] *y se dice que es la cláusula vacía*

Definición 1.2 *Las fórmulas borrosas se definen recursivamente como sigue:*

1. *Una variable es una fórmula borrosa.*
2. *Si* F *es una fórmula borrosa, entonces* $\breve{F}$ *es una fórmula borrosa.*
3. *Si* F *y* G *son fórmulas borrosas, entonces* $F \bigvee G$ *y* $F \bigwedge G$ *son fórmulas borrosas.*
4. *Las anteriores son las únicas fórmulas borrosas.*

Dada una interpretación I, el valor de verdad de una cláusula C se determina de forma única sustituyendo un valor del intervalo cerrado $[0, 1]$ determinado por la interpretación I para cada variable de la cláusula. Se representa por $\mathrm{T_I(C)}$. Para un conjunto S de cláusulas el valor de verdad de S bajo una interpretación I se escribe $\mathrm{T_I(S)}$, donde si $\mathrm{S} = \{\mathrm{C}_1, \ldots, \mathrm{C}_n\}$, entonces $\mathrm{T_I(S)} = \min\ \mathrm{T_I}(\mathrm{C}_1, \ldots, \mathrm{C}_n) = \min\ (\mathrm{T_I}(\mathrm{C}_1), \ldots, \mathrm{T_I}(\mathrm{C}_n))$.

En lógica borrosa, las leyes del complementario ($\mathrm{T}(\breve{\mathrm{x}}) \bigvee \mathrm{T}(\mathrm{x}) = 1$, $\mathrm{T}(\breve{\mathrm{x}}) \bigwedge \mathrm{T}(\mathrm{x}) = 0$ para todas las interpretaciones) no son válidas. Por tanto una cláusula que contiene simultáneamente a x y a $\breve{\mathrm{x}}$ puede ser significativa en lógica borrosa. Llamaremos a tales cláusulas *cláusulas complementarias*.

Definición 1.3 *Si* x_i *y* $\breve{\mathrm{x}}_i$ *son un par de variables complementarias e* I *una interpretación, entonces se dice que* $\mathrm{x}_i \bigwedge \breve{\mathrm{x}}_i$ *es una* contradicción *en la interpretación* I*. Si* $\mathrm{T_I}(\mathrm{x}_i \bigwedge \breve{\mathrm{x}}_i) = 0$ *entonces se dice que son* contradictorios completos*. Si* $\mathrm{T_I}(\mathrm{x}_i \bigwedge \breve{\mathrm{x}}_i) = 0.5$ *entonces se dice que son* no contradictorios*. Si* $\mathrm{T_I}(\mathrm{x}_i \bigwedge \breve{\mathrm{x}}_i) \in (0, 0{,}5)$ *entonces se dice que son* contradictorios incompletos

Definición 1.4 *El grado de contradicción de* $\mathrm{x}_i \bigwedge \breve{\mathrm{x}}_i$ *se representa por* $\mathbf{gc}(\mathrm{x}_i)$ *y bajo una interpretación* I

$$\mathbf{gc}(\mathrm{x}_i) = \max\ (\mathrm{T_I}(\mathrm{x}_i), \mathrm{T_I}(\breve{\mathrm{x}}_i))\ \text{-}\ \min\ (\mathrm{T_I}(\mathrm{x}_i), \mathrm{T_I}(\breve{\mathrm{x}}_i))$$

Definición 1.5 *El valor de verdad de una contradicción* $\mathrm{x}_i \bigwedge \breve{\mathrm{x}}_i$ *es igual al "valor de verdad" de su grado de contradicción* $\mathbf{gc}(\mathrm{x}_i)$ *y se define como sigue:*

$$\begin{aligned} \mathrm{T}(\mathbf{gc}(\mathrm{x}_i)) &= (\text{-}\mathbf{gc}(\mathrm{x}_i)) \times 0{,}5 + 0{,}5 \\ &= \min\ (\mathrm{T_I}(\mathrm{x}_i), \mathrm{T_I}(\breve{\mathrm{x}}_i)) \\ &= \mathrm{T_I}(\mathrm{x}_i \bigwedge \breve{\mathrm{x}}_i) \end{aligned} \tag{1.1}$$

para todas las interpretaciones

Definición 1.6 *Si* $C_1 = x_i \bigvee L_1$ *y* $C_2 = \breve{x}_i \bigvee L_2$ *son dos cláusulas donde* L_1 *y* L_2 *no contienen como factores a los literales* x_1, $\breve{x}_i$ *y no tienen pares de variables complementarias, entonces la cláusula* $L_1 \bigvee L_2$ *se dice que es un* resolvente *de* C_1 *y* C_2 *cuya clave es* x_i*, y el grado contradictorio de la clave es* $\mathbf{gc}(x_i)$*. Un resolvente de* C_1 *y* C_2 *es escribe* $R(C_1,C_2)$ *y un* resolvente borroso *de* C_1 *y* C_2 *se escribe como* $R(C_1,C_2)_{\mathbf{gc}}$*, donde* $\mathbf{gc} = \mathbf{gc}(x_i)$ *es el grado de contradicción de la clave o se dice que es la* confianza del resolvente *de* $R(C_1,C_2)$

Definición 1.7 *El valor de verdad de un resolvente borroso* $R(C_1,C_2)_{\mathbf{gc}}$ *es:*

$$T(R(C_1,C_2)_{\mathbf{gc}}) = T(R(C_1,C_2)) \bigvee T(\mathbf{gc})$$

para todas la interpretaciones

Cuando A, B, C, D $\in [0, 1]$ las siguientes reglas de inferencia:

modus ponens: Si A y A $\rightarrow$ B, entonces B

modus tollens: Si $\breve{\text{B}}$ y A $\rightarrow$ B, entonces $\breve{\text{A}}$

silogismo disyuntivo: Si A $\bigvee$ B y $\breve{\text{A}}$, entonces B

silogismo hipotético: Si A $\rightarrow$ B y B $\rightarrow$ C, entonces A $\rightarrow$ C

dilema constructivo: Si (A $\rightarrow$ B) $\bigwedge$ (C $\rightarrow$ D) y A $\bigvee$ C, entonces B $\bigvee$ D

dilema destructivo: Si (A $\rightarrow$ B) $\bigwedge$ (C $\rightarrow$ D) y $\breve{\text{B}} \bigvee \breve{\text{D}}$, entonces $\breve{\text{A}} \bigvee \breve{\text{C}}$

son casos especiales de la inferencia borrosa 1.1

Definición 1.8 *Una interpretación* I *se dice que satisface a una fórmula* S *si* $T_I(S) \geq 0{,}5$*. Una interpretación* I *se dice que falsea una fórmula* S *si* $T_I(S) \leq 0{,}5$*.*

Definición 1.9 *Una fórmula se dice que es* insatisfacible *si y solo si es falsable en todas sus interpretaciones.*

Definición 1.10 *Sea S un conjunto de cláusulas. La resolución de S, representada por R(S), es el conjunto por S junto con los resolventes de todos los pares. La n-ésima resolución de S,* $R^n(S)$*, se define para* $n \geq 0$ *de la siguiente forma*

$$R^0(S) = S \text{ y } R^{n+1}(S) = R(R^n(S))$$

Teorema 1.1 *Sea* $C_1, C_2, \ldots, C_m \in S$ *un conjunto de cláusulas. Sea*

$$\max[T(C_1), T(C_2), \cdots, T(C_m)] = b$$

y

$$\min[T(C_1), T(C_2), \cdots, T(C_m)] = a$$

Sea C^n *cualquier cláusula del conjunto* $R^n(S)$*. Entonces* $\forall n \geq 0$ $a \leq T(C^n) \leq b$

Este resultado garantiza que el valor de verdad de cualquier consecuencia lógica obtenida por aplicación reiterada del principio de resolución está comprendido entre los valores de verdad de la cláusula más confiable y la menos confiable. Para más detalles consultar [48]

1.5.2. Máquinas de Turing borrosas (Santos, 1970)

Desde el punto de vista clásico se tiene:

Definición 1.11 *Una máquina de Turing M es una quíntupla $M = (S, X, q_0, q_a, \delta)$ donde*

S es un conjunto finito de estados

X es un alfabeto. Se escribe $X' = X \cup \{B\}$ para representar al conjunto X aumentado por el símbolo B (el blanco)

s_0 es el estado inicial

s_a es el estado de aceptación

δ es la función parcial de transición

$$\delta : S \times X' \to (X' \cup \{-1, 0, +1\}) \times S$$

con la condición de que $\delta(s_a, x)$ no está definida para cualquier $x \in X'$.

δ se puede representar por una lista de cuádruplas, a la que pertenecería $(s\ x\ x'\ s')$ si y sólo si $\delta(s, x) = (x', s')$. Lo que se hace de esta forma es considerar una aplicación $p : S \times X' \times (X' \cup \{-1, 0, +1) \times S\} \to \{true, false\}$, en la lista solo aparecerán los elementos cuya imagen sea true. Para definir una máquina de Turing Borrosa (MTB) se modifica p para que su imagen $\in [0, 1]$. Con ello tenemos la siguiente definición:

Definición 1.12 *Una máquina de Turing Borrosa (MTB) M es una quíntupla $M = (S, X, q_0, q_a, p)$ donde*

S es un conjunto finito de estados

X es un alfabeto. Se escribe $X' = X \cup \{B\}$ para representar al conjunto X aumentado por el símbolo B (el blanco)

s_0 es el estado inicial

s_a es el estado de aceptación

p es la función

$$p : S \times X' \times (X' \cup \{-1, 0, +1\}) \times S \to [0, 1]$$

con la condición de que $p(s_a, x)$ no está definida para cualquier $x \in X'$.

p puede interpretarse como el grado de pertenencia de la "siguiente actuación" de la máquina a partir de estado actual y del símbolo examinado.

Definición 1.13 *Sea M una MTB. Una* **descripción instantánea** *(DI) de M es una cadena w_1sw_2 del conjunto $(X')^* \cdot S \cdot (X')^*$. Cada DI w_1sw_2 es equivalente a los DI Bw_1sw_2 y w_1sw_2B*

Definición 1.14 *Sea M una máquina de Turing. Para todo par de DI α y β de M, se define:*

$$q_M(\alpha,\beta) = \begin{cases} p(s,x,x',s') & si \quad \alpha = \sigma sx\tau,\ \beta = \sigma s'x'\tau,\ x' \in X \\ p(s,x,+1,s') & si \quad \alpha = \sigma sxx'\tau,\ \beta = \sigma xs'x'\tau,\ x' \in X \\ & o \quad \alpha = \sigma sx,\ \beta = \sigma xs'B \\ p(s,x,-1,s') & si \quad \alpha = \sigma x'sx\tau,\ \beta = \sigma s'x'x\tau,\ x' \in X \\ & o \quad \alpha = sx\tau,\ \beta = s'Bx\tau \\ 0 & \text{en caso contrario} \end{cases}$$

donde σ y τ pueden ser vacíos.

La función $q_M(\alpha,\beta)$ puede interpretarse como el grado de pertenencia que la "siguiente" DI es β siendo la DI "actual" α, y puede extenderse a $q_M^{(n)}(\alpha,\beta)$, como sigue:

$$q_M^{(0)}(\alpha,\beta) = \begin{cases} 1 & \text{si } \alpha = \beta \\ 0 & \text{si } \alpha \neq \beta \end{cases}$$

$$q_M^{(n+1)}(\alpha,\beta) = \bigsqcup_{\gamma} \{Min[q_M^{(n)}(\alpha,\gamma), q_M(\gamma,\beta)]\}$$

donde $\bigsqcup$ (la menor de las cotas superiores) se toma sobre todas las DI de M. $q_M^{(n)}(\alpha,\beta)$ puede interpretarse como el grado de pertenencia que "tras n pasos" la DI es β siendo la DI "actual" α.

Definición 1.15 *Sea M una máquina de Turing. Para todo par de DI α y β de M, se define:*

$$t_M^{(n)}(\alpha,\beta) = Min[p(s,x,0,s), q_M^{(n-1)}(\alpha,\beta)]$$

donde s es un estado de M en β y x es el símbolo analizado por M en α. Además, se define:

$$t_M(\alpha,\beta) = \bigsqcup_{n} t_M^{(n)}(\alpha,\beta)$$

$t_M^{(n)}(\alpha,\beta)$ puede interpretarse como el grado de pertenencia que "tras n pasos" la máquina "termina" con β suponiendo que comienza con α.

$t_M(\alpha,\beta)$ puede interpretarse como el grado de pertenencia que "tras número finito de pasos" la máquina "termina" con β suponiendo que comienza con α.

Las máquinas de Turing pueden extenderse (varias cintas, varias cabezas, ...) para facilitar el manejo de algunas problemas, sin que cambie su potencia expresiva. Desde el punto de vista teórico, algunas construcciones y demostraciones se realizan usando las máquinas de Turing, ellas permiten una definición del número de "pasos" y "celdas" necesarias para un cálculo, y por tanto de la complejidad temporal y espacial de un algoritmo, por lo que las máquinas de Turing dan una descripción convincente de la computación, pero sin embargo su influencia en el diseño de los lenguajes de programación es muy limitado, por lo que no son muy útiles para los objetivos de esta tesis. En [13] se estudian las propiedades de las funciones borrosas con la ayuda de de máquinas de Turing borrosas.

1.5.3. Lenguajes Borrosos (Lee and Zadeh, 1969)

Sea V_T un conjunto finito llamado *alfabeto*. Representaremos por V_T^* el conjunto de cadenas finitas construidas por concatenación de elementos de V_T, incluyendo la cadena vacía Λ. V_T^* es un monoide libre sobre V_T. Un lenguaje es un subconjunto de V_T^*. Un *lenguaje formal borroso* es un subconjunto borroso $\mathcal{L}$ de V_T^*, decir:

$$\mathcal{L} = \sum_{x \in V_T^*} \mu_{\mathcal{L}}(x)/x$$

siendo $\mu_{\mathcal{L}}$ una función de V_T^* a $[0,1]$. $\mu_{\mathcal{L}}(x)$ es el grado de pertenencia de x a $\mathcal{L}$

Se define la unión e intersección de lenguajes borrosos de la siguiente forma:

$\mathcal{L}_1 \cup \mathcal{L}_2$: $\mu_{\mathcal{L}_1 \cup \mathcal{L}_2} = \max\left(\mu_{\mathcal{L}_1}(x), \mu_{\mathcal{L}_2}(x)\right) \forall x \in V_T^*$

$\mathcal{L}_1 \cap \mathcal{L}_2$: $\mu_{\mathcal{L}_1 \cup \mathcal{L}_2} = \min\left(\mu_{\mathcal{L}_1}(x), \mu_{\mathcal{L}_2}(x)\right) \forall x \in V_T^*$

Y $\widetilde{\mathcal{L}}$, el complementario de $\mathcal{L}$, tiene como función de pertenencia $1 - \mu_{\mathcal{L}}$.

Una operación especifica entre lenguajes es la *concatenación*: cualquier cadena $x \in V_T^*$ es la concatenación de un prefijo u y un sufijo v tal que $x = uv$. De acuerdo con el principio de extensión, la concatenación $\mathcal{L}_1\mathcal{L}_2$ de dos lenguajes borrosos $\mathcal{L}_1$ y $\mathcal{L}_2$ se define por

$$\mu_{\mathcal{L}_1\mathcal{L}_2} = \sup_{x=uv} \min(\mu_{\mathcal{L}_1}(x), \mu_{\mathcal{L}_2}(x))$$

La concatenación de lenguajes borrosos es asociativa. Representamos por $\mathcal{L}^n$ la concatenación de $\mathcal{L}$ n veces. La clausura de Kleene de $\mathcal{L}$ es

$$\widehat{\mathcal{L}} = \{\Lambda\} \cup \mathcal{L} \cup \mathcal{L}^2 \cup \cdots \cup \mathcal{L}^n \cup \cdots$$

Notar que $\forall x \in V_T^*$, si $x = a_1 a_2 \cdots a_k$, $a_i \in V_T$, $i = 1, k$, entonces

$$\mu_{\widehat{\mathcal{L}}} = \sup_{i=1,k} \left[\sup_{\substack{x=u_1u_2\cdots u_j \\ \forall j, u_j \in V_T^*}} \left[\min_{j=1,i} \mu_{\mathcal{L}}(u_j) \right] \right]$$

k es la longitud de x, se representa por $l(x)$.

Gramáticas borrosas

Definición 1.16 *Una gramática borrosa es una cuádrupla $G = (V_N, V_T, P, S)$ donde:*

V_T es un conjunto de terminales o alfabeto

V_N es un conjunto de de no terminales ($V_N \cap V_T = \emptyset$)

P es un conjunto finito de reglas llamadas producciones

$S \in V_N$ es el símbolo inicial

Los elementos de P son expresiones de la forma $\alpha \xrightarrow{\rho} \beta$ donde $\alpha, \beta \in (V_T \cup V_N)^*$ y $\rho \in [0, 1]$ es el grado de pertenencia de β a la derivación de α

Sea $\alpha_1, \ldots, \alpha_m$ cadenas de $(V_T \cup V_N)^*$, y $\alpha_1 \xrightarrow{\rho_2} \alpha_2$, $\alpha_2 \xrightarrow{\rho_3} \alpha_3, \ldots, \alpha_{m-1} \xrightarrow{\rho_m} \alpha_m$ son producciones. Entonces se dice que α_m es *derivable* de α_1 en G, y se representa por $\alpha_1 \Rightarrow_G \alpha_m$

Una gramática borrosa G genera un lenguaje borroso $\mathcal{L}(G)$ de la siguiente forma: una cadena $x \in V_T^*$ se dice que está en $\mathcal{L}(G)$ si y solo si x es derivable desde S. El grado de pertenencia

$$\mu_G(x) = \sup \min(\mu(S \to \alpha_1), \mu(\alpha_1 \to \alpha_2), \ldots, \mu(\alpha_m \to x)) > 0$$

donde para $\mu(\alpha_i \to \alpha_{i+1})$ ρ_{i+1} es no nulo, de forma que:

$$(\alpha_i \xrightarrow{\rho_{i+1}} \alpha_{i+1}) \in P \; \forall i = 0, m$$

Si $\alpha_0 = S$ y $\alpha_{m+1} = x$.

El supremo se toma sobre todas las cadenas de derivación de S a x.

La frase "x es de $\mathcal{L}(G)$" significa $x \in \text{supp}\, \mathcal{L}(G)$. El grado de conveniencia de una cadena de derivación es el menor grado de sus eslabones, y $\mu_G(x)$ se calcula sobre la "mejor" cadena.

Dadas dos gramáticas borrosas G_1 y G_2 se dice que son equivalentes si y sólo si $\forall x \in V_T^*,\ \mu_{G_1}(x) = \mu_{G_2}(x)$.

Paralelamente a la clasificación de las gramáticas ordinarias se pueden distinguir cuatro tipos de gramáticas borrosas:

Gramáticas de tipo 0. Las producciones permitidas son de la forma $\alpha \xrightarrow{\rho} \beta$, donde $\rho > 0$, y $\alpha, \beta \in (V_T \cup V_N)^*$.

Gramáticas de tipo 1 (sensibles al contexto). Las producciones permitidas son de la forma $\alpha_1 A \alpha_2 \xrightarrow{\rho} \alpha_1 \beta \alpha_2$, donde $\rho > 0$, $\alpha_1, \alpha_2, \beta \in (V_T \cup V_N)^*$, $A \in V_N$, $\beta \neq \Lambda$ y $S \xrightarrow{1} \Lambda$.

Gramáticas de tipo 2 (libres de contexto). Las producciones permitidas son de la forma $A \xrightarrow{\rho} \beta$, donde $\rho > 0$, $\beta \in (V_T \cup V_N)^*$, $A \in V_N$, $\beta \neq \Lambda$ y $S \xrightarrow{1} \Lambda$.

Gramáticas de tipo 3 (regulares). Las producciones permitidas son de la forma $A \xrightarrow{\rho} aB$ o $A \xrightarrow{\rho} a$ donde $\rho > 0$, $a \in V_T$, $A, B \in V_N$ y $S \xrightarrow{1} \Lambda$.

Si G es de tipo i, $\mathcal{L}(G)$ se dice que es de tipo i.

Una gramática se dice que es recursiva si y sólo si existe un algoritmo que calcula $\mu_G(x)$

Para más detalles consultar [41]

Traducción borrosa dirigida por la sintaxis(Thomason, 1974)

Una traducción T de un lenguaje $\mathcal{L}_1$ de V_T^* en un lenguaje $\mathcal{L}_2$ de W_T^* donde V_T, W_T son alfabetos, es una relación borrosa de $V_T^* \times W_T^*$ tal que dominio $T = \mathcal{L}_1$ y rango $T = \mathcal{L}_2$. $\mu_T(x, y)$ es el grado de conveniencia de traducir x por y, donde $x \in \mathcal{L}_1$, $y \in \mathcal{L}_2$

Definición 1.17 *Un* esquema borroso de traducción dirigido por la sintaxis *es una 5-tupla* $\mathcal{F}(V_N, V_T, W_T, S, D)$ *donde*

V_N *es el conjunto de no terminales*

V_T *y* W_T *son alfabetos*

S *el símbolo inicial*

D *es un conjunto de producciones dobles* $A \xrightarrow{\rho} \alpha, \beta$ *donde* $A \in V_N$, $(\alpha, \beta) \in (V_T \cup V_N)^* \times (W_T \cup V_N)^*$, *y* $\rho > 0$ *valora la traducción de* α *a* β

Para más detalles consultar [76].

Este método, que es un caso particular de las gramáticas atribuidas, tiene la clara ventaja de que, al menos en principio, es posible generar automáticamente un compilador eficiente a partir de la definición del lenguaje con este método. El

inconveniente es que, de la misma forma que las gramáticas atribuidas, constituyen una descripción del lenguaje más que una abstracción. Este método no define el lenguaje, lo que en realidad hace es relacionarlo con otro (el código máquina o algún código intermedio), ya que describe un proceso de traducción. En general, en las gramáticas atribuidas, la conexión entre un conjunto de atributos y un entendimiento intuitivo del lenguaje es muy remota. En [50] se estudian las propiedades de las funciones borrosas con la ayuda de gramáticas borrosas, además se diseña un lenguaje borroso muy simple, cuya semántica viene dada por medio de atributos.

Lenguajes y Autómatas (Mizumoto *et al.*(1969)

Sea $\mathcal{A} = (U, S, Y, \tilde{s}_0, \delta, \sigma)$ donde:

U es un conjunto finito de entrada

S es un conjunto finito de estados

Y es un conjunto finito de salida

$\tilde{s}_0$ es un conjunto borroso sobre S, el estado inicial borroso

δ es una relación ternaria borrosa sobre $S \times U \times S$. A partir de ella se construye δ_u la relación binaria borrosa entre los estados cuando la entrada es u

σ es una relación borrosa sobre $S \times Y$

Cuando el autómata procesa una entrada no borrosa u, la salida puede escribirse simbólicamente como $\tilde{y} = \tilde{s}_0 \circ \delta_u \circ \sigma$ donde $\circ$ es la composición de relaciones binarias ($\circ$ es asociativa). Una ves que el autómata ha procesado una cadena de entrada $\theta = u_1 u_2 \cdots u_k$, la salida borrosa es:

$$\tilde{y} = \tilde{s}_0 \circ (\delta_{u_1} \circ \delta_{u_2} \circ \cdots \circ \delta_{u_k}) \circ \sigma = \tilde{s}_0 \circ \Delta \circ \sigma$$

Donde Δ_θ es el resultado de la composición $\delta_{u_1} \circ \delta_{u_2} \circ \cdots \circ \delta_{u_k}$. Por medio de Δ_Λ se representa la relación identidad.

A partir de ahora se considera que la salida del autómata está formada por un único elemento y. Lo que el autómata calcula son los valores de pertenencia, es decir $\tilde{y} = f_{\mathcal{A}}(\theta)/y$ donde $f_{\mathcal{A}}$ es la función de respuesta del autómata. Por simplicidad se escribe $f_{\mathcal{A}}(\theta) = \tilde{s}_0 \circ \Delta \circ \sigma$

La imagen de U^* (el monoide libre sobre U formado por todas las cadenas finitas de U) bajo $f_{\mathcal{A}}$ es el lenguaje borroso $\mathcal{L}(\mathcal{A})$. Por tanto, los autómatas borrosos pueden reconocer a los lenguajes borrosos.

Propiedades estructurales de los Autómatas generados por Lenguajes Borrosos

1. Si $\mathcal{A}_1$ y $\mathcal{A}_2$ son autómatas borrosos, $\exists \mathcal{A} = \mathcal{A}_1 \coprod \mathcal{A}_2$ tal que

   $$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$$

   $\mathcal{A}$ es definido por

   $$U = U_1 \cup U_2;\ S = S_1 \cup S_2;\ Y = Y_1 \cup Y_2$$
   $$\delta = \begin{pmatrix} \delta_1 & 0 \\ 0 & \delta_2 \end{pmatrix};\ \tilde{s}_0 = (\tilde{s}_{0_1}, \tilde{s}_{0_2});\ \sigma = \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix}$$

   donde los subíndices 1 y 2 denotan a los componentes de $\mathcal{A}_1$ y $\mathcal{A}_2$ respectivamente.

2. Si $\mathcal{A}_1$ y $\mathcal{A}_2$ son autómatas borrosos, $\exists \mathcal{A} = \mathcal{A}_1 \otimes \mathcal{A}_2$ tal que

   $$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$$

   $\mathcal{A}$ es definido por

   $$U = U_1 \times U_2;\ S = S_1 \times S_2;\ Y = Y_1 \times Y_2$$
   $$\delta = c(\delta_1) \cap c(\delta_2);\ \tilde{s}_0 = c(\tilde{s}_{0_1}) \cap c(\tilde{s}_{0_2});\ \sigma = c(\sigma_1) \cap c(\sigma_2)$$

   donde si R es una relación borrosa sobre $X_1 \times \cdots \times X_n$, $c(R)$ es la extensión cilíndrica sobre $X_1 \times \cdots \times X_n$

3. $\forall \mathcal{A}, \exists \overline{\mathcal{A}}$ tal que:

   $$1 - f_{\overline{\mathcal{A}}}(\theta) = \overline{\tilde{s}_0 \circ \Delta_\theta \circ \sigma} = \overline{\tilde{s}}_0 \,\overline{\circ}\, \overline{\Delta}_\theta \,\overline{\circ}\, \overline{\sigma} = f_{\overline{\mathcal{A}}}(\theta)$$

4. Si $\mathcal{A}_1$ y $\mathcal{A}_2$ son autómatas borrosos, $\exists \mathcal{A} = \mathcal{A}_1 \circ \mathcal{A}_2$ tal que

   $$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1)\mathcal{L}(\mathcal{A}_2) \ \text{ (concatenación)}$$

5. Si $\mathcal{A}$ es un autómata borroso, $\exists \hat{\mathcal{A}}$ tal que

   $$\mathcal{L}(\hat{\mathcal{A}}) = \hat{\mathcal{L}}(\mathcal{A}) \ \text{ /clausura de Kleene)}$$

6. Si $\mathcal{A}_1$ es un autómata borroso, $\exists \mathcal{A}$ tal que

   $$f_A(\theta) = \ \min(a, f_{\mathcal{A}_1}(\theta)),\ \forall \theta \in U^* \text{ y } a \in [0, 1]$$

   $\mathcal{A}$ coincide con $\mathcal{A}_1$ excepto en sus funciones de salida. En efecto si σ_1 es la función de salida de $\mathcal{A}_1$ entonces σ definida por $\forall q_j \in S,\ \mu_\sigma(q_j, y) = \min(a, \mu_{\sigma_1}(q_j, y))$ es la función de salida de $\mathcal{A}$

Autómatas Borrosos y Gramáticas Regulares

Definición 1.18 *Un autómata borroso se dice que es* restringido *cuando su estado inicial no es borroso, y σ es una función clásica de F en $\{y\}$, donde F es un subconjunto no borroso de estados llamado "estados finales", es decir $\mu_\sigma(s_j, y) = 1$ si y sólo si $s_j \in F$*

Cualquier autómata borroso es equivalente a un autómata borroso restringido. En efecto $\mathcal{A} = (U, S, Y, \tilde{s}_0, \delta, \sigma)$ y $\mathcal{A}' = (U, S \cup \{\hat{s}, y\}, Y', \hat{s}, \delta', \sigma')$ son equivalentes si:

$Y' = \{y'\}$. $y' \notin (U \cup S \cup Y)$

$\hat{s} \notin S$, es el estado inicial de $\mathcal{A}'$

δ' se define de la siguiente manera:

$$\mu_{\delta'}(s, a, s') = \mu_\delta(s, a, s')) \ \forall s, s' \in S, \ \forall a \in U$$
$$\mu_{\delta'}(\hat{s}, a, s) = \max_{s' \in S} \min(\mu_{\tilde{s}_0}(s'), \mu_\delta(s', a, s)) \ \forall s \in S$$
$$\mu_{\delta'}(s, a, \hat{s}) = 0 \ \ \forall s \in S \cup \{\hat{s}, y\}$$
$$\mu_{\delta'}(s, a, y) = \max_{s' \in S} \min(\mu_\delta(s, a, s'), \mu_\sigma(s', y)) \ \forall s \in S$$
$$\mu_{\delta'}(y, a, s) = 0 \ \ \forall s \in S \ \text{ o } s = \hat{s}$$
$$\mu_{\delta'}(\hat{s}, a, y) = \max_{s \in S} \min(\mu_{\delta'}(\hat{s}, a, s), \mu_\sigma(s, y))$$

$$\sigma' \text{ es tal que } \mu_{\sigma'}(s, y') = \begin{cases} 0 & \forall s \in S \cup \{\hat{s}\} \\ 1 & \text{si } s = y \end{cases}$$

En [48] se demuestra que dada una gramática regular borrosa G, existe un autómata borroso $\mathcal{A}$ tal que $\mathcal{L}(G) = \mathcal{L}(\mathcal{A})$. En particular, se tiene que, considerando los autómatas como aceptadores de lenguajes, la capacidad de un autómata borroso es igual a la de un autómata no borroso.

1.5.4. Desarrollos

Aunque nuestra tesis no tiene como objeto la implantación de un lenguaje, en la parte final presentaremos el prototipo de un lenguaje imperativo, su finalidad es la de mostrar de manera fehaciente que los razonamientos hechos a lo largo de la misma son realizables. Por su carácter incompleto y experimental no se le da nombre ni se suministra un manual de usuario. Su génesis ha sido muy enriquecedora y señalamos [1] y [2] como inspirador de muchos aspectos del lenguaje protopitado. Como acertadamente se dice en [20] es el trabajo más completo que existe, dentro de la categoría de lo que nosotros hemos llamado dialecto, de un lenguaje imperativo.

Por último, no queremos dejar de nombrar a un buen número de productos que, como muestra de la pujanza de la tecnología borrosa, empresas dedicadas a sistemas de información borrosa han puesto en el mercado. Generalmente estos productos, que hacen un buen uso de gráficos, ventanas etc, sólo suministran, como por otra parte es lógico, manuales de usuario, en esencia sintaxis comentadas de sus lenguajes. La semántica se muestra, a la antigua usanza, por medio del ejemplo. Los hay que generan código C como FUZZ-C de ByteCraft, Ltd., FuzzySoft de FuzzySoft AG o fuzzyTECH 3.0 de Inform Software Corp. Otros toman como base la programación lógica como FRIL de Fril Systems Ltd. Finalmente otros adoptan la forma de hojas de cálculo o se utilizan para acceder a bases de datos como FuziCalc de FuziWare, Inc. o DB-fuzzy de Aria Ltd.

En cualquier caso nuestra aportación a los desarrollos de lenguajes borrosos, pretende introducir la formalidad y el rigor en la definición de dichos lenguajes, que posibiliten en un futuro implementaciones con sólidos fundamentos.

Para ello, en los capítulos siguientes necesitamos introducir las bases formales sobre las que construiremos nuestro lenguaje para a continuación ir progresivamente dotándole de constructores, tipos y operadores que conformen un clásico lenguaje de alto nivel con un sustrato capaz de manejar nociones de borrosidad.

Capítulo 2

El lambda cálculo_b

El objetivo que vamos a abordar en este capítulo es la de añadir, a uno de los sistemas formales para la computabilidad, el λ-cálculo de Church, los elementos mínimos necesarios para dar cuenta de la teoría de Zadeh [85]. Para ello modificaremos el lambda cálculo, consultar página 16, en el siguiente sentido: el alfabeto de dicho cálculo no se cambiará, y además postulamos la existencia de un conjunto no vacío $\mathbb{D}$. Modificaremos el concepto de término exigiendo que todo término esté etiquetado con elementos de $\mathbb{D}$. Las fórmulas construidas por medio de las relaciones de igualdad y reducción serán también nuestras fórmulas ampliadas con los elementos de $\mathbb{D}$. Conservaremos la notación sintáctica, así como, la definición de variable libre y ligada. El cálculo resultante será una variante del lambda cálculo etiquetado de Hyland y Wadsworth [7], en él cada término se etiqueta con un número natural y se utiliza como una herramienta para examinar los λ-modelos D_∞, el espacio de funciones $[D \to D]$ de la misma cardinalidad que D y P_ϖ, el conjunto de los subconjuntos de $\mathbf{N}$ ordenados por inclusión. Posteriormente Levy [43] generaliza el lambda cálculo etiquetado al considerar la etiqueta de cada término como una lista y se utiliza para estudiar la optimización de las reducciones.

Al definir el concepto auxiliar de sustitución, así como al establecer los esquemas de axiomas y reglas, que nos permitan establecer la corrección de una fórmula, deberemos exigir que el conjunto $\mathbb{D}$ esté dotado de ciertas operaciones y con las propiedades necesarias a fin de satisfacer el teorema de Church-Rosser [7], [16]. Este será el primer resultado que obtengamos para $\mathbb{D}$, en el capítulo siguiente obtendremos más resultados.

2.1. Notación

Como en un sistema formal los términos son representables por medio de árboles, una forma sencilla de visualizar nuestro cálculo, al que en adelante llamaremos lambda cálculo-b, será etiquetar las ramas con los elementos de $\mathbb{D}$, de manera que sus objetos tendrán dos partes:

1. La parte-λ perteneciente al λ-cálculo clásico
2. La parte-b será una construcción realizada sobre el conjunto $\mathbb{D}$

Por ello para representar un objeto genérico T, utilizaremos:

- El elemento del λ-cálculo clásico
- La lista sobre $\mathbb{D}$ resultante de recorrer en profundidad el árbol, la representaremos por $\ell(T)$.

Es decir, cualquier término T lo podemos descomponer en su parte$-\lambda$ y su parte$-b$ escribiendo $[t, \ell(T)]$. Además, cuando necesitemos manipular o poner de manifiesto el primer elemento de $\ell(T)$, escribiremos $\ell(T)$ como $c(T)\cdot r(T)$, donde $c(T)$ es el primer elemento de $\ell(T)$ y $r(T)$ es el resto. De esta forma $T = [t, c(T) \cdot r(T)]$. Pasamos pues a definir formalmente nuestro cálculo.

2.2. Las variables

Definición 2.1 *Las variables serán elementos de un conjunto numerable, las representaremos por $x_1, x_2, \ldots$ o por $x, y, z, \ldots$*

2.3. Las reglas de construcción de términos

Definición 2.2 *Las reglas de construcción de términos vienen dadas por:*

a) Para cada $\gamma \in \mathbb{D}$ existe una regla que representaremos por I_γ tal que si x es una variable entonces $I_\gamma(x) = [x, \gamma]$ es un término.
Esta es una diferencia notable entre nuestro cálculo y el caso clásico, en él toda variable es un término, no ocurre así en nuestro caso, las variables, aquí no están etiquetadas, los términos sí.

b) Para cada $\gamma \in \mathbb{D}$ existe una regla que representaremos por F_γ tal que si x es una variable y $T = [t, \ell(T)]$ es un término, entonces $F_\gamma(x, T) = [\lambda x.t, \gamma \cdot \ell(T)]$ es un término.

c) Para cada $\gamma \in \mathbb{D}$ existe una regla que representaremos por G_γ tal que si $T = [t, \ell(T)]$ y $U = [u, \ell(U)]$, son términos, entonces $G_\gamma(T, U) = [(t\ u), \gamma \cdot \ell(T) \cdot \ell(U)]$ es un término.

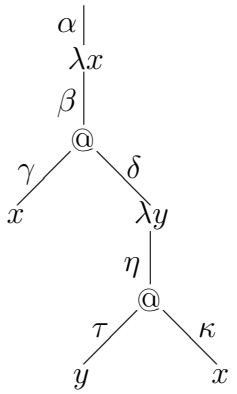
A continuación mostramos, por medio de un ejemplo, las distintas notaciones que se pueden utilizar para representar un término de nuestro cálculo, para el caso clásico ver [40].

Ejemplo.- Sea el término $A = (\lambda x.(x\ (\lambda y.(y\ x))))$ del lambda cálculo clásico y suponemos que los hemos etiquetado con los elementos $\alpha, \beta, \gamma, \delta, \eta, \tau, \kappa \in \mathbb{D}$. Consultar página 19 para la notación en el caso clásico. Las distintas notaciones equivalentes serían:

1. La parte-b como lista

$$A = [(\lambda x.(x\ (\lambda y.(y\ x)))), \alpha \cdot \beta \cdot \gamma \cdot \delta \cdot \eta \cdot \tau \cdot \kappa]$$

2. Como grafo decorado



3. La parte-b conservando su estructura

$$\mathrm{A} = [\lambda x.x\ (\lambda y.y\ x), \lambda\alpha@\beta\gamma\lambda\delta@\eta\tau\kappa]$$

   esta notación se utilizará extensamente en el siguiente capítulo

4. Como árbol
   El árbol subyacente es $\{\epsilon, 0, 01, 02, 020, 0201, 0202\}$. El árbol etiquetado está definido por la aplicación φ tal que:

$$\begin{array}{lcl} \varphi(\epsilon) & = & [\lambda x, \alpha] \\ \varphi(0) & = & [@, \beta] \\ \varphi(01) & = & [x, \gamma] \\ \varphi(02) & = & [\lambda y, \delta] \\ \varphi(020) & = & [@, \eta] \\ \varphi(0201) & = & [y, \tau] \\ \varphi(0202) & = & [x, \kappa] \end{array}$$

Convenio de Barendregt.- *Si $M_1, ..., M_n$ se presentan en una definición, demostración etc, convenimos que todas las variables ligadas de estos términos se eligen de forma que sean diferentes de las variables libres.*

2.4. La sustitución

Definición 2.3 *Sean U y T términos y x una variable, se define $U\langle T/x\rangle$ como el resultado de sustituir T en las ocurrencias libres de x en U*

1. *Si la parte-λ de U, u, es una variable*

   a) *Si u es igual a x*

   $$[x,\alpha]\langle[t,c(T)\cdot r(T)]/x\rangle = [t,s(\alpha,c(T))\cdot r(T)] \tag{2.1}$$

   Aparece en nuestro cálculo la necesidad de introducir sobre el conjunto $\mathbb{D}$ algún mecanismo que refleje, en la lista del término resultante, que ha tenido lugar una sustitución. Proponemos que dicho mecanismo venga dado por s, una función de $\mathbb{D}\times\mathbb{D}$ en $\mathbb{D}$, que operará sobre α y $c(T)$. Precisamente, la caracterización de esta función s, y de otras funciones que tendremos que introducir más adelante, constituyen el objeto de este capítulo y el siguiente.

   b) *Si u es distinto de x*

   $$U\langle T/x\rangle = U$$

2. *Si U es un término de la forma $[(w\ v),\gamma\cdot\ell(W)\cdot\ell(V)]$, es decir la última operación de construcción de U fue una operación G_γ, entonces*

   $$U\langle T/x\rangle \equiv [(w\langle t/x\rangle\ \ v\langle t/x\rangle),\gamma\cdot\alpha\cdot\beta]$$

   donde $\alpha=\ell(W\langle T/x\rangle)$ y $\beta=\ell(V\langle T/x\rangle)$

3. *Si U es un término de la forma $[\lambda y.w,\gamma\cdot\ell(W)]$, es decir la última operación de construcción de U fue una operación F_γ, entonces*

   $$U\langle T/x\rangle = F_\gamma(y,W\langle T/x\rangle) = [\lambda y.w\langle t/x\rangle,\gamma\cdot\ell(W\langle T/x\rangle)]$$

   No es necesario distinguir casos, suponiendo que $x\neq y$ y que y no aparece libre en T, pues utilizamos el convenio de Barendregt sobre variables.

2.4.1. Propiedades de la sustitución

1. *Si x no aparece libre en M, entonces $M\langle N/x\rangle = M$.*

   Demostración.- La demostración, por inducción sobre la estructura de M, es similar al caso clásico, y a partir ella no se obtiene caracterización alguna para s, por lo que la omitimos. □

2. *Si $x \neq y$ y x no es libre en P, entonces*

$$M\langle N/x\rangle\langle P/y\rangle = M\langle P/y\rangle\langle N\langle P/y\rangle/x\rangle \tag{2.2}$$

Demostración.- Por inducción sobre la estructura de M

a) Parte-λ de M es una variable. Es decir $M = [m, \gamma]$

1) Sea $m = x$.
Llamemos N' a $M\langle N/x\rangle = [n, s(\gamma, c(N)) \cdot r(N)]$ entonces

$$N'\langle P/y\rangle = [n\langle p/y\rangle, \ell(N'\langle P/y\rangle)] \tag{2.3}$$

$M\langle P/y\rangle = M$ y $N\langle P/y\rangle = [n\langle p/y\rangle, \ell(N\langle P/y\rangle)]$ por lo que

$$\begin{array}{l} M\langle N\langle P/y\rangle/x\rangle = \\ \quad [n\langle p/y\rangle, s(\gamma, c(N\langle P/y\rangle)) \cdot r(N\langle P/y\rangle)] \end{array} \tag{2.4}$$

En 2.3 y 2.4 sus partes-λ coinciden, así como $r(N'\langle P/y\rangle)$ y $r(N\langle P/y\rangle)$, por tanto la igualdad entre ambos términos sólo depende del primer elemento de su parte-b, es decir de

$$c(N'\langle P/y\rangle) \tag{2.5}$$

y

$$s(\gamma, c(N\langle P/y\rangle)) \tag{2.6}$$

para estudiar con detalle 2.5 y 2.6 veamos los distintos casos que puede presentar N

a' Si $N = [y, \alpha]$ entonces 2.5 sería

$$s(s(\gamma, \alpha), c(P)) \tag{2.7}$$

y 2.6 sería

$$s(\gamma, s(\alpha, c(P))) \tag{2.8}$$

b' Si $N = [z, \alpha]$ con $z \neq y$ tanto 2.5 como 2.6 coinciden con $s(\gamma, \alpha)$

c' Si la parte-λ de N es distinta de variable tanto 2.5 como 2.6 coinciden con $s(\gamma, c(N))$

2) Sea $m = y$ entonces los dos lados de 2.2 son iguales a

$$[P, s(\gamma, c(P)) \cdot r(P)]$$

3) Sea $m \neq x, y$ entonces los dos lados de 2.2 son iguales a M

b) Sea $M = [\lambda z.M_1, \sigma \cdot \ell\,(M_1)]$ por el convenio sobre las variables podemos suponer que $z \neq x, y$ y que z no es libre en N ni en P. Entonces por la hipótesis de inducción es cierto que

$$M_1\,\langle N/x\rangle\,\langle P/y\rangle = M_1\,\langle P/y\rangle\,\langle N\,\langle P/y\rangle\,/x\rangle$$

y por tanto

$$M\,\langle N/x\rangle\,\langle P/y\rangle = M\,\langle P/y\rangle\,\langle N\,\langle P/y\rangle\,/x\rangle$$

c) Sea $M = [(m_1\ m_2)\,, \gamma \cdot \ell\,(M_1) \cdot \ell\,(M_2)]$. La igualdad de 2.2 se sigue nuevamente de la hipótesis de inducción. □

Por tanto, para que se verifique 2.2, hemos de admitir que la función s ha de ser tal que haga coincidir 2.7 y 2.8 para cualesquiera valores $a, b, c \in \mathbb{D}$, es decir que

$$s\,(a, s\,(b, c)) = s\,(s\,(a, b)\,, c) \tag{2.9}$$

o lo que es lo mismo s ha de ser asociativa. Con ello hemos obtenido el primer resultado importante. A continuación introduciremos en nuestro cálculo las dos reglas de equivalencia α y β

2.5. La $\alpha-$equivalencia

Vamos a definir una relación de equivalencia sobre L, conjunto de términos definido en el proceso anterior, a la que llamaremos α-equivalencia, y la representaremos por $\equiv_\alpha$. Intuitivamente si U y U' son términos cuyas partes b coinciden, $U \equiv_\alpha U'$ significa que U' se ha obtenido a partir de U renombrando las variables ligadas de U; es decir se tiene que $U \equiv_\alpha U'$ si y solamente si U y U' tienen la misma sucesión de símbolos (confundiendo las variables), las mismas ocurrencias libres de las mismas variables y si cada λ liga las mismas ocurrencias de variables en U y en U'.

Definición 2.4 *Dados dos elementos de L decimos que $U \equiv_\alpha U'$*

- *Si la parte$-\lambda$ de U y U' son variables y $\ell(U) = \ell(U')$*
- *Si $U = [(w\ v), \gamma \cdot \ell(W) \cdot \ell(V)]$, entonces $U \equiv_\alpha U'$ sii*

$$U' = [(w'\ v'), \gamma \cdot \ell(W') \cdot \ell(V')]$$

con $W \equiv_\alpha W'$ y $V \equiv_\alpha V'$

- *Si* $U = [\lambda x.v, \gamma \cdot \ell(V)]$ *entonces* $U \equiv_\alpha U'$ *sii*

$$U' = [\lambda y.v \langle y/x \rangle, \gamma \cdot \ell(V)]$$

siendo la variable y *cualquier variable no libre de* v

Teorema 2.1 .- *La relación* $\equiv_\alpha$ *es una relación de equivalencia compatible en* L. *Llamaremos* Λ *a* $L/\equiv_\alpha$

Demostración.- La demostración de este teorema es similar a la demostración del teorema correspondiente en el caso clásico ver [7], y no suministra ninguna condición sobre la estructura de $\mathbb{D}$, por ello la omitimos. □

2.6. La β-reducción

Siguiendo [7] partiremos de una relación, a la que llamaremos ρ, esta tendrá la propiedad del diamante y la β-reducción, definida como el cierre transitivo de ρ, también tendrá dicha propiedad.

Nuestra preocupación se dirigirá a la parte-b, pues deseamos que ella también tenga la propiedad del diamante. Para ello vamos a introducir unas funciones, la f y la g, que aparecerán cada vez que hagamos una β-reducción. Para caracterizar a estas funciones seguiremos, paso a paso, la demostración de que la relación ρ verifica la propiedad del diamante, tal y como se hace en la referencia dada anteriormente, cada una de cuyas etapas nos suministrará condiciones sobre f, g y s para que la parte-b de nuestro cálculo cumpla también la propiedad deseada.

2.6.1. La relación ρ

Definición 2.5 *Dados dos elementos* M *y* N *de* Λ *decimos que* M *y* N *están relacionados, y lo representaremos por* $M\ \rho\ N$, *si* N *se obtiene aplicando un número finito de veces las siguientes reglas:*

1. *Para todo* $T \in \Lambda \quad T\ \rho\ T$
2. *Si* $T\ \rho\ T'$ *entonces* $[\lambda x.t, \alpha \cdot \ell(T)]\ \rho\ [\lambda x.t', \alpha \cdot \ell(T')]$
3. *Si* $T\ \rho\ T'$ *y* $U\ \rho\ U'$ *entonces*

$$[(u\ t), \beta \cdot \ell(U) \cdot \ell(T)]\ \rho\ [(u'\ t'), \beta \cdot \ell(U') \cdot \ell(T')]$$

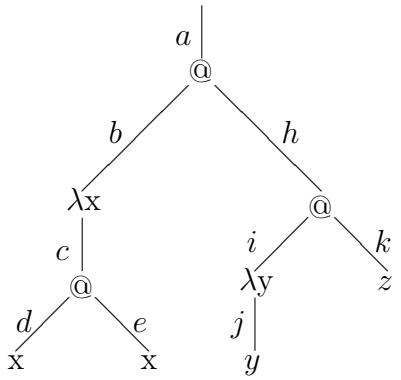
4. *Si* $T\ \rho\ T'$ *y* $U\ \rho\ U'$ *entonces*

$$[(\lambda x.u\ t), \alpha \cdot \beta \cdot c(U) \cdot r(U) \cdot c(T) \cdot r(T)]\ \rho\ [u' \langle t'/x \rangle, f(\alpha, \beta, c(U' \langle Z/x \rangle)) \cdot r(U' \langle Z/x \rangle)] \tag{2.10}$$

donde $Z = [t', g(\beta, c(T')) \cdot r(T')]$ *y siendo* f *una función de* $\mathbb{D} \times \mathbb{D} \times \mathbb{D}$ *en* $\mathbb{D}$ *y* g *una función de* $\mathbb{D} \times \mathbb{D}$ *en* $\mathbb{D}$. *Es decir* Z *y* T' *coinciden en su parte-*λ, *la única diferencia que existe entre ambas es que la* $c(Z) = g(\beta, c(T'))$, *siendo* $r(Z) = r(T')$.

Ejemplo.- Sea el término

$$M = [(\lambda x.(x\ x)(\lambda y.y\ z)), a \cdot b \cdot c \cdot d \cdot e \cdot h \cdot i \cdot j \cdot k]$$



El árbol subyacente es $\{\epsilon, 1, 2, 10, 101, 102, 21, 210, 22\}$.
Las etiquetas sobre dicho árbol serían:

$$\begin{aligned}
\varphi(\epsilon) &= [@, a] \\
\varphi(1) &= [\lambda x, b] \\
\varphi(2) &= [@, h] \\
\varphi(10) &= [@, c] \\
\varphi(101) &= [x, d] \\
\varphi(102) &= [x, e] \\
\varphi(21) &= [\lambda y, i] \\
\varphi(22) &= [z, k] \\
\varphi(210) &= [y, j]
\end{aligned}$$

Existen dos redex $M/\epsilon \equiv M$ y $M/2 \equiv [(\lambda y.y\ z), h \cdot i \cdot j \cdot k]$, y por tanto dos posibles vías de reducción (notar que $\epsilon < 2$):

a) Reducir primero $M/2$
b) Reducir primero M/ϵ

a) La reducción de $M/2$ da lugar al término N con

$$N \equiv [z, f(h, i, s(j, g(i, k)))]$$

Por lo que $M \to M'$ siendo

$$M' = [(\lambda x.(x\ x)\ z), a \cdot b \cdot c \cdot d \cdot e \cdot f(h, i, s(j, g(i, k)))]$$

en M' sólo existe un redex, que al reducirlo da lugar al término

$$T = [(z\ z), f(a, b, c) \cdot s(d, g(b, f(h, i, s(j, g(i, k))))) \cdot \\ s(e, g(b, f(h, i, s(j, g(i, k)))))]$$

b) La reducción de M/ϵ da lugar al término M'' con

$$M'' = [((\lambda y.y\ z)\ (\lambda y.y\ z)), f(a, b, c) \cdot s(d, g(b, h)) \cdot \\ i \cdot j \cdot k \cdot s(e, g(b, h)) \cdot i \cdot j \cdot k]$$

en el cual existen dos redex el $M''/1$ y el $M''/2$, ahora 1 y 2 no son comparables en el orden del prefijo. Igualmente vamos a seguir ambos caminos.

b1) Reducimos primero $M''/2$ dando lugar a

$$N_2'' \equiv [z, f(s(e, g(b, h)), i, s(j, g(i, k)))]$$

por lo que $M'' \to M_2''$ siendo

$$M_2'' \equiv [((\lambda y.y\ z)\ z), f(a, b, c) \cdot s(d, g(b, h)) \cdot \\ i \cdot j \cdot k \cdot f(s(e, g(b, h)), i, s(j, g(i, k)))]$$

reduciendo el único redex que nos queda

$$T_2 \equiv [(z\ z), f(a, b, c) \cdot f(s(d, g(b, h)), i, s(j, g(i, k))) \cdot \\ f(s(e, g(b, h)), i, s(j, g(i, k)))]$$

b2) Reducimos primero $M''/1$ dando lugar a

$$N_1'' = [z, f(s(d, g(b, h)), i, s(j, g(i, k)))]$$

por lo que $M'' \to M_1''$ siendo

$$M_1'' = [(z\ (\lambda y.y\ z)), f(a, b, c) \cdot f(s(d, g(b, h)), i, s(j, g(i, k))) \cdot \\ s(e, g(b, h)) \cdot i \cdot j \cdot k]$$

y reduciendo este último redex resultaría

$$T_1 = [(z\ z), f(a, b, c) \cdot f(s(d, g(b, h)), i, s(j, g(i, k))) \cdot \\ f(s(e, g(b, h)), i, s(j, g(i, k)))]$$

Como era de esperar, a consecuencia de las definiciones dadas para la β-reducción y la sustitución y de que los redex en M" no estaban relacionados, los resultados para T_1 y T_2 coinciden, independientemente de la elección que hagamos respecto de s, f y g. Si queremos que el cálculo$-b$ verifique el teorema de Church-Rosser, hemos de imponer condiciones a s, f y g de forma que, aunque dos redex están relacionados, el resultado no dependa del orden en el que se hace la reducción. En nuestro ejemplo hemos de lograr que T coincida con T_1, es decir que

$$s(d, g(b, f(h, i, s(j, g(i, k))))) \cdot s(e, g(b, f(h, i, s(j, g(i, k)))))$$

coincida con

$$f(s(d, g(b, h)), i, s(j, g(i, k))) \cdot f(s(e, g(b, h)), i, s(j, g(i, k)))$$

No nos queda más que imponer las condiciones a las funciones s, g y f para que la parte-b de nuestro cálculo cumpla las condiciones del teorema de Church-Rosser.

2.6.2. La transformación Φ_α^β

Dada la relación 2.10 representaremos la parte-b del segundo miembro de la relación por $\Phi_\alpha^\beta(U' \langle T'/x \rangle)$. El símbolo Φ_α^β con dos parámetros, α y β, realiza dos transformaciones:

1. Transformar el término por el cual vamos a sustituir la variable ligada
2. Transformar el término resultante de realizar la sustitución anterior.

La intención que nos lleva a introducir esta definición es la siguiente: cuando tenemos un término de la forma $[(\lambda x.u\ t), \alpha \cdot \beta \cdot \ell(U) \cdot \ell(T)]$, donde α corresponde a la aplicación y β corresponde a la abstracción al llevar a cabo la ρ-relación ambos, desde el punto de vista del λ-cálculo clásico, van a desaparecer, nos quedamos con el término $u \langle t/x \rangle$. Por tanto, en la parte-b debemos quedarnos con constancia de haber realizado una ρ-relación en la han intervenido α y β. Por ello, proponemos, a fin de mantener dicha información, que la ρ-relación se realice de manera que afecte:

- A todas la sustituciones (puede que no haya ninguna sustitución o puede que haya múltiples) que hagamos de x por T en U, para ello, si ϕ es el primer elemento-b de T lo sustituimos por el resultado de aplicar una cierta función g a β y ϕ.
- Globalmente a U y que sea independiente de que x aparezca en U o no, para ello, tomamos el primer elemento de la lista de U después de haber efectuado las sustituciones del punto anterior y lo modificamos, por medio de una cierta función f que dependerá de α, β y de dicho primer elemento.

El resultado de aplicar Φ_α^β a dos λ-términos U y T dependerá de la estructura de ambos. Para estudiar las distintas formas que adopta $\Phi_\alpha^\beta(U\,\langle T/x\rangle)$ suponemos que $U = [u, c(U)\cdot r(U)]$ y $T = [t, c(T)\cdot r(T)]$ entonces:

1. Si u es una variable, es decir $U = [u, \zeta]$, puede ocurrir que:

   a) $u \equiv x$ en cuyo caso

   1) Si t es una variable cualquiera, por ejemplo y, $c(T)$ será un elemento de $\mathbb{D}$ y $r(T) = \emptyset$ y por tanto $T = [y, \kappa]$ luego $T' = [y, g(\beta, \kappa)]$ por lo que:

   $$\Phi_\alpha^\beta(U < T/x >) = f(\alpha, \beta, s(\zeta, g(\beta, \kappa))$$

   2) Si t no es una variable $T' = [t, g(\sigma, c(T),)\cdot r(T)]$ por lo que

   $$\Phi_\alpha^\beta(U < T/x >) = f(\alpha, \beta, s(\zeta, g(\beta, c(T)))) \cdot r(T)$$

   b) $u \not\equiv x$, entonces independientemente de T tendríamos que

   $$\Phi_\alpha^\beta(U < T/x >) = f(\alpha, \beta, \zeta)$$

2. Si u no es una variable, entonces

   a) Si t es una variable, como en apartado 1.i anterior, $T' = [y, g(\beta, \kappa)]$ por lo que:

   $$\Phi_\alpha^\beta(U < T/x >) = f(\alpha, \beta, s(c(U), g(\beta, \kappa)) \cdot r(U < y/x >)$$

   b) Si t no es una variable $T' = [t, g(\sigma, c(T),)\cdot r(T)]$ por lo que

   $$\Phi_\alpha^\beta(U < T/x >) = f(\alpha, \beta, s(c(U), g(\beta, c(T)))) \cdot r(U < T'/x >)$$

2.6.3. Propiedades de la relación ρ

i) *Si $X = [x, \alpha]$ y $X\ \rho\ T$, entonces $T = X$*

Demostración.- En efecto si la parte-λ de X es una variable y se verifica que $X\ \rho\ T$, esto únicamente se puede alcanzar por medio de la regla 1 de la definición de ρ. □

ii) *Si $[\lambda x.U, \alpha\cdot\ell(U)]\ \rho\ T$, entonces T es de la forma $[\lambda x.U', \alpha\cdot\ell(U')]$ y $U\ \rho\ U'$*

Demostración.- Consideremos la última regla aplicada para obtener

$$[\lambda x.U, \alpha\cdot\ell(U)]\ \rho\ T$$

dicha regla sólo ha podido ser la regla 1 o la 2, de ahí el resultado. □

iii) *Si* $[(v\ u), \alpha \cdot \ell(V) \cdot \ell(U)]\ \rho\ T'$, *donde* $T' = [(v'\ u'), \alpha \cdot \ell(V') \cdot \ell(U')]$ *entonces* $U\ \rho\ U'$ *y* $V\ \rho\ V'$

Demostración.- Ahora hemos aplicando la regla 3 de la definición de ρ, de ahí el resultado. □

iv) *Si* $[(v\ u), \alpha \cdot \ell(V) \cdot \ell(U)]\ \rho\ T'$ *y* v *empieza por* λ, *para ser más precisos,* $v = \lambda x.w$, $\ell(V) = \beta \cdot \ell(W)$ *y* $U = [u, c(U) \cdot r(U)]$ *entonces*

$$T' = \left[w \left\langle u/x \right\rangle, \Phi_\alpha^\beta(W \left\langle U/x \right\rangle)\right]$$

Demostración.- Ahora estamos aplicando la regla 4 de la definición de ρ. □

v) *Si* $T\ \rho\ T'$ *y* $U\ \rho\ U'$ *entonces* $U \left\langle T/x \right\rangle\ \rho\ U' \left\langle T'/x \right\rangle$

Demostración.- Esta demostración no es tan directa como las anteriores, es bastante larga, pues hay que hacerla por inducción sobre la relación $U\ \rho\ U'$, es decir hay considerar las cuatro reglas dadas en la definición de ρ, y dentro de la consideración de cada regla hemos de considerar a su vez por medio de que regla se relacionan T y T'. Como contrapartida obtendremos los primeros resultados interesantes.

1. Consideramos que $U\ \rho\ U'$ es de la forma $U\ \rho\ U$. Tenemos que demostrar que

$$U \left\langle T/x \right\rangle\ \rho\ U \left\langle T'/x \right\rangle$$

1.1. Si $U = [x, \phi]$ entonces dicha relación se transformaría en

$$[t, s(\phi, c(T)) \cdot r(T)]\ \rho\ [t', s(\phi, c(T')) \cdot r(T')]$$

En el lambda cálculo clásico este caso es trivial, pero en nuestro cálculo, a fin de establecer las relaciones que ligan a s, f y g, es interesante cuando la relación entre T y T' se da como consecuencia de:

- $T = [(v\ u), \alpha \cdot \ell(V) \cdot \ell(U)]$ con v empezando por λ, es decir $T = G_\alpha(P, Q)$ donde:
  $P = [\lambda z.w, \beta \cdot \ell(W)]$ y
  $Q = [q, c(Q) \cdot r(Q)]$
- $T' = \left[w \left\langle q/z \right\rangle, \Phi_\alpha^\beta(W \left\langle Q/z \right\rangle)\right]$

Pues la relación

$$[x, \phi] \left\langle [(p\ q), \alpha \cdot \ell(P) \cdot \ell(Q)]/x \right\rangle\ \rho\ [x, \phi] \left\langle \left[w \left\langle q/z \right\rangle, \Phi_\alpha^\beta(W \left\langle Q/z \right\rangle)\right]/x \right\rangle$$

se puede escribir de dos formas

1.1.1 Primera forma. Realizando la sustitución en ambos miembros

$$[(p\ q), s(\phi,\alpha)\cdot\ell(P)\cdot\ell(Q)]\ \ \rho \\ \left[w\left\langle q/z\right\rangle, s(\phi, c(\Phi^{\beta}_{\alpha}(W\left\langle Q/z\right\rangle))\cdot r(\Phi^{\beta}_{\alpha}(W\left\langle Q/z\right\rangle))\right]$$

1.1.2 Segunda forma. Si tenemos en cuenta la forma de P, tras realizar la sustitución $U\left\langle T/x\right\rangle$ resulta

$$[(p\ q), s(\phi,\alpha)\cdot\ell(P)\cdot\ell(Q)]$$

que está ρ-relacionada con

$$\left[w\left\langle q/z\right\rangle, \Phi^{\beta}_{s(\phi,\alpha)}(W\left\langle Q/z\right\rangle))\right]$$

es decir que

$$[(p\ q), s(\phi,\alpha)\cdot\ell(P)\cdot\ell(Q)]\ \ \rho\ \ \left[w\left\langle q/z\right\rangle, \Phi^{\beta}_{s(\phi,\alpha)}(W\left\langle Q/z\right\rangle)\right]$$

si queremos que ambas formas coincidan es necesario que:

$$s(\phi, c(\Phi^{\beta}_{\alpha}(W\left\langle Q/z\right\rangle))\cdot r(\Phi^{\beta}_{\alpha}(W\left\langle Q/z\right\rangle))$$

coincida con

$$\Phi^{\beta}_{s(\phi,\alpha)}(W\left\langle Q/z\right\rangle)$$

y para ello basta imponer a s y f la siguiente condición:

$$s(\phi, f(\alpha,\beta,c)) = f(s(\phi,\alpha),\beta,c) \tag{2.11}$$

1.2 Si la parte-λ de U es una variable distinta de x entonces dicha relación se transformaría en $U\ \rho\ U$

1.3 Si U es de la forma $[(p\ q),\alpha\cdot\ell(P)\cdot\ell(Q)]$ entonces la relación $[(p\ q),\alpha\cdot\ell(P)\cdot\ell(Q)]\left\langle T/x\right\rangle\ \ \rho\ \ [(p\ q),\alpha\cdot\ell(P)\cdot\ell(Q)]\left\langle T'/x\right\rangle$ se transforma en

$$[(p\left\langle t/x\right\rangle\ \ q\left\langle t/x\right\rangle),\alpha\cdot\ell(P\left\langle T/x\right\rangle)\cdot\ell(Q\left\langle T/x\right\rangle)]\ \ \rho \\ [(p\left\langle t'/x\right\rangle\ \ q\left\langle t'/x\right\rangle),\alpha\cdot\ell(P\left\langle T'/x\right\rangle)\cdot\ell(Q\left\langle T'/x\right\rangle)]$$

y teniendo en cuenta 3) esta es cierta si

$$P\left\langle T/x\right\rangle\ \ \rho\ P\left\langle T'/x\right\rangle\quad\text{y}\quad Q\left\langle T/x\right\rangle\ \ \rho\ Q\left\langle T'/x\right\rangle$$

y ambas se verifican si aplicamos la hipótesis de inducción.

1.4 Si U es de la forma $[\lambda y.P,\alpha\cdot\ell(P)]$ la relación se transformaría en

$$[\lambda y.P,\alpha\cdot\ell(P)]\left\langle T/x\right\rangle\ \ \rho\ \ [\lambda y.P,\alpha\cdot\ell(P)]\left\langle T'/x\right\rangle$$

(como U sólo es un representante de la clase a la que pertenece podemos tomar, eligiendo, si es necesario, otro representante, ya que y está ligado, de forma que $x\neq y$) Ahora bien por ii) esta relación es cierta si $P\left\langle T/x\right\rangle\ \ \rho\ P\left\langle T'/x\right\rangle$ y esta se verifica aplicando la hipótesis de inducción

2 Si $U\ \rho\ U'$es de la forma $[\lambda y.p, \gamma \cdot \ell(P)]\ \rho\ [\lambda y.p', \gamma \cdot \ell(P')]$ entonces por 2) $P\ \rho\ P'$ y aplicando la hipótesis de inducción llegamos a que $P\langle T/x\rangle\ \rho\ P'\langle T'/x\rangle$. En cuyo caso se verificará:

$$[\lambda y.p\langle t/x\rangle, \gamma \cdot \ell(P\langle T/x\rangle)]\ \rho\ [\lambda y.(p'\langle t'/x\rangle), \gamma \cdot \ell(P'\langle T'/x\rangle)]$$

3 Si $U\ \rho\ U'$ es de la forma $[(p\ q), \gamma \cdot \ell(P) \cdot \ell(Q)]\ \rho\ [(p'\ q'), \gamma \cdot \ell(P') \cdot \ell(Q')]$ donde p no empieza por λ. Por la propiedad iii) tenemos que $P\ \rho\ P'$ y $Q\ \rho\ Q'$. Podemos pues aplicar la hipótesis de inducción a ambas ρ-relaciones y tendríamos que $P\langle T/x\rangle\ \rho\ P'\langle T'/x\rangle$ y $Q\langle T/x\rangle\ \rho\ Q'\langle T'/x\rangle$ y por 3) de la definición tendríamos que

$$\begin{array}{l}[(p\langle t/x\rangle\ q\langle t/x\rangle), \gamma \cdot \ell(P\langle T/x\rangle) \cdot \ell(Q\langle T/x\rangle)]\ \rho \\ \qquad [(p'\langle t'/x\rangle\ q'\langle t'/x\rangle), \gamma \cdot \ell(P'\langle T'/x\rangle) \cdot \ell(Q'\langle T'/x\rangle)]\end{array}$$

4 Si $U\ \rho\ U'$ es de la forma

$$[(\lambda y.p\ q), \gamma \cdot \sigma \cdot \ell(P) \cdot \ell(Q)]\ \rho\ \left[p'\langle q'/y\rangle, \Phi^\sigma_\gamma(P'\langle Q'/y\rangle)\right]$$

es por que deben existir las relaciones $P\ \rho\ P'$ y $Q\ \rho\ Q'$. Por la hipótesis de inducción, se tiene pues que $P\langle T/x\rangle\ \rho\ P'\langle T'/x\rangle$ y $Q\langle T/x\rangle\ \rho\ Q'\langle T'/x\rangle$. Por tanto

$$U\langle T/x\rangle = [(\lambda y.p\langle t/x\rangle\ q\langle t/x\rangle), \gamma \cdot \sigma \cdot \ell(P\langle T/x\rangle) \cdot \ell(Q\langle T/x\rangle)]$$

como consecuencia de la definición de sustitución. Aplicando el 4) de la definición de ρ se deduce que

$$\begin{array}{l}[(\lambda y.p\langle t/x\rangle\, q\langle t/x\rangle), \gamma \cdot \sigma \cdot \ell(P\langle T/x\rangle) \cdot \ell(Q\langle T/x\rangle)]\ \rho \\ \quad \left[p'\langle t'/x\rangle\langle q'\langle t'/x\rangle/y\rangle, \Phi^\sigma_\gamma((P'\langle T'/x\rangle)\langle Q'\langle T'/x\rangle/y\rangle)\right]\end{array}$$

si aplicamos al segundo miembro de esta relación las propiedades de la sustitución y suponiendo que y no aparece en t', este se podría escribir como

$$\left[p'\langle q'/y\rangle\langle t'/x\rangle, \Phi^\sigma_\gamma(P'\langle Q'/y\rangle)\langle T'/x\rangle\right] = U'\langle T'/x\rangle$$

Hemos de imponer a f, g y s las condiciones necesarias para que

$$\Phi^\sigma_\gamma((P'\langle T'/x\rangle)\langle Q'\langle T'/x\rangle/y\rangle) = \Phi^\sigma_\gamma(P'\langle Q'/y\rangle)\langle T'/x\rangle$$

se verifique para cualesquiera γ, σ, P', Q', y T'. Por un lado tenemos

$$\begin{array}{l}\Phi^\sigma_\gamma((P'\langle T'/x\rangle)\langle Q'\langle T'/x\rangle/y\rangle) = \\ \quad f(\gamma, \sigma, c(P'\langle T'/x\rangle\langle B/y\rangle)) \cdot r(P'\langle T'/x\rangle\langle Q'\langle T'/x\rangle/y\rangle)\end{array} \tag{2.12}$$

donde $B = [q'\langle t'/x\rangle, g(\sigma, c(Q'\langle T'/x\rangle) \cdot r(Q'\langle T'/x\rangle)]$. Por otro lado

$$\Phi^\sigma_\gamma(P'\langle Q'/y\rangle) = f(\gamma, \sigma, c(P'\langle A/y\rangle)) \cdot r(P'\langle Q'/y\rangle)$$

donde $A = [q', g(\sigma, c(Q')) \cdot r(Q')]$ con lo que $\Phi^\sigma_\gamma(P'\langle Q'/y\rangle)\langle T'/x\rangle$ puede tomar dos valores, según que la parte-λ de $P'\langle A/y\rangle$ sea igual a x o distinta de x.

- En el primer caso $P' \langle A/y \rangle = [x, f(\gamma, \sigma, c(P' \langle A/y \rangle)]$ y por tanto

$$\begin{array}{l} \Phi_\gamma^\sigma(P' \langle Q'/y \rangle) \langle T'/x \rangle = \\ \quad s(f(\gamma, \sigma, c(P' \langle A/y \rangle)), c(T')) \cdot r(\langle T'/x \rangle) \end{array} \tag{2.13}$$

- Y en caso contrario

$$\begin{array}{l} \Phi_\gamma^\sigma(P' \langle Q'/y \rangle) \langle T'/x \rangle = \\ \quad f(\gamma, \sigma, c(P' \langle A/y \rangle)) \cdot r(P' \langle Q'/y \rangle \langle T'/x \rangle \end{array} \tag{2.14}$$

La igualdad entre los segundos miembros de 2.12 y 2.13 o entre los segundos miembros de 2.12 y 2.14 sólo depende del primer elemento ya que

$$r(P' \langle T'/x \rangle \langle Q' \langle T'/x \rangle /y \rangle) \text{ y } r(P' \langle Q'/y \rangle \langle T'/x \rangle)$$

coinciden como consecuencia de las propiedades de la sustitución. Ahora bien que la parte-λ de $P' \langle Q'/y \rangle = x$ sólo puede provenir de dos casos:

4.1) Que $P' = [x, n]$ y Q' sea cualquiera

4.2) Que $P' = [y, n]$ y $Q' = [x, m]$

4.1) En este caso el primer elemento de la segunda parte de la igualdad 2.12, es decir, $f(\gamma, \sigma, c(P' \langle T'/x \rangle \langle B/y \rangle))$ se escribiría como

$$f(\gamma, \sigma, s(n, c(T'))) \tag{2.15}$$

Y ya que y no aparece libre en T', el primer elemento de la segunda parte de la igualdad 2.13, es decir, $s(f(\gamma, \sigma, c(P' \langle A/y \rangle)), c(T'))$ se escribiría como

$$s(f(\gamma, \sigma, n), c(T')) \tag{2.16}$$

Por tanto, para que 2.15 y 2.16 coincidan, hemos de admitir:

$$s(f(\gamma, \sigma, n), c) = f(\gamma, \sigma, s(n, c)) \tag{2.17}$$

4.2) Ahora teniendo en cuenta que

$$B = [q' \langle t'/x \rangle, g(\sigma, c(Q' \langle T'/x \rangle) \cdot r(Q' \langle T'/x \rangle)]$$

y al ser $Q' = [x, m]$ se puede escribir

$$B = [t', g(\sigma, s(m, c(T'))) \cdot r(T')]$$

lo que da lugar a que el primer elemento de la segunda parte de 2.12 se pueda escribir como

$$f(\gamma, \sigma, s(n, g(\sigma, s(m, c(T'))))) \tag{2.18}$$

Por otro lado, ahora $A = [x, g(\sigma, m)]$ con lo que el primer miembro de la segunda parte de 2.13 se puede escribir como

$$s(f(\gamma, \sigma, s(n, g(\sigma, m))), c(T'))$$

que utilizando 2.17, se puede escribir como

$$f(\gamma, \sigma, s(s(n, g(\sigma, m)), c(T')))$$

y utilizando 2.9 se podría poner como

$$f(\gamma, \sigma, s(n, s(g(\sigma, m), c(T')))) \tag{2.19}$$

Para que 2.18 y 2.19 coincidan es suficiente admitir que

$$g(\sigma, s(m, c)) = s(g(\sigma, m), c) \tag{2.20}$$

□

2.6.4. El teorema de Church-Rosser (versión borrosa)

Conforme a lo señalado en la página 51 establecemos la siguiente definición:

Definición 2.6 *La β-reducción es el cierre transitivo de la relación ρ*

Teniendo en cuenta las definiciones dadas en la página 18, para demostrar el teorema de Church-Rosser:

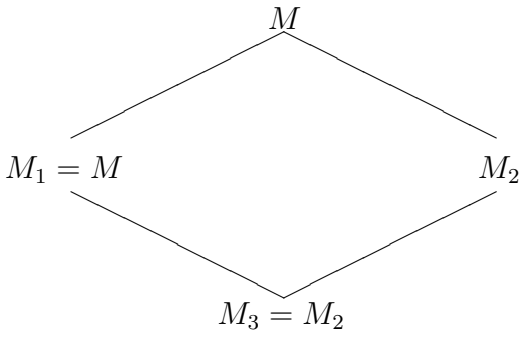
Teorema 2.2 *La β-reducción tiene la propiedad de Church-Rosser*

basta demostrar el siguiente teorema. Consultar [39]:

Teorema 2.3 *Si se verifica que $M\ \rho\ M_1$ y $M\ \rho\ M_2$ entonces existe un M_3 tal que $M_1\ \rho\ M_3$ y $M_2\ \rho\ M_3$.*

Demostración.- Este teorema, fundamental del lambda cálculo, está demostrado de distintas formas, por ejemplo en [16], [7], [39], etc. Nosotros vamos a tomar una de ellas, la dada en [7] capítulo 3, y la seguiremos paso a paso, para investigar las consecuencias que tiene sobre las funciones s, f y g definidas sobre $\mathbb{D}$. La demostración se hace por inducción sobre la definición de $M\ \rho\ M_1$. Necesitamos demostrar que para todo $M\ \rho\ M_2$ existe un M_3 tal que $M_1\ \rho\ M_3$ y $M_2\ \rho\ M_3$

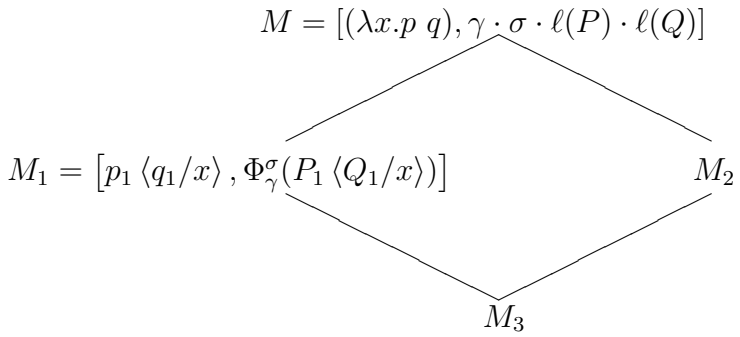
Caso 1.- Si $M\ \rho\ M_1$ porque $M = M_1$ entonces basta tomar $M_3 = M_2$.



Caso 2.- Consideramos que $M\ \rho\ M_1$ es de la forma

$$[(\lambda x.p\ q), \gamma \cdot \sigma \cdot \ell(P) \cdot \ell(Q)]\ \ \rho\ \ \left[p_1 \langle q_1/x \rangle, \Phi^{\sigma}_{\gamma}(P_1 \langle Q_1/x \rangle)\right]$$

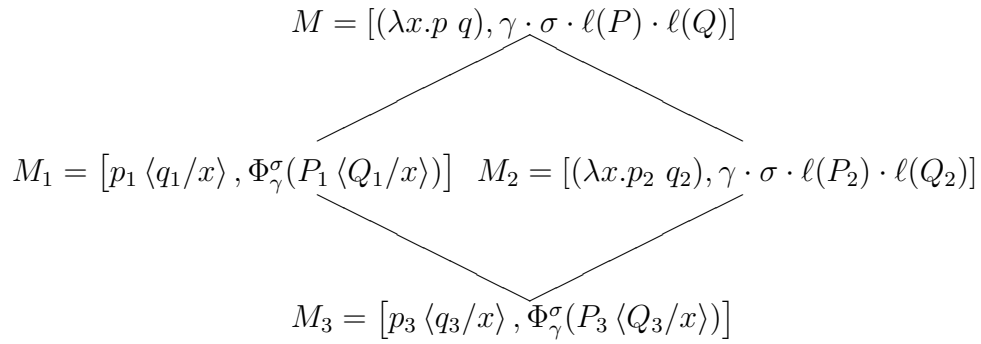
como consecuencia de que $P\ \rho\ P_1$ y $Q\ \rho\ Q_1$, podemos distinguir dos subcasos, aplicando la propiedad iii) o la iv) de ρ



2.1 (Aplicando iii)) $M\ \rho\ M_2$ es de la forma

$$[(\lambda x.p\ q), \gamma \cdot \sigma \cdot \ell(P) \cdot \ell(Q)]\ \ \rho\ \ [(\lambda x.p_2\ q_2), \gamma \cdot \sigma \cdot \ell(P_2) \cdot \ell(Q_2)]$$

donde $P\ \rho\ P_2$ y $Q\ \rho\ Q_2$.



Por la hipótesis de inducción existen términos P_3 y Q_3 tales que $P_1\ \rho\ P_3$, $P_2\ \rho\ P_3$, $Q_1\ \rho\ Q_3$, $Q_2\ \rho\ Q_3$. Por la propiedad v) de ρ podemos tomar $M_3 = \left[p_3\left\langle q_3/x\right\rangle, \Phi^{\sigma}_{\gamma}(P_3\left\langle Q_3/x\right\rangle)\right]$. Es decir realizamos una ρ-reducción para establecer la relación entre M y M_1, pero no al establecer la relación entre M y M_2. Para estudiar las consecuencias que podemos deducir acerca de las funciones s, f y g vamos a considerar dos casos de forma separada. En el primer caso consideramos que P y P_2 coinciden realizandose el paso de M a M_2 por una ρ-reducción de Q a Q_2. En el segundo caso quienes coinciden son Q y Q_2 y que dicho paso se ha realizado por una ρ-reducción entre P y P_2

2.1.1 Supongamos

1) la relación que existe entre Q y Q_2 es consecuencia de ser
   $Q = [(\lambda w.t\ s), \gamma_1 \cdot \sigma_1 \cdot \ell(T) \cdot \ell(S)]$ y
   $Q_2 = \left[t\left\langle s/w\right\rangle, \Phi^{\sigma_1}_{\gamma_1}(T\left\langle S/w\right\rangle)\right]$
2) $Q_1 = Q$
3) La relación entre Q_1 y Q_3 es consecuencia de haber realizado la misma reducción que la efectuada en el punto 1)

Tendriamos dos resultados para M_3. Los términos de partida son:

- $M = [(\lambda x.p\ (\lambda w.t\ s)), \gamma \cdot \sigma \cdot \ell(P) \cdot \gamma_1 \cdot \sigma_1 \cdot \ell(T) \cdot \ell(S)]$
- $M_1 = \left[p_1\left\langle q_1/x\right\rangle, \Phi^{\sigma}_{\gamma}(P_1\left\langle Q_1/x\right\rangle)\right]$ es decir
  $M_1 = [p_1\left\langle q_1/x\right\rangle, f(\gamma, \sigma, c(P_1\left\langle A/x\right\rangle)) \cdot r(P_1\left\langle A/x\right\rangle)]$
  donde $A = [(\lambda w.t\ s), g(\sigma, \gamma_1) \cdot \sigma_1 \cdot \ell(T) \cdot \ell(S)]$
- $M_2 = \left[(\lambda x.p_2\ q_2), \gamma \cdot \sigma \cdot \ell(P_2) \cdot \Phi^{\sigma_1}_{\gamma_1}(T\left\langle S/w\right\rangle)\right]$

$$M = [(\lambda x.p\ (\lambda w.t\ s)), \gamma \cdot \sigma \cdot \ell(P) \cdot \gamma_1 \cdot \sigma_1 \cdot \ell(T) \cdot \ell(S)]$$

$$M_1 = \left[p_1\left\langle q_1/x\right\rangle, \Phi^{\sigma}_{\gamma}(P_1\left\langle Q_1/x\right\rangle)\right] \quad M_2 = \left[(\lambda x.p_2\ q_2), \gamma \cdot \sigma \cdot \ell(P_2) \cdot \Phi^{\sigma_1}_{\gamma_1}(T\left\langle S/w\right\rangle)\right]$$

$$M_3 = \left[p_3\left\langle q_3/x\right\rangle, \Phi^{\sigma}_{\gamma}(P_3\left\langle Q_3/x\right\rangle)\right]$$

- **Primera forma de** M_3.- Como consecuencia de $M_2\ \rho\ M_3$ esta sería de la forma $\left[p_3\left\langle q_3/x\right\rangle, \Phi^{\sigma}_{\gamma}(P_3\left\langle Q_3/x\right\rangle)\right]$ donde

$$\Phi^{\sigma}_{\gamma}(P_3\left\langle Q_3/x\right\rangle) = f(\gamma, \sigma, c(P_3\left\langle B/x\right\rangle)) \cdot r(P_3\left\langle B/x\right\rangle) \tag{2.21}$$

con $B = \left[q_3, g(\sigma, c(\Phi^{\sigma_1}_{\gamma_1}(T\left\langle S/w\right\rangle))) \cdot r(Q_3)\right]$ es decir

$$B = [q_3, g(\sigma, f(\gamma_1, \sigma_1, c(T\left\langle S'/w\right\rangle))) \cdot r(T\left\langle S'/w\right\rangle)] \tag{2.22}$$

donde $S' = [s, g(\sigma_1, c(S)) \cdot r(S)]$. En 2.21 dependiendo de P_3 se puede dar los siguientes casos:

a) si la parte-λ de P_3 es distinta de variable se puede escribir como

$$\Phi_\gamma^\sigma(P_3 \langle Q_3/x \rangle) = f(\gamma, \sigma, c(P_3)) \cdot r(P_3 \langle B/x \rangle)) \tag{2.23}$$

b) si la parte-λ de P_3 es una variable distinta de x

$$\Phi_\gamma^\sigma(P_3 \langle Q_3/x \rangle) = f(\gamma, \sigma, c(P_3)) \tag{2.24}$$

c) si la parte-λ de P_3 es la variable x

$$\Phi_\gamma^\sigma(P_3 \langle Q_3/x \rangle) = f(\gamma, \sigma, s(c(P_3), c(B))) \cdot r(B)$$

Haciendo uso de 2.22 esta última igualdad se escribiría como

$$\begin{aligned} &\Phi_\gamma^\sigma(P_3 \langle Q_3/x \rangle) = \\ &\quad f(\gamma, \sigma, s(c(P_3), g(\sigma, f(\gamma_1, \sigma_1, c(T \langle S'/w \rangle)))) \\ &\qquad \cdot r(T \langle S'/w \rangle) \end{aligned} \tag{2.25}$$

A su vez, en esta, dependiendo de T, se puede dar los siguientes casos:

c1) si la parte-λ de T es distinta de variable se puede escribir como

$$\begin{aligned} &\Phi_\gamma^\sigma(P_3 \langle Q_3/x \rangle) = \\ &f(\gamma, \sigma, s(c(P_3), g(\sigma, f(\gamma_1, \sigma_1, c(T))) \cdot r(T \langle S'/w \rangle) \end{aligned} \tag{2.26}$$

c2) si la parte-λ de T es una variable distinta de w

$$\begin{aligned} &\Phi_\gamma^\sigma(P_3 \langle Q_3/x \rangle) = \\ &\quad f(\gamma, \sigma, s(c(P_3), g(\sigma, f(\gamma_1, \sigma_1, c(T))))) \end{aligned} \tag{2.27}$$

c3) si la parte-λ de T es la variable w

$$\begin{aligned} &\Phi_\gamma^\sigma(P_3 \langle Q_3/x \rangle) = \\ &f(\gamma, \sigma, s(c(P_3), g(\sigma, f(\gamma_1, \sigma_1, s(c(T), g(\sigma_1, c(S))))))) \\ &\qquad \cdot r(T \langle S'/w \rangle) \end{aligned} \tag{2.28}$$

- **Segunda forma de** M_3.- Como consecuencia de $M_1 \ \rho \ M_3$, M_3 sería de la forma

$$\left[p_3 \langle q_3/x \rangle , \Phi_\gamma^\sigma(P_3 \langle Q_3/x \rangle)\right]$$

donde

$$\Phi_\gamma^\sigma(P_3 \langle Q_3/x \rangle) = f(\gamma, \sigma, c(P_3 \langle C/x \rangle)) \cdot r(P_3 \langle C/x \rangle) \tag{2.29}$$

con $C = \left[t \langle s/w \rangle , \Phi_{g(\sigma,\gamma_1)}^{\sigma_1}(T \langle S/w \rangle)\right]$ es decir

$$C = t \langle s/w \rangle , f(g(\sigma, \gamma_1), \sigma_1, c(T \langle S'/w \rangle)) \cdot r(T \langle S'/w \rangle) \tag{2.30}$$

donde $S' = [s, g(\sigma_1, c(S)) \cdot r(S)]$. En 2.29 dependiendo de P_3 se puede dar los siguientes casos:

a) si la parte-λ de P_3 es distinta de variable se puede escribir como

$$\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = f(\gamma,\sigma,c(P_3))\cdot r(P_3\,\langle C/x\rangle) \tag{2.31}$$

b) si la parte-λ de P_3 es una variable distinta de x

$$\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = f(\gamma,\sigma,c(P_3)) \tag{2.32}$$

c) si la parte-λ de P_3 es la variable x

$$\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = f(\gamma,\sigma,s(c(P_3),c(C)))\cdot r(C) \tag{2.33}$$

Haciendo uso de 2.30 esta última se escribiría como

$$\begin{array}{l}\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = \\ f(\gamma,\sigma,s(c(P_3),f(g(\sigma,\gamma_1),\sigma_1,c(T\,\langle S'/w\rangle)))) \\ \qquad\cdot r(T\,\langle S'/w\rangle)\end{array} \tag{2.34}$$

En donde dependiendo de T se puede dar los siguientes casos:

- c1) si la parte-λ de T es distinta de variable se puede escribir como

$$\begin{array}{l}\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = \\ f(\gamma,\sigma,s(c(P_3),f(g(\sigma,\gamma_1),\sigma_1,c(T)))) \\ \qquad\cdot r(T\,\langle S'/w\rangle)\end{array} \tag{2.35}$$

- c2) si la parte-λ de T es una variable distinta de w

$$\begin{array}{l}\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = \\ f(\gamma,\sigma,s(c(P_3),f(g(\sigma,\gamma_1),\sigma_1,c(T))))\end{array} \tag{2.36}$$

- c3) si la parte-λ de T es la variable w

$$\begin{array}{l}\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = \\ f(\gamma,\sigma,s(c(P_3),f(g(\sigma,\gamma_1),\sigma_1,s(c(T),g(\sigma_1,c(S)))))) \\ \qquad\cdot r(T\,\langle S'/w\rangle)\end{array} \tag{2.37}$$

Y ahora hemos de imponer condiciones a s, f y g para que coincidan:

- 2.23 y 2.31
- 2.26 y 2.35
- 2.27 y 2.36
- 2.28 y 2.37

pues la coincidencia entre 2.24 y 2.32 se da independientemente de s, f y g. Para ello es suficiente que

$$f(g(b,c),d,e) = g(b,f(c,d,e)) \tag{2.38}$$

para cualesquiera valores b, c, d, e

2.1.2 Ahora suponemos que:

- 1) la relación que existe entre P y P_2 es consecuencia de:
  $P = [(\lambda w.t\ s), \gamma_1 \cdot \sigma_1 \cdot \ell(T) \cdot \ell(S)]$ y
  $P_2 = \left[t\left\langle s/w\right\rangle, \Phi^{\sigma}_{\gamma}(T\left\langle S/w\right\rangle)\right]$
- 2) $P_1 = P$
- 3) La relación entre P_1 y P_3 es consecuencia de haber realizado la misma reducción que la efectuada en el punto 1)

Tendríamos dos resultados para M_3. Los términos de partida son:

- $M = [(\lambda x.(\lambda w.t\ s)\ q), \gamma \cdot \sigma \cdot \gamma_1 \cdot \sigma_1 \cdot \ell(T) \cdot \ell(S) \cdot \ell(Q)]$
- $M_1 = \left[(\lambda w.t\ s)\left\langle q/x\right\rangle, \Phi^{\sigma}_{\gamma}(E\left\langle H/x\right\rangle)\right]$
  donde $E = [(\lambda w.t\ s), \gamma_1 \cdot \sigma_1 \cdot \ell(T) \cdot \ell(S)]$ y
  $H = [q, g(\sigma, c(Q)) \cdot r(Q)]$
- $M_2 = \left[(\lambda x.p_2\ q_2), \gamma \cdot \sigma \cdot \Phi^{\sigma_1}_{\gamma_1}(T\left\langle S/w\right\rangle) \cdot \ell(Q_2)\right]$
  donde $\Phi^{\sigma_1}_{\gamma_1}(T\left\langle S/w\right\rangle) = f(\gamma_1, \sigma_1, c(T\left\langle S'/x\right\rangle)) \cdot r(T\left\langle S'/x\right\rangle)$
  siendo $S' = [s, g(\sigma_1, c(S)) \cdot r(S)]$

$$M = [(\lambda x.(\lambda w.t\ s)\ q), \gamma \cdot \sigma \cdot \gamma_1 \cdot \sigma_1 \cdot \ell(T) \cdot \ell(S) \cdot \ell(Q)]$$

$$M_1 = \left[(\lambda w.t\ s)\left\langle q/x\right\rangle, \Phi^{\sigma}_{\gamma}(E\left\langle H/x\right\rangle)\right] \quad M_2 = \left[(\lambda x.p_2\ q_2), \gamma \cdot \sigma \cdot \Phi^{\sigma_1}_{\gamma_1}(T\left\langle S/w\right\rangle) \cdot \ell(Q_2)\right]$$

$$M_3 = \left[p_3\left\langle q_3/x\right\rangle, \Phi^{\sigma}_{\gamma}(P_3\left\langle Q_3/x\right\rangle)\right]$$

- **Primer resultado para** M_3 Como consecuencia de $M_2\ \rho\ M_3$, M_3 sería de la forma

  $$\left[p_3\left\langle q_3/x\right\rangle, \Phi^{\sigma}_{\gamma}(P_3\left\langle Q_3/x\right\rangle)\right]$$

  donde

  $$\Phi^{\sigma}_{\gamma}(P_3\left\langle Q_3/x\right\rangle) = f(\gamma, \sigma, c(P_3\left\langle B/x\right\rangle)) \cdot r(P_3\left\langle B/x\right\rangle) \tag{2.39}$$

  con

  $$B = [q, g(\sigma, c(Q_3)) \cdot r(Q_3)] \tag{2.40}$$

  y

  $$P_3\left\langle B/x\right\rangle = \left[t\left\langle s/w\right\rangle\left\langle q/x\right\rangle, \ell(\Phi^{\sigma_1}_{\gamma_1}(T\left\langle S/w\right\rangle))\left\langle B/x\right\rangle\right] \tag{2.41}$$

  Para escribir 2.39 de forma más detallada es necesario especificar la parte-b de 2.41 y esta a su vez depende de $T\left\langle S/w\right\rangle$, por tanto vamos a considerar los siguientes casos:

- a) si la parte-λ de $T\,\langle S/w\rangle$ es distinta de variable tenemos

$$\ell(\Phi^{\sigma_1}_{\gamma_1}(T\,\langle S/w\rangle)\,\langle B/x\rangle) = \\ f(\gamma_1,\sigma_1,c(T\,\langle S'/w\rangle\,\langle B/x\rangle))\cdot r(T\,\langle S'/w\rangle\,\langle B/x\rangle)$$

por lo que

$$\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = \\ f(\gamma,\sigma,f(\gamma_1,\sigma_1,c(T\,\langle S'/w\rangle\,\langle B/x\rangle)))\cdot r(T\,\langle S'/w\rangle\,\langle B/x\rangle) \tag{2.42}$$

lo cual únicamente puede ocurrir en dos casos:

  - a1) parte-λ de T no es una variable y S es cualquiera entonces 2.42 se escribe

$$\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = \\ f(\gamma,\sigma,f(\gamma_1,\sigma_1,c(T)))\cdot r(T\,\langle S'/w\rangle\,\langle B/x\rangle) \tag{2.43}$$

  - a2) $T=[w,n]$ y parte-λ de S no es una variable entonces 2.42 se escribe

$$\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = \\ f(\gamma,\sigma,f(\gamma_1,\sigma_1,s(n,g(\sigma_1,c(S)))))\cdot \\ r(T\,\langle S'/w\rangle\,\langle B/x\rangle) \tag{2.44}$$

- b) si la parte-λ de $T\,\langle S/w\rangle$ es una variable distinta de x, entonces

$$\ell(\Phi^{\sigma_1}_{\gamma_1}(T\,\langle S/w\rangle)\,\langle B/x\rangle) = f(\gamma_1,\sigma_1,c(T\,\langle S'/w\rangle)$$

por lo que

$$\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = \\ f(\gamma,\sigma,f(\gamma_1,\sigma_1,c(T\,\langle S'/w\rangle)))\cdot r(T\,\langle S'/w\rangle) \tag{2.45}$$

lo cual sólo puede ocurrir cuando:

  - b1) $T=[w,n]$ y $S=[z,m]$ con $z\neq x$

$$\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = f(\gamma,\sigma,f(\gamma_1,\sigma_1,s(n,g(\sigma_1,m)))) \tag{2.46}$$

  - b2) $T=[z,n]$ con $z\neq x,w$ y S cualquiera

$$\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = f(\gamma,\sigma,f(\gamma_1,\sigma_1,n)) \tag{2.47}$$

- c) si la parte-λ de $T\,\langle S'/w\rangle$ es la variable x entonces

$$\ell(\Phi^{\sigma_1}_{\gamma_1}(T\,\langle S/w\rangle)\,\langle B/x\rangle) = \\ f(\gamma_1,\sigma_1,s(c(T\,\langle S'/w\rangle),c(B)))\cdot r(B)$$

por lo que

$$\Phi^{\sigma}_{\gamma}(P_3\,\langle Q_3/x\rangle) = \\ f(\gamma,\sigma,s(f(\gamma_1,\sigma_1,c(T\,\langle S'/w\rangle),c(B))))\cdot r(B) \tag{2.48}$$

pero esto, que la parte-λ de $T\,\langle S'/w\rangle$ sea x, sólo puede ocurrir como consecuencia de ser

- c1) $T = [w, n]$ y $S = [x, m]$ que teniendo en cuenta 2.40 y 2.41 nos llevaría a que

$$\begin{array}{l}\Phi^{\sigma}_{\gamma}(P_3 \langle Q_3/x\rangle) = \\ f(\gamma, \sigma, s(f(\gamma_1, \sigma_1, s(n, g(\sigma_1, m))), g(\sigma, c(Q)))) \\ \qquad \cdot r(Q)\end{array} \tag{2.49}$$

- c2) $T = [x, n]$ y S cualquiera

$$\begin{array}{l}\Phi^{\sigma}_{\gamma}(P_3 \langle Q_3/x\rangle) = \\ f(\gamma, \sigma, s(f(\gamma_1, \sigma_1, n), g(\sigma, c(Q)))) \cdot r(B)\end{array} \tag{2.50}$$

- **Segundo resultado para $\mathbf{M_3}$**

  Como consecuencia de que $M_1 \ \rho \ M_3$, M_3 sería de la forma

$$\left[p_3 \langle q_3/x\rangle , \Phi^{\sigma}_{\gamma}(P_3 \langle Q_3/x\rangle)\right]$$

  aquí $M_1 = [p_1 \langle q_1/x\rangle , f(\gamma, \sigma, c(P_1 \langle A/x\rangle)) \cdot r(P_1 \langle A/x\rangle)]$ donde

$$\begin{array}{l}P_1 = [(\lambda w.t \ s), \gamma_1 \cdot \sigma_1 \cdot \ell(T) \cdot \ell(S)] \text{ y} \\ A = [q, g(\sigma, c(Q)) \cdot r(Q)]\end{array}$$

  se tendría por tanto que

$$\begin{array}{l}c(P_1 \langle A/x\rangle) = \gamma_1 \text{ y} \\ r(P_1 \langle A/x\rangle = \sigma_1 \cdot \ell(T \langle A/x\rangle) \cdot \ell(S \langle A/x\rangle)\end{array}$$

  es decir que M_1 se podría escribir como

$$[(\lambda w.ts) \langle q_1/x\rangle , f(\gamma, \sigma, \gamma_1) \cdot \sigma_1 \cdot \ell(T \langle A/x\rangle) \cdot \ell(S \langle A/x\rangle)]$$

  por ello $M_3 = \left[t \langle s/w\rangle \langle q/x\rangle , \Phi^{\sigma_1}_{f(\gamma,\sigma,\gamma_1)}((T \langle A/x\rangle) \langle S \langle A/x\rangle /w\rangle)\right]$ donde

$$\begin{array}{l}\Phi^{\sigma_1}_{f(\gamma,\sigma,\gamma_1)}((T \langle A/x\rangle) \langle S \langle A/x\rangle /w\rangle) = \\ \quad f(f(\gamma, \sigma, \gamma_1), \sigma_1, c(T \langle A/x\rangle \langle (S \langle A/x\rangle)'/w\rangle) \\ \qquad \cdot r(T \langle A/x\rangle \langle (S \langle A/x\rangle)'/w\rangle)\end{array} \tag{2.51}$$

  siendo

$$\begin{array}{l}(S \langle A/x\rangle)' = S' \langle A/x\rangle = \\ [s \langle q/x\rangle , g(\sigma_1, c(S \langle A/x\rangle)) \cdot r(S \langle A/x\rangle)]\end{array} \tag{2.52}$$

  En 2.51 $T \langle A/x\rangle$ toma varios valores dependiendo de T, consideramos los siguientes casos:

  - a) Si $T = [x, n]$ entonces $T \langle A/x\rangle = [q, s(n, g(\sigma, c(Q))) \cdot r(Q)]$
  - b) Si $T = [z, n]$ con $z \neq x$ entonces $T \langle A/x\rangle = [z, n]$

- c) Si parte-λ de T distinta de variable entonces

$$T\langle A/x\rangle = [t\langle q/x\rangle, c(T)\cdot r(T\langle A/x\rangle)]$$

Consideremos ahora los valores de $T\langle A/x\rangle\langle(S\langle A/x\rangle)'/w\rangle$ haciendo uso de los anteriores valores y dependiendo ahora de S

- En el caso a)

$$T\langle A/x\rangle\langle(S\langle A/x\rangle)'/w\rangle = [q, s(n, c(Q))\cdot r(Q)] \tag{2.53}$$

para cualquier valor de S (En Q no aparece libre w). Por lo que 2.51 se escribiría

$$f(f(\gamma,\sigma,\gamma_1),\sigma_1, s(n, g(\sigma, c(Q)))\cdot r(Q) \tag{2.54}$$

- En el caso b) hemos de considerar dos casos
  - b1) Si $z = w$, entonces

$$\begin{array}{l} T\langle A/x\rangle\langle(S\langle A/x\rangle)'/w\rangle = \\ [s\langle q/x\rangle, s(n, c((S\langle A/x\rangle)')\cdot r((S\langle A/x\rangle)')] \end{array} \tag{2.55}$$

Ahora hemos de considerar los distintos valores de S

  - ◇ b1a) Si $S = [x, m]$ entonces

$$(S\langle A/x\rangle)' = [q, g(\sigma 1, s(m, g(\sigma, c(Q))))\cdot r(Q)]$$

por lo que 2.51 es escribiría como

$$\begin{array}{l} f(f(\gamma,\sigma,\gamma_1),\sigma_1, s(n, g(\sigma_1, s(m, g(\sigma, c(Q))))))) \\ \quad \cdot r(Q) \end{array} \tag{2.56}$$

  - ◇ b1b) Si $S = [z, m]$ con $z \neq x$ entonces

$$(S\langle A/x\rangle)' = [z, g(\sigma_1, m)]$$

por lo que 2.51 se escribiría como

$$f(f(\gamma,\sigma,\gamma_1),\sigma_1, s(n, g(\sigma_1, m))) \tag{2.57}$$

  - ◇ b1c) Si la parte-λ de S es distinta de variable entonces

$$(S\langle A/x\rangle)' = [s\langle q/x\rangle, g(\sigma_1, c(S)\cdot r(S\langle A/x\rangle)]$$

por lo que 2.51 se escribiría como

$$f(f(\gamma,\sigma,\gamma_1),\sigma_1, s(n, g(\sigma_1, c(S)))\cdot r(S\langle A/x\rangle) \tag{2.58}$$

- b2) Si $z \neq w$, entonces

$$f(f(\gamma, \sigma, \gamma_1), \sigma_1, n) \tag{2.59}$$

- En el caso c)

$$T\left\langle A/x\right\rangle \left\langle (S\left\langle A/x\right\rangle)'/w\right\rangle = \\ [t\left\langle q/x\right\rangle \left\langle \left\langle s/x\right\rangle /w\right\rangle , c(T)\cdot r(T\left\langle A/x\right\rangle \left\langle (S\left\langle A/x\right\rangle)'/w\right\rangle)]$$

con lo cual 2.51 se escribiría

$$f(f(\gamma, \sigma, \gamma_1), \sigma_1, c(T)\cdot r(T\left\langle A/x\right\rangle \left\langle (S\left\langle A/x\right\rangle)'/w\right\rangle) \tag{2.60}$$

y ahora hemos de imponer condiciones a s, f y g para que coincidan:

- 2.54 con 2.50
- 2.56 con 2.49
- 2.57 con 2.46
- 2.58 con 2.44
- 2.59 con 2.47
- 2.60 con 2.43

para lo que sería suficiente que

$$f(f(a, b, c), d, s(y, g(d, s(x, e)))) \tag{2.61}$$

coincida con

$$f(a, b, s(f(c, d, s(y, g(d, x))), e)) \tag{2.62}$$

Aplicando 2.17 a 2.62 se podría escribir como

$$f(a, b, f(c, d, s(s(y, g(d, x)), e)))$$

que por 2.9 se escribiría como

$$f(a, b, f(c, d, s(y, s(g(d, x), e))))$$

que por 2.20 se escribiría como

$$f(a, b, f(c, d, s(y, g(d, s(x, e)))))$$

por lo cual, para que se diera la igualdad entre 2.61 y 2.62 bastaría con imponer la siguiente condición

$$f(a, b, f(c, d, e)) = f(f(a, b, c), d, e) \tag{2.63}$$

2.2 (Aplicando iv)) $M\ \rho\ M_2$ es de la forma

$$[(\lambda x.p\ q), \gamma \cdot \sigma \cdot \ell(P) \cdot \ell(Q)]\ \rho\ \left[p_2 \langle q_2/x \rangle, \Phi^{\sigma}_{\gamma}(P_2 \langle Q_2/x \rangle)\right]$$

donde $P\ \rho\ P_2$ y $Q\ \rho\ Q_2$. Entonces, utilizando la hipótesis de inducción, uno puede tomar nuevamente términos P_3 y Q_3 con tales que $P_1\ \rho\ P_3$,y $P_2\ \rho\ P_3$ (igual para las Q). Tomamos $M_3 = \left[p_3 \langle q_3/x \rangle, \Phi^{\sigma}_{\gamma}(P_3 \langle Q_3/x \rangle)\right]$

Caso 3.- Consideramos que $M\ \rho\ M_1$ es de la forma

$$[(p\ q), \gamma \cdot \ell(P) \cdot \ell(Q)]\ \rho\ [(p_1\ q_1), \gamma \cdot \ell(P_1) \cdot \ell(Q_1)]$$

como consecuencia de que $P\ \rho\ P_1$ y $Q\ \rho\ Q_1$, podemos distinguir dos subcasos, aplicando la propiedad iii) o la iv) de ρ

3.1 (Aplicando iii) $M\ \rho\ M_2$ es de la forma

$$[(pq), \gamma \cdot \ell(P) \cdot \ell(Q)]\ \rho\ [(p_2\ q_2), \gamma \cdot \ell(P_1) \cdot \ell(Q_1)]$$

donde $P\ \rho\ P_2$ y $Q\ \rho\ Q_2$. Entonces utilizando la hipótesis de inducción de una forma obvia, se puede tomar $M_3 = [(p_3\ q_3), \gamma \cdot \ell(P_3) \cdot \ell(Q_3)]$

3.2 (Aplicando iv) Consideremos que $P = [\lambda x.p_1, \sigma \cdot \ell(P_1)]$ y que la relación $M\ \rho\ M_2$ es de la forma

$$[(\lambda x.p_1\ q), \gamma \cdot \sigma \cdot \ell(P_1) \cdot \ell(Q)]\ \rho\ \left[p_2 \langle q_2/x \rangle, \Phi^{\sigma}_{\gamma}(P_2 \langle Q_2/x \rangle)\right]$$

donde $P_1\ \rho\ P_2$ y $Q\ \rho\ Q_2$. Por la propiedad ii) de ρ tenemos que $P_1 = [\lambda x.p_1, \sigma \cdot \ell(P_1)]$ con $P_1\ \rho\ P_1$. Utilizando la hipótesis de inducción de una forma obvia podemos tomar $M_3 = \left[p_3 \langle q_3/x \rangle, \Phi^{\sigma}_{\gamma}(P_3 \langle Q_3/x \rangle)\right]$

Caso 4.- Consideramos que $M\ \rho\ M_1$ es de la forma

$$[\lambda x.p, \gamma \cdot \ell(P)]\ \rho\ [\lambda x.p_1, \gamma \cdot \ell(P_1)]$$

como consecuencia de que $P\ \rho\ P_1$. Entonces $M_2 = [\lambda x.P_2, \gamma \cdot \ell(P_2)]$. Por la hipótesis de inducción podemos tomar $M_3 = [\lambda x.P_3, \gamma \cdot \ell(P_3)]$ □

2.7. Resumen

Recapitulemos los resultados obtenidos hasta ahora. Para que nuestro cálculo etiquetado con elementos de un conjunto $\mathbb{D}$ satisfaga el teorema de Church-Rosser dicho conjunto $\mathbb{D}$ deberá estar dotado de 3 operaciones: s, f y g tales para cualesquiera que sean los valores de $a, b, c, d, e \in \mathbb{D}$ se verifique:

$\underline{s(s(a,b),c) = s(a,s(b,c))}$ es la condición 2.9, obtenida al estudiar la sustitución, en la página 50.

$\underline{s(a, f(b, c, d)) = f(s(a, b), c, d)}$ es la condición 2.11, obtenida al estudiar las propiedades de la relación ρ, en la página 57.

$\underline{s(f(a, b, c), d) = f(a, b, s(c, d))}$ es la condición 2.17, obtenida al estudiar las propiedades de la relación ρ, en la página 59.

$\underline{s(g(a, b), c) = g(a, s(b, c))}$ es la condición 2.20, obtenida al estudiar las propiedades de la relación ρ, en la página 60.

$\underline{f(g(a, b), c, d) = g(a, f(b, c, d))}$ es la condición 2.38, obtenida al imponer que la relación ρ verifique la propiedad del diamante, en la página 64.

$\underline{f(a, b, f(c, d, e)) = f(f(a, b, c), d, e)}$ es la condición 2.63, obtenida al imponer que la relación ρ verifique la propiedad del diamante, en la página 69.

Estos resultados, muy generales, serán imprescindibles en el siguiente capítulo, cuando introduzcamos símbolos externos en nuestro cálculo, pues juntos con otros resultados que obtendremos, nos permitirán una mejor caracterización de $\mathbb{D}$. En el apéndice A se nuestra un intérprete del lambda cálculo_b que sigue exactamente las especificaciones dadas en este capítulo.

Capítulo 3

El lambda delta cálculo_b

3.1. Introducción

En este capítulo vamos ha seguir investigando las propiedades exigibles al conjunto $\mathbb{D}$ y a las operaciones que hemos establecido sobre él. Ahora nuestro hilo conductor será el mecanismo por el que el lambda-cálculo enriquece su sintaxis para permitir que en sus expresiones aparezcan símbolos tales como "1", "2", "+", ... etc. Para ello, sea $X \subseteq \Lambda$ un conjunto de formas normales cerradas. Generalmente se toma $X \subseteq C$, y sea $\mathfrak{f} : X \longrightarrow \Lambda$ una función "definida externamente". En el lambda cálculo clásico, a fin de representar $\mathfrak{f}$, se añade una regla llamada δ-regla definida de la siguiente forma:

1) Seleccionamos una constante de C y se le da una nombre, por ejemplo δ

2) Las siguientes reglas de contracción se añaden a las del lambda cálculo:

$$\delta M_1 \dots M_k \longrightarrow \mathfrak{f}(M_1, \dots, M_k)$$

donde $M_1, \dots, M_k \in X$

Para una $\mathfrak{f}$ dada, la fórmula anterior no es una regla de contracción, es un esquema de reglas. Llamaremos $\lambda\delta$-cálculo a la extensión resultante. La δ-reducción no es una noción absoluta pues depende de la elección de $\mathfrak{f}$. La nueva sintaxis se introduce por medio del reemplazamiento. Aquí no hay que tomar decisiones. Su uso permite alcanzar una λ-expresión pura, es decir una λ-expresión donde no aparecen nombres añadidos, tras un número finito de pasos que envuelven simples reemplazamientos. De esta forma no es necesario modificar el λ-cálculo. Suponemos además que los reemplazamientos se producen antes de comenzar la evaluación.

Ejemplo.- Para cada $n \in N$ seleccionamos una constante de C y la llamamos $\lceil n \rceil$. También seleccionamos las siguientes constantes:

$$\lceil mas \rceil \ , \lceil menos \rceil \ , \lceil mult \rceil \ , \lceil div \rceil \ , \lceil error \rceil$$

A continuación introducimos las siguientes δ-reglas:

$\lceil mas \rceil\ \lceil n \rceil\ \lceil m \rceil \longrightarrow \lceil n + m \rceil$

$\lceil menos \rceil\ \lceil n \rceil\ \lceil m \rceil \longrightarrow \lceil n - m \rceil$

$\lceil mult \rceil\ \lceil n \rceil\ \lceil m \rceil \longrightarrow \lceil n * m \rceil$

$\lceil div \rceil\ \lceil n \rceil\ \lceil m \rceil \longrightarrow \lceil n/m \rceil$

$\lceil div \rceil\ \lceil n \rceil\ \lceil 0 \rceil \longrightarrow \lceil error \rceil$

Este proceso es el que necesitamos introducir en nuestro cálculo. Ello nos planteará problemas relacionados con la representación de los números. Esperamos resolverlos estableciendo clases de equivalencia que nos permitan seleccionar algún elemento canónico y como consecuencia afinaremos las propiedades que deben cumplir las funciones s, f y g definidas sobre $\mathbb{D}$.

3.2. Sistemas numéricos

En el λ-cálculo clásico, se parte de que los naturales son objetos de una clase inductiva generada a partir de un elemento inicial, el 0, por medio de una función sucesor, indicada por $'$, de tal forma que los números forman una secuencia de elementos distintos $0, 0', 0'', \ldots$ y a continuación su busca establecer una representación para esta secuencia, lo cual puede hacerse de varias formas. Cualquier secuencia generada sistemáticamente serviría para este propósito; ya que es necesario que exista un elemento inicial, el Z_0, este se tomaría como el cero, y el iterador como la función sucesor, debemos asegurarnos que en cada aplicación del iterador el elemento resultante es distinto de los generados anteriormente. Hay otra formas de introducir los naturales, pero en esta tesis utilizaremos los naturales definidos de esta manera, son los llamados numerales de Church. Para un tratamiento más extenso ver [17].

3.2.1. Caso clásico

En general tenemos las siguientes definiciones [7]:

Definición 3.1 *Un sistema numérico es un secuencia* $\mathbf{d} = d_0, d_1, \ldots$ *formada por términos cerrados tales que para unos determinados* λ*-términos* S_d *y* $Cero_d$ *se tiene:*

- $(S_d\, d_n) = d_{n+1}$
- $(Cero_d\, d_0) = True$
- $(Cero_d\, d_{n+1}) = False$

para todo $n \in N$. S_d *y* $Cero_d$ *son la función sucesor y el test para cero de* $\mathbf{d}$.

Definición 3.2 *Se dice que* **d** *es un sistema numérico normal si cada* d_n *tiene una forma normal*

Cualquier sistema numérico **d** está determinado por d_0 y S_d. Por ello se puede escribir $\mathbf{d} = (d_0, S_d)$.

Definición 3.3 *Sea* **d** *un sistema numérico*

(i) Una función numérica $\psi : N^p \longrightarrow N$ *es* λ*-definible respecto a* **d** *si existe un* λ*-término* F *tal que* $\forall n_1, ..., n_p \in N$

$$F \ \lceil n_1 \rceil \dots \lceil n_p \rceil = \lceil \psi(n_1 \dots n_p \rceil$$

(ii) **d** *es adecuado sii toda función recursiva es* λ*-definible con respecto a* **d**.

Proposición.- *Sea* **d** *un sistema numérico de dice que* **d** *es adecuado sii*

$$\exists P_d \forall n \in N \ (P_d \ d_{n+1}) = d_n$$

3.2.2. Caso borroso

Para el caso borroso proponemos la siguiente definición:

Definición 3.4 *Un sistema numérico está formado por dos familias:*

· *La familia* $\mathfrak{d}_0$*, de términos del lambda cálculo-b, tal que su parte* λ *coincide con* d_0 *y su parte-b es cualquier lista adecuada a la estructura de* d_0.

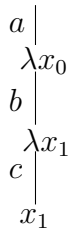
· *La familia* $\mathfrak{S}_d$*, de términos del lambda cálculo-b, tal que su parte* λ *coincide con* S_d *y su parte b es cualquiera adecuada a la estructura de* S_d

Esto nos lleva a plantearnos una cuestión previa:

> ?'Bajo que condiciones el sistema numérico, así definido, queda determinado por $\mathfrak{d}_0$ y $\mathfrak{S}_d$?. O dicho de otra forma ?'cualquier elemento del sistema numérico se puede generar por sucesivas aplicaciones de los elementos de $\mathfrak{S}_d$ a un elemento de $\mathfrak{d}_0$?

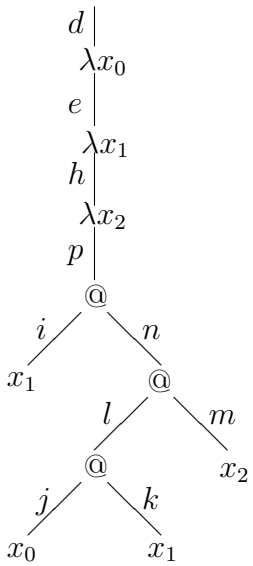
Para responderla tomemos como base los números de Church. Sea $Ch0$ un elemento de $\mathfrak{d}_0$ dado por:

$$Ch0 = [\lambda x_0.\lambda x_1.x_1, a \cdot b \cdot c]$$



y ChS un elemento de $\mathfrak{S}_d$ dado por:

$$ChS = [\lambda x_0.\lambda x_1.\lambda x_2.(x_1\ ((x_0\ x_1)\ x_2), d \cdot e \cdot h \cdot p \cdot i \cdot n \cdot l \cdot j \cdot k \cdot m]$$



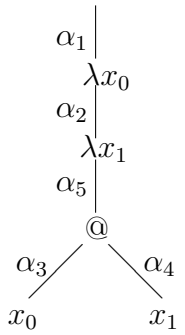
La secuencia generada a partir de ellos no presenta repeticiones, pues al escribir sus elementos en forma normal, las longitudes de sus λ-términos van aumentando. Si hubiésemos elegido otro sistema numérico tendríamos que haber demostrado esa distinguibilidad en los elementos o definir un operador $\mathbf{Z}$ para cada $n \geq 0$

$$\mathbf{Z}\lceil n \rceil = Z_n$$

para asegurar una biyección entre dichos numerales y los naturales.

Ahora, para responder a la cuestión, bastará mostrar que es posible generar cualquier sucesor de Ch_0. Sea A uno de tales elementos, dado por

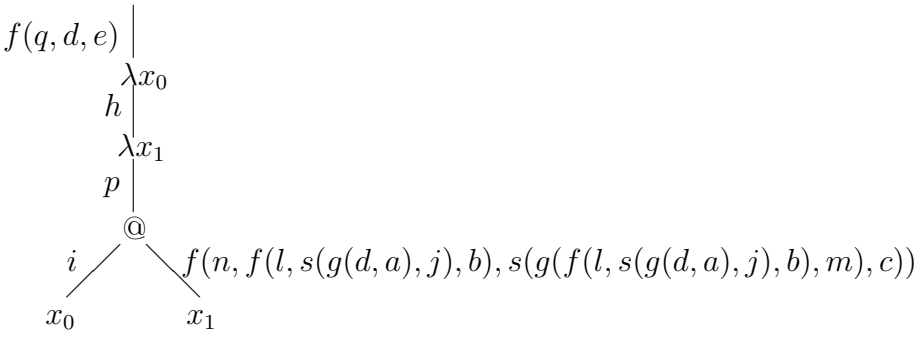
$$A = [\lambda x_0.\lambda x_1.(x_0\ x_1), \alpha_1 \cdot \alpha_2 \cdot \alpha_5 \cdot \alpha_3 \cdot \alpha_4 \cdot]$$



Considerando A como generado a partir de los elementos de $\mathfrak{S}_d$ y de $\mathfrak{d}_0$ se tendría que

$$A = [(Ch0\ ChS)q]$$

que tras realizar dos reducciones queda
$\lambda x_0 \lambda x_1 (x_1\ x_2), f(q,d,e) \cdot h \cdot p \cdot f(n, f(l, s(g(d,a),j),b), s(g(f(l, s(g(d,a),j),b),m),c))$



por lo que bastará elegir:

- h = α_2
- i = α_3
- p = α_5

y exigir que, en $\mathbb{D}$, las funciones s, f y g satisfagan las ecuaciones

- α_1 = f(q,d,e)
- α_4 = f(n,f(l,s(g(d,a),j),b),s(g(f(l,s(g(d,a),j),b),m),c))

Observar que si existe una solución entonces existen varias, pues el parámetro k puede tomar cualquier valor de $\mathbb{D}$. Como veremos en las dos secciones siguientes las funciones s, f y g satisfarán dichas condiciones.

Supongamos por tanto que nuestro sistema numérico esta formado por las familias $\mathfrak{d}_0$ y $\mathfrak{S}_d$ y que partimos de los numerales de Church, tendríamos entonces que para el numeral "0" dispondríamos de tantos como número de elecciones podamos hacer tomando los elementos de $\mathbb{D}$ de tres en tres, siendo elecciones distintas aquellas que difieran en algún elemento o en la ordenación de los mismos. Por tanto, si el conjunto $\mathbb{D}$ tiene d elementos la familia $\mathfrak{d}_0$ tendría d^3 elementos. Para el numeral "1" dispondríamos de d^5 elementos y así sucesivamente. Para evitar esta sobreabundancia y hablar de un numeral afectado de un índice de borrosidad, definiremos a continuación una función, que llamaremos de valuación y a partir de ella estableceremos una relación adecuada entre los distintos numerales cuyas partes-λ coincidan.

3.3. La valuación $\mathcal{V}$

Vamos a introducir una función $\mathcal{V}$ definida para todos los términos normales cerrados $\mathbf{T}$ tal que $\mathcal{V} : \ell(\mathbf{T}) \to \mathbb{D}$. La finalidad de esta función será la de establecer una relación de equivalencia entre los términos que nos permita llevar a cabo las δ-reducciones.

Aún cuando hasta este momento la parte-b la hemos representado por una lista, ha sido por comodidad y porque su estructura era deducible a partir de la parte-λ. Pero ahora, cuando vamos a estudiar $\mathcal{V}$, hemos de recuperar dicha estructura. Utilizamos la notación 3 de la página 47. En ella, la parte-b de un λ-término, obedece a la siguiente gramática

$$B \to \lambda LB \mid @LBB \mid L \tag{3.1}$$

$$L \to D \mid s(L,L) \mid f(L,L,L) \mid g(L,L) \tag{3.2}$$

Siendo $D \in \mathbb{D}$. Es decir aparecen:

los elementos de $\mathbb{D}$ que han intervenido en su construcción

los símbolos λ y @ que le dotan de una estructura

las operaciones s, f y g que guardan la historia de las transformaciones

Por tanto los elementos de 3.2 pertenecen a $\mathbb{D}$. Por ello proponemos la siguiente definición.

Definición 3.5 *Dado un término* $\mathbf{T} = [\mathbf{t}, \ell(\mathbf{T})]$ *del lambda cálculo_b definimos* $\mathcal{V}(\mathbf{T})$ *de la siguiente forma:*

1. *Si la parte-*λ *de* $\mathbf{T}$*,* $\mathbf{t}$*, es una variable entonces* $\ell(\mathbf{T})$ *es una palabra derivada de 3.2*

   $$\mathcal{V}(\mathbf{T}) = L$$

2. *Si* $\mathbf{T}$ *es un término de la forma* $[(w\ v), L \cdot \ell(W) \cdot \ell(V)]$*, entonces*

   $$\mathcal{V}(\mathbf{T}) = @L\mathcal{V}(\mathbf{V})\mathcal{V}(\mathbf{W})$$

   y lo reprentaremos por $L \oplus (\mathcal{V}(\mathbf{V}) \otimes \mathcal{V}(\mathbf{W}))$

3. *Si* $\mathbf{T}$ *es un término de la forma* $[\lambda y.w, L \cdot \ell(W)]$*, entonces*

   $$\mathcal{V}(\mathbf{T}) = @L\mathcal{V}(\mathbf{W})$$

   y lo reprentaremos por $L \odot \mathcal{V}(\mathbf{W})$

Si aceptamos la existencia en $\mathbb{D}$ de un elemento neutro para la operación s, al que llamaremos n, y hacemos uso de los resultados obtenidos anteriormente, introducimos la siguiente notación:

- Usaremos + para representar de forma infija la operación s.
- Teniendo en cuenta 2.11 y 2.17

  $$s(a, f(b, c, d)) = f(s(a, b), c, d)$$

  $$s(f(a, b, c), d) = f(a, b, s(c, d))$$

  y que s es asociativa, podemos poner $f(a, b, c) = a + f(n, b, n) + c$, por lo que introduciremos el símbolo $\mathfrak{f}$ definido por $\mathfrak{f}(a) = f(n, a, n)\ \forall a \in \mathbb{D}$
- Teniendo en cuenta 2.20

  $$s(g(a, b), c) = g(a, s(b, c))$$

  podemos poner $g(a, b) = g(a, n)+b$, por lo introduciremos el símbolo $\mathfrak{g}$ definido por $\mathfrak{g}(a) = g(a, n)\ \forall a \in \mathbb{D}$

3.3.1. La valuación $\mathcal{V}$ y los sistemas numéricos

Dado un sistema numérico $(\mathfrak{d}_0, \mathfrak{S}_d)$ y la función $\mathcal{V}$ existirá una interrelación entre ellos si queremos que nuestro cálculo tenga unas determinadas propiedades. Por ejemplo sería deseable que :

A) Si d_0 es cualquier elemento de la familia $\mathfrak{d}_0$ y S_d es el elemento de la familia $\mathfrak{S}_d$ tal que todos los elementos que aparecen en su parte-b son iguales a n, el neutro de s, entonces

$$\mathcal{V}(\ell(d_0)) = \mathcal{V}([\overbrace{(S_d(S_d(\ldots(S_d}^{m}\ d_0))\ldots))), \overbrace{n \cdot n \ldots \cdot n}^{m} \cdot \ell(d_0)]$$

B) Si $\mathbf{I}$ es la Identidad $\mathbf{I} = [\lambda x.x, n \cdot n]$

$$\begin{aligned}\mathcal{V}(((\overbrace{S_d\ (S_d\ (...\ (S_d}^{m}\ \ (\mathbf{I}\ d_0))\ldots)), \alpha\cdot \overbrace{n\cdot n\ldots\cdot n}^{m}\cdot \ell(d_0) = \\ \mathcal{V}((\mathbf{I}\ \overbrace{S_d\ (S_d\ (...\ (S_d}^{m}\ \ d_0), \overbrace{n\cdot n\ldots\cdot n}^{m}\cdot \alpha\cdot \ell(d_0)\end{aligned} \tag{3.3}$$

$\forall m \in \mathbb{D}$

Lo cual nos permitirá descartar funciones $\mathcal{V}$ arbitrarias así como determinar equivalencias en los términos cuyas partes-λ coincidan.

Hemos supuesto que la función $\mathcal{V}$ depende de $\lambda, @, s, \mathfrak{f}$ y $\mathfrak{g}$. El hacer que se verifiquen los puntos A) y B) nos dará un camino para poder simplificar esta dependencia. En particular para que se cumpla B) se debe cumplir que el término

$$\mathbf{T} \equiv [(\mathbf{I}\ \mathbf{I}), n] = [\lambda x.x, n + \mathfrak{f}(n) + \mathfrak{g}(n) + n \cdot n]$$

verifique que

$$\mathcal{V}(\ell(\mathbf{T})) = \mathcal{V}(\ell(\mathbf{I}))$$

para lo cual es necesario que

$$\mathfrak{f}(n) + \mathfrak{g}(n) = n \tag{3.4}$$

Veamos el apartado A). Para fijar ideas, supongamos que hemos elegido el sistema numeral de Church, la valuación de sus distintos elementos vendría dada por[1]:

- El **sucesor**

$$\begin{aligned}\mathbf{S} &= [S, \ell(\mathbf{S})] \\ &= [(\lambda x_0.(\lambda x_1.(\lambda x_2.(x_1((x_0\ x_1)x_2))))), n\cdot n\cdot n\cdot n\cdot n\cdot n\cdot n\cdot n\cdot n\cdot n]\end{aligned}$$

Su valuación sería, con una y otra notación:

$$\mathcal{V}(\ell(\mathbf{S})) = \begin{cases}\lambda n(\lambda n(\lambda n(@nn(@n(@nnn)n))))\\ n\odot(n\odot(n\odot(n\oplus(n\otimes(n\oplus(n\oplus n\otimes n)\otimes n))))\end{cases}$$

- El **cero**

$$\begin{aligned}\mathbf{0} &= [0, \ell(\mathbf{0})] \\ &= (\lambda x0(\lambda x1.x1)), a\cdot b\cdot c]\end{aligned}$$

[1] A partir de aquí hasta el fin de capítulo hacemos un uso intensivo de los resultados suministrados por el intérprete del lambda cálculo_b, cuya implantanción se hace en el apéndice A

Su valuación:

$$\mathcal{V}(\ell(\mathbf{0})) = \begin{cases} \lambda a \lambda b c \\ a \odot (b \odot c) \end{cases} \tag{3.5}$$

- El **uno**

$$\begin{aligned} \mathbf{1} &= [1, \ell(\mathbf{1})] \\ &= [(\mathbf{S}\ \mathbf{0}), e \cdot \ell(\mathbf{S}) \cdot \ell(\mathbf{0})] \\ &= [(\lambda x_0.(\lambda x_1.(x_0\ x_1))), A \cdot n \cdot n \cdot n \cdot W + n] \end{aligned}$$

Donde

$$A = e + \mathfrak{f}(n) + n \tag{3.6}$$

$$Z = n + \mathfrak{f}(n + \mathfrak{g}(n) + a) + b \tag{3.7}$$

$$W = n + \mathfrak{f}(Z) + c + \mathfrak{g}(Z) \tag{3.8}$$

$$\mathcal{V}(\ell(\mathbf{1})) = \begin{cases} \lambda A \lambda n @ n n (W + n) \\ A \odot (n \odot (n \oplus (n \otimes (W + n)))) \end{cases} \tag{3.9}$$

- El **dos**

$$\begin{aligned} \mathbf{2} &= [2, \ell(\mathbf{2})] \\ &= [(\mathbf{S}\ \mathbf{1}), e \cdot \ell(\mathbf{S}) \cdot \ell(\mathbf{1})] \\ &= [(\lambda x_0 (\lambda x_1.(x_0\ (x_0\ x_1)))), A \cdot n \cdot n \cdot n \cdot R \cdot T + d \cdot W + V + d] \end{aligned}$$

donde, además de las sustituciones anteriores:

- $T = n\ +\ \mathfrak{g}(\ n\ +\ \mathfrak{g}(\ n\ )\ +\ A\ )$
- $V = n\ +\ n\ +\ \mathfrak{g}(\ n\ +\ \mathfrak{f}(\ n\ +\ \mathfrak{g}(\ n\ )\ +\ A\ )\ +\ n\ )$
- $R = n\ +\ \mathfrak{f}(\ n\ +\ \mathfrak{f}(\ n\ +\ \mathfrak{g}(\ n\ )\ +\ A\ )\ +\ n\ )\ +\ n$

$$\mathcal{V}(\ell(\mathbf{2})) = \begin{cases} \lambda A \lambda n @ n n @ R (T + n)(W + V + n) \\ A \odot (n \oplus n \otimes (R \oplus ((T + n) \otimes (W + V + n)))) \end{cases} \tag{3.10}$$

- El **tres**

$$\begin{aligned} \mathbf{3} &= [3, \ell(\mathbf{3})] \\ &= [(\lambda x_0.(\lambda x_1.(x_0\ (x_0\ (x_0\ x_1))))), A \cdot n \cdot n \cdot n \cdot R \cdot T + d \cdot R \cdot \\ &\quad T + T + n \cdot W + V + V + n] \end{aligned}$$

$$\mathcal{V}(\ell(\mathbf{3})) = \begin{cases} \lambda A\lambda n@nn@R(T+n)@R(T+T+n)(W+V+V+n) \\ A \odot (n \oplus n \otimes (R \oplus ((T+n) \otimes (R \oplus (T+T+n) \\ \qquad\qquad \otimes (W+V+V+n))))) \end{cases} \tag{3.11}$$

y en general

$$\begin{aligned} \mathbf{k} &= [k, \ell(\mathbf{k})] \\ &= [(\mathbf{S}\ \mathbf{k} - \mathbf{1}), e] \\ &= [(\lambda x_0.(\lambda x_1.\ \overbrace{(x_0\ \ldots (x_0}^{k-1}\ (x_0\ x_1))\ldots))), \\ &\quad A \cdot n \cdot n \cdot n \cdot R \cdot T + n \cdot R \cdot T + T + n \cdot \ldots \cdot R \cdot \overbrace{T + \ldots + T}^{k-2} + n \cdot \\ &\quad R \cdot \overbrace{T + \ldots + T}^{k-1} + n \cdot W + \overbrace{V + \ldots + V}^{k-1} + n] \end{aligned}$$

$$\mathcal{V}(\ell(\mathbf{k})) = \begin{cases} \lambda A\lambda n@nn@R(T+n)(@R(T+T+n)(\ldots(@R(\overbrace{T+\ldots+T}^{k-2} \\ \qquad +n)(@R(\overbrace{T+\ldots+T}^{k-1}+n)(W+\overbrace{V+\ldots+V}^{k-1}+n))\ldots) \\ A \odot (n \oplus n \otimes (R \oplus ((T+n) \otimes (R \oplus (T+T+n) \otimes \ldots (R \oplus \\ \qquad ((\overbrace{T+\ldots+T}^{k-2}+n) \otimes (R \oplus ((\overbrace{T+\ldots+T}^{k-1}+n) \otimes \\ \qquad\qquad (W+\overbrace{V+\ldots+V}^{k-1}+n)))))\ldots)))) \end{cases} \tag{3.12}$$

3.3.2. Condiciones para la valuación $\mathcal{V}$

Ahora podemos plantear la cuestión: ¿Cuales son las restricciones a imponer a a, b, c, n y e y cuales serían las operaciones λ, @, $+$, $\mathfrak{f}$ y $\mathfrak{g}$ para que se verifique $\mathcal{V}(\mathbf{0}) = \mathcal{V}(\mathbf{1}) = \ldots = \mathcal{V}(\mathbf{k})$?

Recordemos que n y e son iguales y coinciden con el neutro de $+$. Una solución trivial sería, considerando que el conjunto $\mathbb{D}$ tiene un orden completo, tomar todas las funciones igual a la función mínimo y no imponer restricciones a a, b y c

Una solución más general sería la siguiente:
Teniendo en cuenta que las valuaciones de T, V y R dependen exclusivamente de la valuación de A, por ello si hacemos que:

- $\mathcal{V}(A) = n$

- $\mathcal{V}(Z) = a + b$
- $\mathcal{V}(W) = a + b + c$

para ello basta con que:

- $\mathfrak{f}(b) = b \ \forall b \in \mathbb{D}$, es decir f(n,b,n) es la función identidad sobre el segundo parámetro
- $\mathfrak{g}(b) = n \ \forall b \in \mathbb{D}$, es decir g(b,n) es la función constante nula

con ello además se cumpliría 3.4. Y entonces exigir que:

- la función λ coincida con s
- $\oplus$ y $\otimes$ deben ser funciones cuyo neutro coincida con el neutro de $+$ (ejemplo la función min y producto o la función máx y suma reunen estas condiciones)

con ello se cumpliría que $\mathcal{V}(\mathbf{0}) = \mathcal{V}(\mathbf{1}) = \ldots = \mathcal{V}(\mathbf{k})$

3.4. La relación $\mathcal{R}$

Una vez fijadas las funciones $+, \oplus$ y $\otimes$ queda fijada la función de valuación $\mathcal{V}$, la representaremos por $\mathcal{V}_{(+,\oplus,\otimes)}$, o por $\mathcal{V}$ cuando no haya lugar a confusión, la cual a su vez establece una relación en el conjunto de los términos cerrados en forma normal que definimos a continuación.

Definición 3.6 *Decimos que los términos* $A = [a, \ell(A)]$ *y* $B = [b, \ell(B)]$ *están* $_{(+,\oplus,\otimes)}$*-relacionados y lo representaremos por*

$$A \, \mathcal{R}_{(+,\oplus,\otimes)} \, B$$

si y sólo si $a \equiv b$ *y* $\mathcal{V}_{(+,\oplus,\otimes)}(\ell(A)) = \mathcal{V}_{(+,\oplus,\otimes)}(\ell(B))$

Definición 3.7 *Dado un término* $A = [a, \ell(A)]$ *tal que* $\mathcal{V}_{(+,\oplus,\otimes)}(\ell(A)) = \phi$*, un representante de la clase de equivalencia será* $[a, \{\phi\}]$ *o simplemente* $[a, \phi]$ *donde* $\{\phi\}$*, es una lista no vacía cuya primer elemento es* ϕ *y los restantes son iguales a* n*, el elemento neutro de* $+, \oplus$ *y* $\otimes$*.*

Teorema 3.1 *Si* $P_1 \, \mathcal{R}_{(+,\oplus,\otimes)} \, P_2$ *y* $Q_1 \, \mathcal{R}_{(+,\oplus,\otimes)} \, Q_2$ *son tales que* $[(p_1 \ q_1), \gamma \cdot \ell(P_1) \cdot \ell(Q_1)]$ *se* β*-reduce al término* $C_1 = [c_1, \ell(C_1)]$*, entonces* $[(p_2 \ q_2), \gamma \cdot \ell(P_2) \cdot \ell(Q_2)]$ *se* β*-reduce también al término* $C_2 = [c_2, \ell(C_2)]$ *con* $c_1 \equiv c_2$*,* c_1 *es una forma normal cerrada y* $C_1 \, \mathcal{R}_{(+,\oplus,\otimes)} \, C_2$

Demostración.- Este teorema es trivial en el caso clásico, en nuestro caso vamos a estudiarlo para ver que condiciones hay que imponer a $+, \oplus$ y $\otimes$ para que $\mathcal{V}_{(+,\oplus,\otimes)}(\ell(C_1)) = \mathcal{V}_{(+,\oplus,\otimes)}(\ell(C_2))$. Suponemos que

$P_1 = [\lambda x.m, \sigma_1 \cdot \ell(M_1)]$

$P_2 = [\lambda x.m, \sigma_2 \cdot \ell(M_2)]$

donde $\lambda x.m$ es una f.n.c. y siendo $\sigma_1 + \mathcal{V}_{(+,\oplus,\otimes)}(\ell(M_1)) = \sigma_2 + \mathcal{V}_{(+,\oplus,\otimes)}(\ell(M_2))$

$Q_1 = [q, \ell(Q_1)]$ y

$Q_2 = [q, \ell(Q_2)]$

donde q es una f.n.c. y siendo $\mathcal{V}_{(+,\oplus,\otimes)}(\ell(Q_1)) = \mathcal{V}_{(+,\oplus,\otimes)}(\ell(Q_2))$. Como la parte-$b$ de los términos obedece a la gramática dada anteriormente para estudiar las valuaciones de C_1 y C_2 consideraremos los tres casos posibles:

1) $\underline{B \to D}$. Es decir las partes-λ de C_1 y C_2 es una variable. Por tanto M_1 y M_2, deben ser de la forma $M_1 = [z, \phi_1]$, y $M_2 = [z, \phi_2]$.

   1a) Si $z = x$ y $Q_1 = Q_2 = [q, \delta]$, entonces

   $$C_1 = [q, f(\gamma, \sigma_1, s(\phi_1, g(\sigma_1, \delta)))]$$

   $$C_2 = [q, f(\gamma, \sigma_2, s(\phi_2, g(\sigma_2, \delta)))]$$

   Luego

   $$\mathcal{V}(C_1) = \gamma + \sigma_1 + \phi_1 + \delta$$

   $$\mathcal{V}(C_2) = \gamma + \sigma_2 + \phi_2 + \delta$$

   y como $\sigma_1 + \phi_1 = \sigma_2 + \phi_2$, entonces $\mathcal{V}_{(+,\oplus,\otimes)}(\ell(C_1)) = \mathcal{V}_{(+,\oplus,\otimes)}(\ell(C_2))$, sin necesidad de imponer ninguna condición.

   1b) Si $z \neq x$ y Q_1 y Q_2 son cualquiera, entonces $C_1 = [z, f(\gamma, \sigma_1, \phi_1)]$ y $C_2 = [z, f(\gamma, \sigma_2, \phi_2)]$ y nuevamente $\mathcal{V}_{(+,\oplus,\otimes)}(\ell(C_1)) = \mathcal{V}_{(+,\oplus,\otimes)}(\ell(C_2))$, sin necesidad de imponer ninguna condición.

2) $\underline{B \to \lambda L B}$. Es decir la parte-$\lambda$ de C_1 y C_2, es de la forma $\lambda w.z$. Por tanto M_1 y M_2 deben ser de la forma $M_1 = [\lambda w.z, \phi_1 \cdot \eta_1]$ y $M_2 = [\lambda w.z, \phi_2 \cdot \eta_2]$.

   2a) Si $z = x$ y $Q_1 = Q_2 = [q, \delta]$, entonces

   $$C_1 = [\lambda w.q, f(\gamma, \sigma_1, \phi_1) \cdot s(\eta_1, g(\sigma_1, \delta))]$$

   $$C_2 = [q, f(\gamma, \sigma_2, \phi_2) \cdot (\eta_2, g(\sigma_2, \delta))]$$

   Luego

   $$\mathcal{V}_{(+,\oplus,\otimes)}(\ell(C_1)) = \gamma + \sigma_1 + \phi_1 + \eta_1 + \delta$$

   $$\mathcal{V}_{(+,\oplus,\otimes)}(\ell(C_2)) = \gamma + \sigma_2 + \phi_2 + \eta_2 + \delta$$

   y como $\sigma_1 + \phi_1 + \eta_1 = \sigma_2 + \phi_2 + \eta_2$ entonces nuevamente $\mathcal{V}_{(+,\oplus,\otimes)}(\ell(C_1)) = \mathcal{V}_{(+,\oplus,\otimes)}(\ell(C_2))$, sin necesidad de imponer ninguna condición.

2b) Si $z \neq x$ y Q cualquiera, entonces

$$C_1 = [\lambda w.z, f(\gamma, \sigma_1, \phi_1) \cdot \eta_1]$$

$$C_2 = [\lambda w.z, f(\gamma, \sigma_2, \phi_2 \cdot \eta_2]$$

por lo que nuevamente se da la igualdad entre las valuaciones de C_1 y C_2 sin imponer ninguna condición.

3) $\underline{B \rightarrow @LBB}$. Es decir la parte-$\lambda$ de C_1 y C_2 es de la forma $(p\ q)$. Por tanto M_1 y M_2 deben ser de la forma

$$M_1 = [(e\ h), \phi_1 \cdot \eta_1 \cdot \nu_1]$$

$$M_2 = [(e\ h), \phi_2 \cdot \eta_2 \cdot \nu_2]$$

Ahora $\lambda\sigma_1(@\phi_1\eta_1\nu_1) = \lambda\sigma_2(@\phi_2\eta_2\nu_2)$, es decir

$$\sigma_1 + (\phi_1 \oplus (\eta_1 \otimes \nu_1)) = \sigma_2 + (\phi_2 \oplus (\eta_2 \otimes \nu_2)) \tag{3.13}$$

Sin pérdida de generalidad podemos suponer que $\phi_1 = \eta_1 = \nu_1 = n$ con lo cual 3.13 se transformaría en:

$$\sigma_1 = \sigma_2 + (\phi_2 \oplus (\eta_2 \otimes \nu_2)) \tag{3.14}$$

3a) Si $e = x, h \neq x$ y $Q_1 = Q_2 = [q, \delta]$ entonces

$$\begin{aligned} C_1 &= [m < q/x >, f(\gamma, \sigma_1, c(M < Q_1'/x >) \cdot r(M < Q_1'/x >)] \\ &= [(q\ h), f(\gamma, \sigma_1, \phi_1) \cdot s(\eta_1, g(\sigma_1, \delta)) \cdot \nu_1] \end{aligned}$$

$$C_2 = [(q\ h), f(\gamma, \sigma_2, \phi_2) \cdot s(\eta_2, g(\sigma_2, \delta)) \cdot \nu_2]$$

Luego $\mathcal{V}(C_1) = @(\gamma + \sigma_1 + \phi_1)(\eta_1 + \delta)\nu_1$ es decir

$$\mathcal{V}(C_1) = (\gamma + \sigma_1 + \phi_1) \oplus ((\eta_1 + \delta) \otimes \nu_1) \tag{3.15}$$

$$\mathcal{V}(C_2) = (\gamma + \sigma_2 + \phi_2) \oplus ((\eta_2 + \delta) \otimes \nu_2) \tag{3.16}$$

utilizando 3.14 la igualdad entre 3.15 y 3.16 se transformaría en la igualdad entre

$$(\gamma + (\sigma_2 + (\phi_2 \oplus (\eta_2 \otimes \nu_2))) \oplus \delta \tag{3.17}$$

y

$$(\gamma + \sigma_2 + \phi_2) \oplus ((\eta_2 + \delta) \otimes \nu_2) \tag{3.18}$$

3b) Si $e \neq x, h = x$ y $Q_1 = Q_2 = [q, \delta]$, entonces

$$\begin{aligned} C &= [m < q/x >, f(\gamma, \sigma_1, c(M < Q_1'/x >) \cdot r(M < Q_1'/x >)] \\ &= [(e\ q), f(\gamma, \sigma_1, \phi 1) \cdot \eta_1 \cdot s(\nu, g(\sigma_1, \delta))] \end{aligned}$$

$$C_2 = [(e\ q), f(\gamma, \sigma_2, \phi_2) \cdot \eta_2 \cdot s(\nu_2, g(\sigma_2, \delta))]$$

Luego $\mathcal{V}(C_1) = @(\gamma + \sigma_1 + \phi_1)\eta_1(\nu_1 + \delta)$ es decir

$$\mathcal{V}(C_1) = (\gamma + \sigma_1 + \phi_1) \oplus (\eta_1 \otimes (\nu_1 + \delta)) \tag{3.19}$$

$$\mathcal{V}(C_2) = (\gamma + \sigma_2 + \phi_2) \oplus (\eta_2 \otimes (\nu_2 + \delta)) \tag{3.20}$$

utilizando 3.14 la igualdad entre 3.19 y 3.20 se transformaría en la igualdad entre

$$(\gamma + (\sigma_2 + (\phi_2 \oplus (\eta_2 \otimes \nu_2))) \oplus \delta \tag{3.21}$$

y

$$(\gamma + \sigma_2 + \phi_2) \oplus (\eta_2 \otimes (\nu_2 + \delta)) \tag{3.22}$$

3c) Si $e = x$, $h = x$ y $Q_1 = [q, \delta]$, entonces

$$\begin{aligned} C_1 &= [m < q/x >, f(\gamma, \sigma_1, c(M < Q_1'/x >) \cdot r(M < Q_1'/x >)] \\ &= [(q\ q), f(\gamma, \sigma_1, \phi_1) \cdot s(\eta_1, g(\sigma_1, \delta)) \cdot s(\nu_1, g(\sigma_1, \delta))] \end{aligned}$$

$$C_2 = [(q\ q), f(\gamma, \sigma_2, \phi_2) \cdot s(\eta_2, g(\sigma_2, \delta)) \cdot s(\nu_2, g(\sigma_2, \delta))]$$

Luego $\mathcal{V}(C_1) = @(\gamma + \sigma_1 + \phi_1)(\eta_1 + \delta)(\nu_1 + \delta)$ es decir

$$\mathcal{V}(C_1) = (\gamma + \sigma_1 + \phi_1) \oplus ((\eta_1 + \delta) \otimes (\nu_1 + \delta)) \tag{3.23}$$

$$\mathcal{V}(C_2) = (\gamma + \sigma_2 + \phi_2) \oplus ((\eta_2 + \delta) \otimes (\nu_2 + \delta)) \tag{3.24}$$

utilizando 3.14 la igualdad entre 3.23 y 3.24 se transformaría en la igualdad entre

$$(\gamma + (\sigma_2 + (\phi_2 \oplus (\eta_2 \otimes \nu_2))) \oplus (\delta \otimes \delta) \tag{3.25}$$

y

$$(\gamma + \sigma_2 + \phi_2) \oplus ((\eta_2 + \delta) \otimes (\nu_2 + \delta)) \tag{3.26}$$

Los restantes casos no proporcionarían nuevas condiciones. Para que se den las igualdades entre

3.17 y 3.18

3.21 y 3.22

3.25 y 3.26

es suficiente con admitir que $\oplus$ y $\otimes$ coinciden con s y que esta es conmutativa. $\square$

3.5. Resumen

En este capítulo y en el anterior, y con el objetivo de introducir los conjuntos borrosos en el lambda cálculo hemos modificado el concepto de λ-término, añadiendole una etiqueta. Ello nos ha llevado a reformular la sustitución, la β-reducción y las δ-reglas, a fin de dar cuenta de como se manipulan dichas etiquetas. Esta ampliación contiene al λ-cálculo clásico como un caso particular. Si $\mathbb{D}$ es el conjunto al que pertenecen dichas etiquetas, en él debemos tener definidas tres operaciones: dos de ellas $\mathfrak{f}$ y $\mathfrak{g}$ triviales y una operación binaria s con las siguientes propiedades:

$$\left.\begin{array}{l} s : \mathbb{D} \times \mathbb{D} \to \mathbb{D} \\ s \text{ tiene un neutro} \\ s \text{ es asociativa} \\ s \text{ es conmutativa} \end{array}\right\} \tag{3.27}$$

en particular si $\mathbb{D} = [0, 1]$ entonces es suficiente, aunque no necesario, que s sea cualquier *t-norma* o *s-conorma* [32], [78], [77], [3], [81].

Ahora tenemos una herramienta suficientemente potente como para poder realizar cualquier cálculo multivaluado. En particular, notaciones como la utilizada para definir por extensión un conjunto borroso, serían fáciles de interpretar. No vamos a seguir este camino, pues al igual que sucede en el caso clásico, los cálculos realizados con el lambda cálculo están faltos de expresividad y son proclives a errores. Lo que haremos en los capítulos siguientes, de acuerdo con los objetivos marcados al principio de esta tesis, será utilizarlo para definir y formalizar lenguajes de programación que sean más expresivos, que tengan tipos adecuados y en los que poder introducir el no determinismo. Todo a fin de facilitar la tarea del programador que trata con datos imprecisos.

Capítulo 4

Semántica de lenguajes con datos borrosos

4.1. Introducción

Al diseñar, con la ayuda de la semántica denotacional, un lenguaje de programación que maneje datos borrosos, encontramos que tanto la semántica denotacional como el paradigma borroso manejan el concepto de información parcial y aproximación de la información. Pero desde dos puntos de vista bien diferentes.

En semántica denotacional: supongamos que tenemos el cpo de las funciones parciales $[\mathbb{N} \rightsquigarrow \mathbb{N}]$. En este cpo, la noción de aproximación viene dada por la inclusión entre los grafos de funciones. La función totalmente indefinida es el conjunto vacío, y una función totalmente definida tiene un grafo tal que, para cada $m \in \mathbb{N}$, existe algún $n \in \mathbb{N}$ tal que $(m, n) \in grafo(f)$ De forma que dadas dos funciones f_1 y f_2 decimos que f_1 es una aproximación de f_2 si $grafo(f_1) \subseteq grafo(f_2)$.

En el cálculo borroso: Un conjunto borroso F queda definido a partir de un referencial Ω y una aplicación, escrita μ_F, de Ω en $[0, 1]$. De forma que $\forall \omega \in \Omega$ $\mu_F(\omega)$ se interpreta como el grado de pertenencia de ω al conjunto borroso F. Esto permite modelizar categorías vagas del lenguaje natural definidas sobre un soporte objetivo con el que clasificarlo con la ayuda de dichas categorías. $\mu_F(\omega)$ expresará entonces hasta que punto el valor ω es compatible con el concepto F.[23]

Necesitamos integrar ambos puntos de vista y para ello hemos de revisar tanto las limitaciones de la teoría de dominios como las exigencias del cálculo con datos borrosos.

La teoría de dominios se estableció para tener espacios apropiados en los que definir funciones semánticas para el enfoque denotacional de la semántica de

los lenguajes de programación. Con este fin se deben cubrir dos necesidades: en primer lugar deben existir espacios de varios tipos distintos que permitan las diversas clases de construcciones semánticas y en segundo lugar la teoría ha de dar cuenta de de las propiedades computacionales de las funciones. Por ello como se señala en [51] "... los dominios son generalmente *planos* (y numerables). Notar en particular que las aproximaciones finitas numéricas a los números reales hechas por el ordenador no son representadas por valores relacionados con el orden de aproximación computacional de los dominios...". Todo ello nos lleva, lo que para nosotros es especialmente importante, a que, como se muestra en [30], el intervalo $[0, 1]$ no es un dominio. Propondremos sustituirlo por un conjunto numerable pero no *plano*.

Por otro lado en [56] podemos leer: " ... la sentencia condicional if P then S_1 else S_2 debe ejecutarse en paralelo. Esto significa que las sentencias S_1 y S_2 deben ejecutarse independientemente y a los resultados de S_1 denotados como $r(S_1)$ se les asigna el grado de pertenencia (de verdad) igual a $v(P)$...De forma similar, a los resultados de S_2 se les asigna el grado igual a $\neg v(P)$. En esta forma, el resultado obtenido por medio de la ejecución de las sentencias dadas puede interpretarse como un subconjunto borroso del conjunto de todos los resultados posibles". Algo similar podría decirse de la sentencia iterativa. Propondremos una estructura que permita formalizar este aspecto de los lenguajes.

Antes de mostrar nuestras propuesta, expondremos los conceptos fundamentales de la Semántica Denotacional clásica

4.2. Semántica denotacional clásica

Las especificaciones denotacionales son muy parecidas a los programas escritos en en lenguaje de programación funcional [73], [68] y [75]. En semántica denotacional se asignan significados a los programas a partir de los dominios semánticos cuya caracterización se debe a Scott. Para ello utilizó un lenguaje llamado LAMBDA, por medio de él se especifica el dominio universal $\mathbf{P}\omega = 2^{\omega}$ con $\omega = \{n | n \geq 0\}$. Una semántica denotacional aplica programas a términos de LAMBDA, que finalmente denota elementos de $\mathbf{P}\omega$.

El lenguaje LAMBDA no es muy expresivo. Por ello, generalmente se siguen dos caminos: uno, extenderlo sintácticamente, cada extensión es un teorema de la teoría LAMBDA. El otro camino es considerar el lenguaje extendido como un λ-cálculo tipado al cual se le puede dar una semántica en términos de LAMBDA.

A fin de asignar significados a los programas se usan las funciones de valuación, estas son colecciones de ecuaciones semánticas, cada de las cuales aplica una determinada construcción sintáctica en su significado. La estructura de las construcciones

sintácticas se da por medio de la sintaxis abstracta del lenguaje.

En los lenguajes de programación, como en los lenguajes naturales, se distingue la "sintaxis" de la "semántica". La *sintaxis* de un lenguaje de programación se ocupa únicamente con la *estructura* de los programas: qué programas son "correctos"; la conexión y relaciones entre los símbolos y frases que aparecen en ellos. La *semántica* se ocupa del *significado* de los programas correctos: su "comportamiento" cuando se ejecutan en un ordenador. La Semántica Denotacional es un marco para la descripción formal de la semántica de los lenguajes de programación. La idea principal de la semántica denotacional es que cada frase del lenguaje descrito produce una *denotación*: un objeto matemático que representa la contribución de la frase al significado de cualquier programa completo del que ella forma parte. Además, la denotación de cada frase se establece exclusivamente por la denotación de sus subfrases.

4.2.1. Sintaxis

Sintaxis concreta

Definición 4.1 *Sea L un conjunto (de etiquetas). Un árbol L-etiquetado t es un par $(l, (t_1 \dots t_n))$, donde $l \in L$, $n \geq 0$, y $t_1 \dots t_n$ es una cadena de árboles L-etiquetados. Se dice que t tiene la etiqueta l y ramas $t_1, \dots, t_n$*

Definición 4.2 *Un árbol de derivación de una gramática $G = (N, T, P, s_0)$ es un árbol con etiqueta s_0 y con finitas ramas etiquetadas con elementos de $(N \cup T)$, tal que si un nodo etiquetado con a tiene ramas etiquetadas con $x_1, \dots, x_n$, $n \geq 0$, entonces $a \in N$ y $(a, (x_1, \dots, x_n)) \in P$, o $a \in T$ y $n = 0$.*

Definición 4.3 *El conjunto de todos los árboles de derivación de la gramática G se representa por* $\mathbf{Árboles}_G$.

Definición 4.4 *Para cualquier $t \in$ $\mathbf{Árboles}_G$, $\mathbf{frontera}(t) \in T^*$ se define de la siguiente forma:*

$$\mathbf{frontera}(l, (t_1, \dots, t_n)) = \begin{cases} l & \text{si } n = 0 \text{ y } l \in T \\ \mathbf{frontera}(t_1) \dots \mathbf{frontera}(t_n) & \text{en otro caso} \end{cases}$$

Definición 4.5 *El lenguaje $\mathcal{L}(G) = T^*$ generado por la gramática $G(N, T, P, S)$ es:*

$$\mathcal{L}(G) = \{w \in T^* | \exists t \in \mathbf{Árboles}_G; w = \mathit{frontera}(t)\}$$

Si cualquier cadena de $\mathcal{L}(G)$ tiene más de un árbol de derivación, se dice que G es ambigua.

Sintaxis abstracta

Definición 4.6 *Sea S un conjunto (de tipos). Una S-tipada signatura Σ es una familia de conjuntos $\{\Sigma_{w,s}\}_{w\in S^*,s\in S}$ de operadores. Un Σ-álgebra consiste de una familia $\{A_s\}_{s\in S}$ de conjuntos y para cada operador $f \in \Sigma_{s_1\ldots s_n,s}$ una función total $f_A : A_{s_1} \times \cdots \times A_{s_n} \to A_s$*

Definición 4.7 *Un Σ-homomorfismo $h : A \to B$ (donde A y B son Σ-álgebras) es una familia $\{h_s\}_{s\in S}$ de funciones $h_s : A_s \to B_s$ tal que para $f \in \Sigma_{s_1\ldots s_n,s}$ y $a_i \in A_{s_i}$*

$$f_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)) = h_s(f_A(a_1, \ldots, a_n))$$

Definición 4.8 *Una Σ-álgebra I es inicial en una clase $\mathbf{C}$ de Σ-álgebras si y sólo si existe un único Σ-homomorfismo desde I a cada álgebra de $\mathbf{C}$.*

Definición 4.9 *Sea $G = (N, T, P, s_0)$ una gramática. La correspondiente signatura Σ_G se define de la siguiente forma:*

$$\Sigma_{G,s_1\ldots s_n,s} = \{p \in P \mid p = (s, (u_0 s_1 \ldots s_n u_n)); u_0 \ldots u_n \in T^*\}$$

para cada $(s_1 \ldots s_n) \in N^*$ y $s \in N$.

Ahora $\mathbf{Árboles}_G$ puede convertirse en una Σ-álgebra, que se denota por $\mathcal{A}(G)$ de la siguiente forma: tomamos $\mathcal{A}(G)_s$ igual a $\mathcal{A}(N, T, P, s)$ para cada $s \in N$. Para cada $p \in \Sigma_{G,s_1\ldots s_n,s}$ con $p = (s, (u_0 s_1 \ldots s_n u_n))$, donde $u_0 \ldots u_n \in T^*$, define una función

$$p_{\mathcal{A}(G)} : \mathcal{A}(G)_{s_1} \times \cdots \times \mathcal{A}(G)_{s_n} \to \mathcal{A}(G)_s$$

tomando, para todo $t_i \in \mathcal{A}(G)_{s_i}$, para $i = 1, \ldots, n$

$$p_{\mathcal{A}(G)}(t_1, \ldots, t_n) = (s, (u_0 t_1 \ldots t_n u_n))$$

En [51] se demuestra que: $\mathcal{A}(G)$ es inicial en la clase de todas las Σ_G-álgebras.

La Semántica Denotacional define la semántica de un lenguaje de programación a partir de su sintaxis abstracta.

4.2.2. Semántica

La semántica de un programa depende sólo del comportamiento objetivo que el programa produce cuando es ejecutado por un ordenador

Los ordenadores son mecanismos complejos y pueden observarse muchas cosas cuando él ejecuta un programa. Pero restringiremos nuestra atención a aquellos programas cuyo comportamiento se quiere que sean independientes de un ordenador particular. Tales programas generalmente escritos en un lenguaje de programación de alto nivel que no permiten a los programadores un control directo sobre algunos detalles del comportamiento físico. La semántica apropiada de estos programas es *independiente de la implementación*, consiste exclusivamente de aquellos aspectos de la ejecución de los programas que son comunes a todas las implementaciones. Tal semántica puede modelarse matemáticamente como una función entre la *entrada* y la *salida*. La representación concreta de las entradas y las salidas como cadenas de bit es, generalmente, dependiente de la implementación y por tanto ignorada, como también se ignora por ejemplo el tiempo que consume un programa para su ejecución. Pero la *terminación* es una propiedad que es independiente de la implementación y por tanto si es tenida en cuenta en la semántica.

Por tanto la semántica de un programa es un objeto matemático que modeliza los comportamientos independientes de la implementación de los programas. La semántica de un lenguaje de programación consiste de la semántica de todos sus programas.

Denotación

Un aspecto característico de la Semántica Denotacional es que se asignan objetos semánticos para todas las frases, no sólo para programas completos. El objeto semántico especificado por una frase se dice que es la *denotación* de la frase. La idea es que la denotación de cada frase representa la contribución de tal frase a la semántica de cualquier programa completo en el que ella ocurra.

La denotación de frases compuestas debe depender sólo de la denotación de sus subfrases. Este aspecto es llamado *composicionalidad*.

Funciones semánticas

Las funciones semánticas aplican frases (de la sintaxis abstracta) en sus denotaciones. La semántica de un lenguaje de programación puede especificarse definiendo una función semántica para cada clase de una frase.

Las entidades de la sintaxis abstracta tiene una estructura no ambigua. Por tanto las funciones semánticas pueden definirse inductivamente especificando, para cada constructor sintáctico, su denotación en términos de la denotación de sus componentes. En general la forma de escribir tales definiciones inductivas es dar un conjunto de ecuaciones semánticas, con una ecuación semántica para cada producción de la sintaxis abstracta.

Sea $a, a_1, \ldots, a_n$ un conjunto de símbolos no terminales que tienen asociadas las frases $s, s_1, \ldots, s_n$. Sean $\mathcal{F}_s$, $\mathcal{F}_{s_1}$,...,$\mathcal{F}_{s_n}$ las funciones semánticas aplicando frases de clase $s_{(i)}$ a su denotación. Entonces la ecuación semántica para la producción "$a ::= u_0 a_1 \ldots a_n u_n$" es de la forma:

$$\mathcal{F}_s[[u_0 a_1 \ldots a_n u_n]] = f(\mathcal{F}_{s_1}[[a_1]], \ldots, \mathcal{F}_{s_n}[[a_n]]) \tag{4.1}$$

La forma de combinar la denotación de las frases $a_1, \ldots, a_n$ se expresa usando cualquier notación disponible para especificar un determinado objeto, es lo que representa la f en la ecuación anterior.

4.2.3. Notación de dominios

Resumimos a continuación la notación para especificar dominios y sus elementos siguiendo a Gunter y Scott [30]. La sintaxis abstracta de la notación es la siguiente:

Expresiones de Dominios

$$\begin{aligned} d ::= {} & w \mid \mathbf{I} \mid \mathbf{O} \mid \mathbf{T} \mid \mathbf{N}_\perp \mid \\ & d_1 \to d_2 \mid d_1 \circ\!\!\to d_2 \mid \\ & d_1 \times \cdots \times d_n \mid d_1 \otimes \cdots \otimes d_n \mid \\ & d_1 + \cdots + d_n \mid d_1 \oplus \cdots \oplus d_n \mid \\ & d_\perp \mid d^* \mid d^\infty \mid d^\natural \end{aligned}$$

Variables de Dominios

$$w ::= \text{símbolos arbitrarios}$$

Expresiones

$$\begin{aligned} e ::= {} & x \mid \perp_d \mid \top \mid \mathbf{true} \mid \mathbf{false} \mid \\ & e_1 =_d e_2 \mid \text{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 \mid 0 \mid \mathbf{succ} \mid \\ & \lambda x \in d.e \mid e_1 e_2 \mid \mathbf{id}_d \mid e_1 \circ e_2 \mid \mathbf{fix}_d \mid \mathbf{stric}_d \mid \\ & (e_1, \ldots, e_n) \mid \langle e_1, \ldots, e_n \rangle \mid \mathbf{on}_i^d \mid \mathbf{smash}_d \mid \\ & [e_1, \ldots, e_n] \mid \mathbf{in}_i^d \mid \mathbf{up}_d \mid \mathbf{down}_d \mid \\ & \{\!|e|\!\} \mid e_1 \uplus e_2 \mid e^\natural \mid \mathbf{ext}_d \end{aligned}$$

Variables

$$x ::= \text{símbolos arbitrarios}$$

Más adelante daremos algunas convenciones para hacer desaparecer ambigüedades y varias abreviaciones comúnmente aceptadas

Comencemos con las expresiones de dominio d. Estas pueden incluir referencias a variables de dominio w, cuya interpretación es suministrada por el contexto de las

expresiones de dominio. Generalmente este contexto es un conjunto de ecuaciones de dominios de la forma

$$w_1 = d_1, \ldots, w_n = d_n$$

donde los w_i son distintos y sin que aparezcan otras variables en d_i. Como se demuestra en Gunter y Scott, para tales ecuaciones siempre existe una solución "mínima"

Las expresiones e pueden incluir referencias a variables x cuyo dominio es suministrado por el contexto. También se permiten definiciones auxiliares de la forma

$$x = e \in d$$

El alcance de una definición auxiliar es toda la especificación. (Definiciones auxiliares mutuamente dependientes pueden verse como abreviaciones de definiciones independientes en las que aparece el operador del menor punto fijo.)

Dominios básicos

- $\perp_d$ denota el menor elemento de un dominio d
- **I** es el dominio con un único elemento $\perp_{\mathbf{I}}$
- **O** es el dominio con dos elementos $\perp_{\mathbf{O}}$ y $\top$

Valores de Verdad

- **T** es el dominio con tres elementos **true**, **false** y $\perp_{\mathbf{T}}$
- $e_1 =_d e_2$ testa la igualdad de e_1 y e_2 en cualquier dominio plano d. El valor es $\perp_{\mathbf{T}}$ si ambos o alguno de ellos es $\perp_d$; en otro caso será **true** o **false**. (la monotonía y continuidad de $=_d$ se sigue del hecho de ser d plano; la igualdad puede no ser monótona sobre dominios no planos).
- **if** e_1 **then** e_2 **else** e_3 necesita que e_1 denote a un elemento de **T**, y que e_2 y e_3 denoten a elementos de algún dominio d. Entonces la expresión denota a e_2 si e_1 denota a **true**; denota a e_3 si e_1 denota **false**; y denota $\perp_d$ si e_1 denota $\perp_{\mathbf{T}}$

Utilizaremos también las funciones Booleanas usuales, extendidas estrictamente (en todos sus argumento) y escritas en forma infija.

Números naturales

- $\mathbf{N}_\perp$ es el dominio plano de los números naturales, formado por $\perp_{\mathbf{N}_\perp}$, 0, 1,$\cdots$ (no considerar el infinito)
- **succ** denota la extensión estricta de la función sucesor de $\mathbf{N}$ en $\mathbf{N}_\perp$.

En la práctica se permiten todas las funciones (computables) conocidas sobre los números naturales, extendidas estrictamente en todos sus argumentos a $\mathbf{N}_\perp$, y escritas usando notación infija.

Funciones

- $d_1 \to d_2$ denota el dominio de todas las funciones continuas del dominio d_1 en el dominio d_2. Se tiene que $f \subseteq_{d_1 \to d_2} g$ si y sólo si $f(x) \subseteq_{d_2} g(x)\ \forall x \in d_1$
- $\lambda x \in d.e$ denota la función continua f dada por $f(x) = e$. Si el contexto es claro eliminaremos $\in d$.
- $e_1 e_2$ denota el resultado $f(x)$ de aplicar la función $f : d \to d'$ denotada por e_1 al valor $x \in d$ denotado por e_2
- $\mathbf{id}_d$ denota a la función identidad del dominio d
- $e_1 \circ e_2$ denota la composición de las funciones $f_1 : d' \to d''$ y $f_2 : d \to d'$ denotadas por e_1 y e_2 respectivamente, de tal forma que $\forall x \in d$, $(e_1 \circ e_2)(x) = e_1(e_2(x))$
- $\mathbf{fix}_d$ denota al operador del menor punto fino del dominio d, que aplica cada función f de $d \to d$ a la menor solución de la ecuación $x = f(x)$
- $d_1 \circ\!\!\to d_2$ denota la restricción de $d_1 \to d_2$ a función estricta
- $\mathbf{stric}_d$ denota la función que aplica cada función de $d_1 \to d_2$ a la correspondiente función estricta en d, donde $d = d_1 \circ\!\!\to d_2$

Producto de Dominios

- $d_1 \times \cdots \times d_n$ denota el dominio de n-plas que generaliza el dominio de pares. Se tiene $(x_1, \ldots, x_n) \subseteq_{d_1 \times \cdots \times d_n} (y_1, \ldots, y_n)$ si y sólo si $x_i \subseteq_{d_i} y_i$ para $i = 1, \ldots, n$
- $(e_1, \ldots, e_n)$ denota la n-pla con componentes $e_1, \ldots, e_n$ para cualquier $n \geq 2$
- $\mathbf{on}_i^d$ denota la proyección de la i-ésima componente, siendo $d = d_1 \times \cdots \times d_n$

- $d_1 \otimes \cdots \otimes d_n$ el producto obtenido a partir del producto cartesiano identificando todas las n-plas que tengan en cualquier componente $\perp$. $\otimes$ preserva el carácter plano de los dominios. Si C y E son cpo $C \otimes E$ es el conjunto

$$\{(x, y) \in C \times E |\ x \neq \perp \text{ e } y \neq \perp\} \cup \{\perp_{C \otimes E}\}$$

- $\mathbf{smash}_d$ denota la función que aplica cada elemento de d donde $d = d_1 \times \cdots \times d_n$, al correspondiente elemento de $d_1 \otimes \cdots \otimes d_n$ dando $\perp$ si cualquiera de los componentes es $\perp$

Suma de Dominios

- $d_1 + \cdots + d_n$ denota el dominio d cuyos elementos son los elementos de d_i junto con un nuevo $\perp_d$. Elementos de d originados desde diferentes sumandos d_i no son comparables en d

- $d_1 \oplus \cdots \oplus d_n$ denota el dominio d donde los elementos $\perp$ de los d_i se identifican con $\perp_d$. $\oplus$ preserva el carácter plano de los dominios. Si C y E son cpo $C \oplus E$ es el conjunto

$$(\{(C - \{\perp_C\}) \times \{0\}) \cup (\{(E - \{\perp_E\}) \times \{1\}) \cup \{\perp_{C \oplus E}\}$$

- $\mathbf{in}_i^d$ denota la función de inyección que aplica los elementos de d_i en los correspondientes elementos de d, donde $d = d_1 \oplus \cdots \oplus d_n$ y $1 \leq i \leq n$

- $[e_1, \ldots e_2]$ denota el "análisis de casos" de las funciones $f_i : d_i \to d'$denotadas por e_i, aplicando $\mathbf{in}_i(x)$ al valor de $f_i(x)$ para $1 \leq i \leq n$ (pero aplicando $\perp$ en $\perp$).

Elevación de Dominios

- $d_\perp$ denota el dominio d' obtenido añadiendo a d un nuevo $\perp_{d'}$

- $\mathbf{up}_d$ denota la función que aplica cada elemento de d al elemento correspondiente de $d_\perp$

- $\mathbf{down}_d$ denota la función que aplica cada elemento de $d_\perp$ al elemento correspondiente de d

Listas

- d^* denota el dominio de listas de longitud finita, con componente de d. No se admite como componente $\perp$. Listas de longitudes distintas no son comparables.

- d^∞ denota el dominio de listas infinitas con componentes de d. Aquí la lista "vacía" es la lista infinita de $\perp$. Se tiene que $l_1 \subseteq_{d^\infty} l_2$ si y sólo si toda componente de l_1 aproxima a la correspondiente componente de l_2. Por tanto las lista vacía aproxima a cualquier otra lista.

Dominios potencia

- $d^\natural$ denota el dominio potencia "natural" (convexo, Plotkin). Sus elementos son clases de equivalencia de conjuntos.
- $\{|e|\}$ denota al elemento de $d^\natural$ correspondiente al conjunto $\{x\}$, donde e denota el elemento $x \in d$
- $e_1 \uplus e_2$ denota el elemento de $d^\natural$ correspondiente a la unión de X_1 y X_2, donde e_1 y e_2 denotan elementos de un dominio potencia $d^\natural$ correspondientes a los conjuntos X_1 y X_2
- $e^\natural$ denota la extensión de e a una aplicación de $d_1^\natural$ en $d_2^\natural$, donde e denota una función perteneciente a $d_1 \to d_2$
- $\mathbf{ext}_d$ extiende funciones de $d = d_1 \to d_2^\natural$ a $d_1^\natural \to d_2^\natural$

Es posible representar el "conjunto vacío" usando el dominio $\mathbf{O} \oplus d^\natural$

Convenciones notacionales

Los paréntesis se utilizan para indicar agrupamientos, las siguientes convenciones permite omitir algunos de ellos:

- Los constructores $\to$ y $\circ\!\!\to$ son asociativos por la derecha y tienen menor precedencia que $+, \oplus, \times$ y $\otimes$
- La aplicación es asociativa por la izquierda y tiene mayor precedencia que los otros operadores
- La composición $\circ$ es asociativa

Además cuando no haya peligro de ser ambiguo las siguientes operaciones pueden omitirse:

- isomorfismo entre w y d, cuando $w = d$ es una ecuación
- los siguientes isomorfismos
  - $d_1, + \cdots + d_n \cong (d_1)_\perp \oplus \cdots \oplus (d_n)_\perp$
  - $\mathbf{O} \cong \mathbf{I}_\perp$
  - $\mathbf{T} \cong (\mathbf{O} \oplus \mathbf{O})$, aplicando **true** en $\mathbf{in}_1(\top)$
  - $d^* \cong (\mathbf{O} \oplus (d \otimes d^*))$
  - $d^\infty \cong (d \times d^\infty)$
  - $d^\infty \cong (\mathbf{N}_\perp \circ\!\!\to d)$

- inyecciones $\mathbf{in}_i : d_i \circ\!\!\rightarrow d_1 \oplus \cdots \oplus d_n$
- "extensiones bottom" de funciones $f : d_i \rightarrow d'$ a suma de dominios:

$$[\ldots, \bot, f, \bot, \ldots] : d_1 \oplus \cdots \oplus d_n \circ\!\!\rightarrow d'$$

- las inclusiones de:
  - $d_1 \otimes \cdots \otimes d_n$ en $d_1 \times \cdots \times d_n$
  - $d_1 \oplus \cdots \oplus d_n$ en $d_1 + \cdots + d_n$ estricta
  - de $d \circ\!\!\rightarrow d'$ en $d \rightarrow d'$

Finalmente, la notación $\lambda(x_1 \in d_1, \ldots, x_n \in d_n.e)$ abrevia a

$$\lambda x \in d_1 \times \cdots \times d_n.(\lambda x_1 \in d_1.\cdots \lambda x_n \in d_n.e)(\mathbf{on}_1 x)\ldots(\mathbf{on}_n x)$$

(x no aparece en e). Y denota a la función f definida por $f(x_1, \ldots, x_n) = e$

4.3. Semántica denotacional borrosa

Hay dos aspectos de los lenguajes para los que vamos a construir una semántica que deseamos discutir: Por un lado queremos que dichos lenguajes sea un ejemplo de lo que Zadeh llama *sistemas humanísticos* [87]. Por otro lado el intervalo $[0, 1]$ es problemático cuando lo tratamos por medio de un lenguaje de programación.

- Los sistemas humanísticos son complejos, incluso contradictorios, en los que las personas juegan un papel fundamental a la hora de comunicar, informar, decidir e implantar políticas de evaluación de datos, [55], y que están sujetos al *principio de incompatibilidad de Zadeh*. Este principio afirma la relación inversa entre complejidad y precisión. Para dar cuenta de estos aspectos utilizaremos el no determinismo. En efecto, si los programas determinísticos pueden considerarse como representantes de una función parcial $z = P(x)$ que aplica un dominio de entrada D_x en un dominio de salida D_z, entonces un programa nodeterminístico puede considerarse como la representación de una función que aplica D_x en los subconjuntos de D_z. Dado un valor de entrada la ejecución de un programa determinístico se describe por un único camino (infinito, si la ejecución no termina), mientras que la ejecución de un programa nodeterminístico se describe por un árbol (posiblemente infinito).

  En general la descripción de los programas nodeterminísticos se lleva a cabo extendiendo la descripción de los programas determinísticos al generalizar las sentencias de asignación y ramificación. Se distinguen entre dos sentencias de asignación: la **∃-Asignación** ("asignar a una variable un valor arbitrario de un conjunto dado de posibles valores") y la **∀-Asignación** ("asignar a

una variable simultáneamente todos los valores de un conjunto dado de posibles valores"). Se distinguen dos tipos de ramificación: la $\vee$**-Ramificación** ("elegir una rama arbitraria de un conjunto dado de posibles ramas") y la $\wedge$**-Ramificación** ("elegir simultáneamente todas las ramas de un conjunto dado de posibles ramas").

- En relación con los números reales en general, y sobre todo con el intervalo $[0, 1]$, hay que dejar claro como definir y representar los reales. Hay tres formas básicas de hacerlo, en cada una de ellas $[0, 1]$ tiene unas características diferentes cuando se le considera a partir de la definición de los dominios:

  1. *Definir los reales en el sentido de la matemática clásica.* Desde este punto de vista el intervalo $[0, 1]$ no es un dominio, consultar [30].
  2. *Representar los reales por medio de cadenas finitas de dígitos.* Generalmente los reales se representan por cadenas de dígitos pertenecientes a algún conjunto de dígitos. La representación de números reales especifica una función que aplica cadenas a los números reales. Por ejemplo, en [34] los números de simple precisión en punto flotante son codificados en 32 bit usando 1 bit para el signo s, 8 bit para el exponente e, y 23 bit para la mantisa normalizada m sin la cabecera 1. El formato básico representa el número real

     $$(-1)^s 2^{e-127}(1.m)$$

     Es evidente que, cadenas finitas de dígitos sólo pueden representar a un subconjunto limitado, finito, de números reales lo que da lugar al concepto de errores de redondeo. Esto es aceptable para una amplia gama de aplicaciones. Sabemos, sin embargo, que la acumulación de errores de redondeo debido a un gran número de cálculos puede producir resultados incorrectos.

     En este caso el intervalo $[0, 1]$ será finito, y bien con su orden usual o bien con el orden inverso, constituirá un dominio.
  3. *Representar los reales por medio de cadenas infinitas de dígitos.* Si permitimos cadenas infinitas todos los números reales pueden representarse de forma exacta. Los dígitos de una cadena infinita de utilizan para construir una secuencia de intervalos reales anidados cuya longitud converge a cero. La intersección de estos intervalos es un conjunto con un único elemento, el número real representado. Las operaciones aritméticas básicas sólo serán computables si la representación es redundante, es decir si existe más de una representación para cada número real. A continuación mostramos algunas de las lineas de trabajo para la aritmética exacta de los reales:

a) *Basadas en secuencias infinitas de aplicaciones lineales* [8], [25]

b) *Basadas en expansiones de fracciones continuas* [58], [37]

c) *Basadas en transformaciones de Möbius* [80].

d) *Basadas en composición de transformaciones fraccionales lineales* [60]

En esta caso $[0, 1]$, con su orden usual, constituye un dominio continuo, consultar [24]

Por ello hacemos las siguientes propuestas:

1. Para el intervalo $[0, 1]$, aceptar que los reales vienen representados por medio de cadenas finitas de dígitos. La otra opción, que los reales vengan representados por cadenas infinitas de dígitos, aunque existen prototipos en varios lenguajes, C++ o CAML, su implantación es bastante ineficiente.

   En estas condiciones $[0, 1]$ tendrá la estructura de un dominio no plano, lo representaremos por $\langle \mathbb{G}, \sqsubseteq_{\mathbb{G}} \rangle$ o por $\mathbb{G}$ cuando el orden pueda sobrentenderse.

   Definición 4.10 *Sea el cpo* $(\mathbb{G}, \sqsubseteq_{\mathbb{G}})$. *La topología de Scott sobre* $\mathbb{G}$ *se define de la siguiente manera:*

   $O \subseteq \mathbb{G}$ *es abierto si*

   a*)* *Si* $x \in O$ *y* $x \sqsubseteq_{\mathbb{G}} y$, *entonces* $y \in O$

   b*)* *Si* $\cup X \in O$ *siendo* $X \subseteq \mathbb{G}$ *dirigido, entonces* $X \cap O \neq \emptyset$

   Por tanto $U_x = \{z \in \mathbb{G} \mid z \not\sqsubseteq_{\mathbb{G}} x\}$ es un abierto.

2. Para el lenguaje, hacer que sea no determinístico. Además, en principio, nos limitaremos a la **$\forall$-Asignación** y a la **$\wedge$-Ramificación**. Las otras opciones, considerar la **$\exists$-Asignación** y/o la **$\vee$-Ramificación**, mejorará la eficacia del lenguaje, pero no enriquecerá su semántica, por lo que las posponemos por ahora. Para introducir el no determinismo nos apoyaremos en los comandos guardados de Dijkstra [21].

4.3.1. Ampliaciones y modificaciones

Para dar cuenta de la semántica denotacional borrosa necesitamos:

1. **Revisión del concepto de "estado"**

   Muchos lenguajes de programación usan una estructura que existe independientemente de cualquier programa. Dicha estructura no se menciona explícitamente en la sintaxis del lenguaje, pero es posible construir frases que tengan acceso a ella y la modifiquen. Esta estructura es el *estado o almacén (store)*, lo representaremos por Σ, y a los lenguajes que la utilizan se les llama imperativos. El interés de los estados radica en que las sentencias se denotan por medio de funciones continuas pertenecientes a $\Sigma \to \Sigma_\perp$. En el caso clásico Σ es un cpo en el que dados dos estados o bien coinciden o bien no son comparables desde el punto de vista de cantidad de información que ambos conllevan, es decir es plano o discreto. Y sabemos que todas las funciones de un cpo discreto en un cpo son continuas. Por tanto las funciones que denotan a las sentencias son continuas.

   Definición 4.11 *Un conjunto parcialmente ordenado tiene la* condición de cadena ascendente (acc) *si no existe una secuencia infinita* $x_0 \subseteq x_1 \subseteq x_2 \subseteq \cdots$ *de elementos distintos.*

   Teorema 4.1 *Cualquier conjunto parcialmente ordenado que satisface la acc es un cpo*

   Consultar [31].

   En la semántica denotacional clásica existe un conjunto llamado *estados* definido por el espacio de funciones

   $$\Sigma' = \mathrm{Var} \to \mathbb{N}$$

   de las variables en los naturales. Sobre Σ' se tienen definidas dos funciones:

   - *Acceso* Si $\sigma' \in \Sigma'$, el valor ligado a la variable x en dicho estado viene dado por:

     $$\lambda v_{\in \mathrm{Var}} \lambda \sigma'_{\in [\mathrm{Var} \to \mathbb{N}]}.\sigma'(v)$$

   - *Modificación* Si $\sigma' \in \Sigma'$ entonces el estado resultante de modificar σ' por ligar la variable i al valor n viene dado por:

     $$\lambda v_{\in \mathrm{Var}} \lambda n_{\in \mathbb{N}} \lambda \sigma'_{\in [\mathrm{Var} \to \mathbb{N}]}.(\lambda v'_{\in \mathrm{Var}}.\text{if } v =_{\mathrm{Var}} v' \text{ then } n \text{ else } \sigma'(v'))$$

     y lo representaremos por $\sigma'[i \mapsto n]$

   En la semántica borrosa Dado el conjunto $\mathbb{G}$ construimos los dominios no planos, $\mathbb{N}_\mathbb{G}$ y $\mathbb{B}_\mathbb{G}$, los productos cartesianos de los naturales $\mathbb{N}$ y los booleanos $\mathbb{B}$ por $\mathbb{G}$.

Definición 4.12 *Dado un operador triangular s definimos:* $[f,i]:\mathbb{N}_{\mathbb{G}} \to \mathbb{N}_{\mathbb{G}}$ *de la siguiente forma:*

$$\forall [n,g] \in \mathbb{N}_{\mathbb{G}} \quad [f,i][n,g] = [f(n), s(i,g)] \tag{4.2}$$

Definición 4.13 *Un estado* σ*, es una función que para toda variable perteneciente a* Var *produce un elemento de* $\mathbb{N}_{\mathbb{G}}$*. Lo representaremos por* Σ*, el espacio de funciones* $\Sigma = Var \to \mathbb{N}_{\mathbb{G}}$*.*

Es decir si $\sigma \in \Sigma$ y $a \in Var$ es una variable, entonces $\sigma(a) \in \mathbb{N}_{\mathbb{G}}$ y lo representaremos por $(\sigma(a)_{\mathbb{N}}, \sigma(a)_{\mathbb{G}})$.
proponemos los definir los estado por

$$\Sigma = \text{Var} \to \mathbb{N}_{\mathbb{G}}$$

el espacio de las funciones de las variable en $\mathbb{N}_{\mathbb{G}}$.

Definición 4.14 *Sean* $\sigma_1, \sigma_2 \in \Sigma$ *dos estados, decimos que* $\sigma_1 \sqsubseteq_\Sigma \sigma_2$ *si y sólo si* $\forall x \in Var$

I $fst(\sigma_1(x)) = fst(\sigma_2(x))$
II $snd(\sigma_1(x)) \sqsubseteq_{\mathbb{G}} snd(\sigma_2(x))$

Es evidente que $\sqsubseteq_\Sigma$ es reflexiva, transitiva y antisimétrica. Por tanto $(\Sigma, \sqsubseteq_\Sigma)$ es un conjunto parcialmente ordenado.

Teorema 4.2 $(\Sigma, \sqsubseteq_\Sigma)$ *satisface la acc*

Demostración.- Sea $\{\sigma_i\}$ una familia de elementos de Σ tal que

$$\sigma_1 \sqsubseteq_\Sigma \sigma_2 \sqsubseteq_\Sigma \cdots \sigma_n \sqsubseteq_\Sigma \cdots$$

entonces $\forall i$ y $\forall x \in$Var se tiene que

I $fst(\sigma_i(x)) = fst(\sigma_{i+1}(x))$
II $snd(\sigma_i(x)) \sqsubseteq_{\mathbb{G}} snd(\sigma_{i+1}(x))$

y como $\mathbb{N}$ es plano, y $(\mathbb{G}, \sqsubseteq_{\mathbb{G}})$ satisface la acc, existe un k tal $\sigma_m(x) = \sigma_{m+1}(x)$ para todo $m \geq k$ □

Teorema 4.3 $(\Sigma, \sqsubseteq_\Sigma)$ *es un conjunto parcialmente ordenado completo* **(cpo)**

Demostración.- En efecto como $(\Sigma, \sqsubseteq_\Sigma)$ satisface la acc, aplicando el teorema 4.1 es un cpo. □

Teorema 4.4 *Sean D y E cpo. Si D satisface la acc entonces toda función monótona de D en E es continua.*

Demostración.- Hemos de probar que si $\{d_i\}$es una familia de elementos de D tal que

$$d_1 \sqsubseteq_D d_2 \sqsubseteq_D \cdots \sqsubseteq_D d_n \sqsubseteq_D \cdots$$

y f es una función monótona de D en E entonces $f(\sqcup d_i) = \sqcup f(d_i)$. Y en efecto si D satisface la acc, entonces existe un k tal que $\forall j, j \geq k\ d_k = d_j$, y por ser f monótona

$$f(d_1) \sqsubseteq_E f(d_2) \sqsubseteq_E \cdots \sqsubseteq_E f(d_n) \sqsubseteq_E \cdots$$

por tanto

$$f(\bigsqcup_i d_i) = f(d_k) = \bigsqcup_i f(d_i)$$

□

Luego toda función monótona de Σ en Σ es continua

Sobre Σ se definen dos funciones:

- *Acceso* Si $\sigma \in \Sigma$, el valor ligado a la variable x en dicho estado viene dado por:

$$\lambda v_{\in \mathrm{Var}} \lambda \sigma_{\in [\mathrm{Var} \to \mathbb{N}_{\mathbb{G}}]}.\sigma(v)$$

- *Modificación* Si $\sigma \in \Sigma$, $\iota \in \mathbb{G}$ y $s \in \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ es un operador triangular, entonces $\sigma[v \mapsto (n, s(\iota, g))]$ denota el estado tal que

$$\begin{array}{l} \lambda v_{\in \mathrm{Var}} \lambda (n,g)_{\in \mathbb{N}_{\mathbb{G}}} \lambda \sigma_{\in [\mathrm{Var} \to \mathbb{N}_{\mathbb{G}}]} \lambda \iota_{\in \mathbb{G}} \lambda s_{\in \mathbb{G} \times \mathbb{G} \to \mathbb{G}}. \\ \quad (\lambda v'_{\in \mathrm{Var}}.\text{if } v =_{\mathrm{Var}} v' \text{ then } (n, s(\iota, g)) \text{ else } \sigma(v')) \end{array} \tag{4.3}$$

Modificación de las funciones semánticas

2. **En la semántica denotacional clásica** se definen las funciones semánticas por medio de ecuación 4.1

$$\mathcal{F}_s[[u_0 a_1 \ldots a_n u_n]] = f(\mathcal{F}_{s_1}[[a_1]], \ldots, \mathcal{F}_{s_n}[[a_n]]) \tag{4.4}$$

En la semántica borrosa proponemos definir las funciones semánticas de forma similar, pero cada una de ellas necesitará dos parámetros: $\iota \in \mathbb{G}$ y s un operador triangular de forma que:

$$\begin{array}{l} (\mathcal{F}_s[[u_0 a_1 \ldots a_n u_n]], s, \iota) = \\ \quad (\lambda x_1 \ldots \lambda x_n.M)(\mathcal{F}_{s_1}[[a_1]], s, \iota) \ldots (\mathcal{F}_{s_n}[[a_n]], s, \iota) \end{array} \tag{4.5}$$

donde M es un lambda-término por medio del cual expresaremos las combinaciones de las denotaciones de las frases $a_1, \ldots, a_n$.

3. **Modificación del resultado de las expresiones**

   Como consecuencia de los dos apartados anteriores la función semántica para la evaluación de expresiones

   En la semántica denotacional clásica tiene funcionalidad:

   $$\mathcal{F}'_{\text{expr}}[[\text{Exp}]] = \Sigma' \to \mathbb{N}$$

   y que, por ejemplo, aplicada a una expresión: E_1 op E_2, sería:

   $$\mathcal{F}'_{\text{expr}}[[E_1 \text{ op } E_2]] = f_{\text{op}}(\mathcal{F}'_{\text{expr}}[[E_1]], \mathcal{F}'_{\text{expr}}[[E_1]])$$

   En la semántica borrosa proponemos sustituirla por:

   $$(\mathcal{F}_{\text{expr}}[[\text{Exp}]], s) = \Sigma \to \mathbb{N}_{\mathbb{G}}$$

   Donde s es un operador triangular.

   Más interesantes son las funciones semánticas para la evaluación de expresiones booleanas

   En la semántica denotacional clásica se tiene:

   $$\mathcal{F}_{\text{expr\_bool}}[[\text{Exp\_bool}]] = \Sigma' \to \mathbb{T}$$

   En la semántica borrosa proponemos sustituirla por:

   $$(\mathcal{F}_{\text{expr\_fuz}}[[\text{expr\_fuz}]], s) = \Sigma \to \mathbb{B}_{\mathbb{G}}$$

   Donde s es un operador triangular.

4. **Modificación de la denotación de las sentencias**

   En la semántica denotacional clásica la denotación de una sentencia Sent es una función continua

   $$\mathcal{F}'_{\text{sent}}[[\text{Sent}]] = \Sigma'_{\perp} \to \Sigma'_{\perp} \tag{4.6}$$

   ya que el objeto de una sentencia es la modificación de los estados. Si $f = \mathcal{F}'_{\text{sent}}[[\text{Sent}]]$ se le puede aplicar el teorema del punto fijo, consultar [31], [84], obteniéndose que

   $$\text{fix}(f) = \bigsqcup_n f^n(\perp)$$

En la semántica borrosa va a tener este mismo objeto, pero además para dar cuenta del no determinismo, proponemos sustituirlo por

$$(\mathcal{F}_{\text{sent}}[[\text{Sent}]], s, \iota) = \Sigma_\perp \to \mathcal{P}(\Sigma) \tag{4.7}$$

donde ι representa el grado con el cual evaluamos la sentencia y s es el operador triangular. Es decir que la denotación de las sentencias, con un determinado operador triangular y con un grado determinado, van a ser las funciones continuas de los almacenes en $\mathcal{P}(\Sigma)$, cuyos elementos representan los conjuntos de posibles estados resultantes de la ejecución de la sentencia. Como sabemos hay varias formas de ordenarlos, elegimos el dominio potencia convexo o natural, los otros dominios potencia no reflejan adecuadamente la posibilidad de divergencia. Utilizaremos $\top$ para representar el conjunto vacío de estados. Por tanto la denotación de las sentencias sera:

$$(\mathcal{F}_{\text{sent}}[[\text{Sent}]], s, \iota) = \Sigma_\perp \to (\mathbf{O} \oplus \Sigma^\natural)$$

y esta diferencia es notable, pues la denotación dada en 4.6 para las sentencias, por ser Σ' plano, cualquier función definida de Σ' en Σ' es continua. Pero en este caso habrá que demostrar la continuidad de cada sentencia, lo que haremos una vez que se establezca el lenguaje.

4.4. Un lenguaje sencillo

Antes de mostrar nuestro lenguaje presentaremos los comandos guardados de Dijkstra.

4.4.1. Los comandos guardados

Llamaremos "comando guardado", a una lista de sentencias prefijadas por una expresión booleana: sólo cuando esta expresión es inicialmente cierta se ejecutan las sentencias de la lista. Para ello, dado un lenguaje, que contiene "otras sentencias" extenderemos su sintaxis por medio del siguiente trozo de gramática:

⟨comando guardado⟩ ::= ⟨guarda⟩ → ⟨lista guardada⟩

⟨guarda⟩ ::= ⟨expresión booleana⟩

⟨lista guardada⟩ ::= ⟨sentencia⟩ {;⟨sentencia⟩}

⟨comandos guardados⟩ ::= ⟨comando guardado⟩ {□⟨comando guardado⟩}

⟨alternativa⟩ ::= **case** ⟨comandos guardados⟩ **esac**

⟨repetición⟩ ::= **do** ⟨comandos guardados⟩ **od**

⟨sentencia⟩ ::= ⟨alternativa⟩ | ⟨repetición⟩ | "otras sentencias"

El punto y coma en la lista guardada tiene el significado usual: cuando hemos de ejecutar la lista guardada sus sentencias deben ejecutarse sucesivamente de izquierda a derecha; una lista guardada sólo se ejecuta en un estado tal que su guarda es cierta. Notar que un comando guardado por si mismo no es una sentencia: él es un componente de lo que hemos llamado comandos guardados a partir de los cuales se pueden construir sentencias. Si la parte derecha de la producción encabezada por comandos guardados consiste de más de un comando guardado, se les separa por medio de □

4.4.2. El lenguaje

Cuando se trabaja con la sintaxis abstracta, nos olvidamos de palabras y de árboles de derivación y trabajamos con conjuntos y funciones, poniendo de manifiesto que los importante en la sintaxis no son los símbolos sino la estructura. Por ello llamaremos *dominio sintáctico* a todos lo valores que tienen en común una determinada estructura sintáctica. Por otro lado vamos a utilizar una versión más legible de la sintaxis abstracta, es la debida a Strachey [73]. Se especifica la sintaxis de un lenguaje dando sus dominios sintácticos y sus reglas con la notación BNF.
El lenguaje que proponemos tiene la siguiente la sintaxis abstracta:

S ::= v := E | skip | abort | (S;S) | case G esac | do G od

G ::= B → S | (G □ G)

E ::= (n,d) | x | E op E

B ::= (true,d) | (false,d) | B Bop B | E rel E

con las siguientes categorías sintácticas

S ∈ Sentencias

G ∈ Comandos guardados

E ∈ Expresiones

B ∈ Expresiones booleanas

v ∈ Variables

n ∈ $\mathbb{N}$

g ∈ $\mathbb{G}$

Definición 4.15 *Sea* $Conf = (S \times \mathbb{G} \times \Sigma) \cup \Sigma$ *es el conjunto de* configuraciones *y definimos una* relación de transición, $\rightarrow \subseteq Conf \times Conf$ *y suponemos que no existen transiciones válidas de la forma* $\sigma \rightarrow C$ *donde* $C \in Conf$ *como la menor relación que satisface los siguiente axiomas y reglas:*

1. *a)* $\dfrac{\langle e, \sigma\rangle \Rightarrow_A \langle (n, d)\rangle}{\langle v := e, i, \sigma\rangle \rightarrow \langle [v \mapsto (n, s(i, d))]\sigma\rangle}$

2. *a)* $\dfrac{}{\langle \underline{skip}, i, \sigma\rangle \rightarrow \langle \sigma\rangle}$

3. *a)* $\dfrac{\langle S_1, i, \sigma\rangle \rightarrow \langle \sigma'\rangle, \langle S_2, i, \sigma'\rangle \Rightarrow_C \langle \sigma''\rangle}{\langle S_1; S_2, i, \sigma\rangle \rightarrow \langle \sigma''\rangle}$

4. *a)* $\dfrac{\langle G, i, \sigma\rangle \rightarrow \langle \sigma'\rangle}{\langle \underline{\text{case}}\ G\ \underline{\text{esac}}, i, \sigma\rangle \rightarrow \langle \sigma'\rangle}$

5. *a)* $\dfrac{\langle G, i, \sigma\rangle \rightarrow \langle \sigma''\rangle, \langle \underline{\text{do}}\ G\ \underline{\text{od}}, i, \sigma''\rangle \rightarrow \langle \sigma'\rangle}{\langle \underline{\text{do}}\ G\ \underline{\text{od}}, i, \sigma\rangle \rightarrow \langle \sigma'\rangle}$

   b) $\dfrac{\langle G, i, \sigma\rangle \rightarrow \langle \delta\rangle}{\langle \underline{\text{do}}\ G\ \underline{\text{od}}, \sigma\rangle \rightarrow \langle \delta\rangle}$

   donde δ se utiliza para señalar la no terminación.

6. *a)* $\dfrac{\langle be, \sigma\rangle \Rightarrow_B \langle true, d\rangle, \langle S, s(i, d), \sigma\rangle \rightarrow \langle \sigma'\rangle}{\langle be \rightarrow S, i, \sigma\rangle \rightarrow \langle \sigma'\rangle}$

7. *a)* $\dfrac{\langle be, \sigma\rangle \Rightarrow_B \langle (false, d)\rangle}{\langle be \rightarrow S, i, \sigma\rangle \rightarrow \langle \sigma\rangle}$

8. *a)* $\dfrac{\langle G_1, i, \sigma\rangle \rightarrow \langle \sigma_1\rangle, \langle G_2, i, \sigma\rangle \rightarrow \langle \sigma_2\rangle}{\langle (G_1 \square G_2), i, \sigma\rangle \rightarrow \langle \{\sigma_1, \sigma_2\}\rangle}$

   b) $\dfrac{\langle \mathrm{G}_1, i, \sigma\rangle \rightarrow \langle \delta\rangle, \langle G_2, i, \sigma\rangle \rightarrow \langle \delta\rangle}{\langle (G_1 \square G_2), i, \sigma\rangle \rightarrow \langle \delta\rangle}$

donde la funcionalidad de $\Rightarrow_A$ es

$$\Rightarrow_A : (\text{Expr} \times \Sigma) \rightarrow \mathbb{N}_{\mathbb{G}}$$

y su definición es la siguiente:

1. $\dfrac{}{\langle (n, d), \sigma\rangle \Rightarrow_A \langle (n, d)\rangle}$

2. $\dfrac{}{\langle x, \sigma\rangle \Rightarrow_A \sigma(x)}$

3. $\dfrac{\langle e_1, \sigma\rangle \Rightarrow_A \langle (n_1, d_1)\rangle, \langle e_2, \sigma\rangle \Rightarrow_A \langle (n_2, d_2)\rangle}{\langle e_1\ \mathbf{op}\ e_2, \sigma\rangle \Rightarrow_A \langle (n_1\ \mathbf{op}\ n_2, s(d_1, d_2))\rangle}$
   donde s es cualquier T-norma o S-conorma. Ver 3.5

Similarmente, el tipo para $\Rightarrow_B$ viene dado por:

$$\Rightarrow_B : (\text{BExpr} \times \Sigma) \to \mathbb{B}_{\mathbb{G}}$$

y su definición es la siguiente:

1. a) $\dfrac{}{\langle (true, d), \sigma \rangle \Rightarrow_B \langle (true, d) \rangle}$
   b) $\dfrac{}{\langle (false, d), \sigma \rangle \Rightarrow_B \langle (false, d) \rangle}$
2. $\dfrac{\langle be_1, \sigma \rangle \Rightarrow_B \langle (vb_1, d_1) \rangle, \langle be_2, \sigma \rangle \Rightarrow_B \langle (vb_2, d_2) \rangle}{\langle be_1 \ \mathbf{Bop} \ be_2, \sigma \rangle \Rightarrow_B \langle (vb_1 \ \mathbf{Bop} \ vb_2, s(d_1, d_2)) \rangle}$
3. $\dfrac{\langle e_1, \sigma \rangle \Rightarrow_A \langle (n_1, d_1) \rangle, \langle e_2, \sigma \rangle \Rightarrow_A \langle (n_2, d_2) \rangle}{\langle e_1 \ \mathbf{rel} \ e_2, \sigma \rangle \Rightarrow_B \langle (n_1 \ \mathbf{rel} \ n_2, s(d_1, d_2)) \rangle}$

4.4.3. Modificadores de estados

En los lenguajes de programación no determinista se consideran la colección de sentencias, cuya función es transformar los estados, son de la forma $\Sigma' \to \Sigma'_\perp$; en lugar de ello nosotros emplearemos las funciones multivaluadas $m : \Sigma \to \mathcal{P}(\Sigma_\perp)$ donde $\mathcal{P}(\Sigma_\perp)$ es un cpo de subconjuntos de $\Sigma_\perp$.

Definición 4.16 $\mathcal{P}(\Sigma_\perp) = \{x \mid x \subseteq \Sigma\}$ *es el conjunto potencia de* Σ*, parcialmente ordenado por inclusión,* $\subseteq$*.*

$\mathcal{P}(\Sigma_\perp)$ es un cpo algebraico y su topología de Scott es determinada por los conjuntos

$$O_e = \{x \mid e \subseteq x, e \text{ finito}\}$$

Definición 4.17 *Dados* $s, t \in \mathcal{P}(\Sigma_\perp)$ *definimos la operación binaria:*

$$s \uplus t = \{w \mid w \sqsubseteq u \cup v \text{ para algún } u \in s \text{ y } v \in t\}$$

Definición 4.18 *La función* $\{\!|\cdot|\!\} : \Sigma \to \mathcal{P}(\Sigma_\perp)$ *definida por*

$$\{\!|x|\!\} = \begin{cases} \{x\} & (x \neq \perp) \\ \perp_{\mathcal{P}(\Sigma_\perp)} & (x = \perp) \end{cases}$$

Proposición 4.5 *Sea* $(f, s, i) : \mathcal{P}(\Sigma_\perp) \to \mathcal{P}(\Sigma_\perp)$

1. *Si* (f, s, i) *es continua entonces* (f, s, i) *es monótona*
2. (f, s, i) *es continua si y sólo si* $(f, s, i)(x) = \cup\{(f, s, i)(e) \mid e \subseteq x, \ e \text{ finito}\}$ *donde si* $e = \{[n_1, g_1], \ldots, [n_m, g_m]\}$ *entonces*

$$(f, s, i)(e) = \{[f(n_1), s(i, g_1)], \ldots, [f(n_m), s(i, g_m)]\}$$

Demostración Sea $\{(m_i, s, g_i)\}_i$ una familia dirigida de aplicaciones continuas de Σ en $\mathcal{P}(\Sigma_\perp)$. Definimos

$$(m, s, g)(\sigma) = \bigsqcup_i (m_i, s, g_i)(\sigma) \tag{4.8}$$

como $\{(m_i, s, g_i)\}_i$ es dirigido, $\{(m_i, s, g_i)(\sigma)\}_i$ es dirigido para todo σ y por tanto $(m, s, g,)$ existe. Además para todo $X \subseteq \Sigma$ dirigido

$$(m, s, g)(\bigsqcup X) = \bigsqcup_i \bigsqcup_{x \in X} (m_i, s, g_i)(x) = \bigsqcup_{x \in X} \bigsqcup_i (m_i, s, g_i)(x) = \bigsqcup (m, s, g)(X)$$

□

Definición 4.19 *Llamaremos modificadores de estados, y lo representaremos por ME, a las funciones $(m, s, i) : \Sigma \to \mathcal{P}(\Sigma_\perp)$ que sean continuas, estrictas y multiplicativas*

Definición 4.20 *Establecemos un orden parcial en ME definiendo*

$$(m, s, i) \sqsubseteq (m', s, i') \text{ si y sólo si } \forall \sigma \in \Sigma \;\; (m, s, i)(\sigma) \sqsubseteq (m', s, i')(\sigma)$$

Proposición 4.6 *El orden parcial ME es un cpo*

Demostración.- En efecto:

- Su menor elemento es: $\lambda \sigma_{\in \Sigma}.\perp_{\mathcal{P}(\Sigma_\perp)}$
- Cualquier cadena $(m_0, s, i_0) \sqsubseteq (m_1, s, i_1) \sqsubseteq \ldots$ tiene la menor de las cotas superiores (m, s, i), donde $(m, s, i)(\sigma) = \sqcup (m_n, s, i_n)(\sigma)$. Ver 4.8 □

Definición 4.21 *La función $Apl : (ME \times \Sigma) \to \mathcal{P}(\Sigma_\perp)$ se define de la siguiente manera:*

$$Apl((m, s, i), x) = \begin{cases} \uplus \; \{(m, s, i)(\sigma) \mid \sigma \in x\} & (si\ x \neq \perp) \\ \perp & (en\ otro\ caso) \end{cases}$$

Proposición 4.7 *La función Apl es*

- *Continua en cada argumento.*
- *Estricta y multiplicativa en su segundo argumento*
- $\forall \sigma \in \Sigma \; Apl((m, s, i), \{\sigma\}) = (m, s, i)(\sigma)$

Demostración.- Ya que ME es el conjunto de aplicaciones continuas de Σ en $\mathcal{P}(\Sigma_\perp)$, y que tanto Σ como $\mathcal{P}(\Sigma_\perp)$ son cpo, basta aplicar la proposición 1.2.13 (pag. 13) de [7] □

Ahora se puede dar la semántica de los modificadores de estados. Para ello en primer lugar veamos que la función condicional definida sobre el cpo $\mathbb{B}_\mathbb{G}$ $(if \cdot then \cdot) : \mathbb{B}_\mathbb{G} \times ME \to ME$ donde $\forall \sigma \in \Sigma$

$$\lambda\sigma.(if\ [t,i]\ then\ (m,s,g))\sigma = \begin{cases} (m,s,s(i,g)) & (t = tt) \\ \sigma & (t = ff) \end{cases}$$

es continua

Definición 4.22 *Definimos* $Conv$: *deter-*$ME \to ME$ *por:*

$$Conv(m,s,i)(\sigma) = \{(m,s,i)(\sigma)\}$$

Definición 4.23 *Definimos* $Comp$: $ME \times ME \to ME$ *por:*

$$Comp((m,s,i),(m',s,i'))(\sigma) = Apl(((m' \circ m),s,max/min(i',i))(\sigma))$$

donde se elige max/min *de acuerdo con el orden considerado en* $\mathbb{G}$

Definición 4.24 *Definimos* $Cond$: $(\Sigma \to \mathbb{B}_\mathbb{G} \times ME) \to ME$ *por:*

$$Cond(p,(m,s,i)(\sigma)) = if\ (p(\sigma) = (tt,i'))\ then\ (m,s,s(i,i'))(\sigma)\ else\ \sigma$$

Definición 4.25 *Dados* $p \in \Sigma \to \mathbb{B}_\mathbb{G}$ *y* $(m,s,i) \in ME$ *se define la función continua,* $H \in ME \to ME$ *de la siguiente forma:*

$$\begin{array}{rcl} H(m',s,i') & = & \lambda\sigma.if\ (p(\sigma) = (tt,g)) \\ & & then\ Comp((m,s,i),(m',s,s(i',g)))(\sigma) \\ & & else\ \{\sigma\} \end{array}$$

Proposición 4.8 *Si* (f,s,k) *es punto fijo de* H*, entonces* $s = min/max$ *y* $k=$ *neutro de* s*.*

Demostración.- En efecto sea $(f,s,k) = \text{fix}H$ por tanto

$$\begin{array}{rcl} (f,s,k) & = & \lambda\sigma.\text{if } (p(\sigma) = (tt,g)) \\ & & \text{then } ((f \circ m),s,max(i,s(k,g)))(\sigma) \\ & & \text{else } \{\sigma\} \end{array}$$

y

$$\begin{array}{rcl} H(f,s,k) & = & \lambda\sigma.\text{if } (p(\sigma) = (tt,g)) \\ & & \text{then Comp}((m,s,i),((f \circ m),s,max(i,s(k,g))))(\sigma) \\ & & \text{else } \{\sigma\} \end{array}$$

es decir se debe verificar que:

$$((f \circ m), s, max(i, s(k, g))) = ((f \circ m \circ m), s, max(i, s(max(i, s(k, g)), g))$$

y por tanto

$$max(i, s(k, g)) = max(i, s(max(i, s(k, g)), g))$$

y esto sólo es posible si $s = min$ y $k =$ neutro de s □

Un resultado similar se obtiene a partir de 4.7. Si $[f, \iota] = [\mathcal{F}_{\text{sent}}[[\text{Sent}]], \iota]$, para calcular el punto fijo de $[f, \iota]$ podemos utilizar $[\Theta, \eta]$, donde Θ es un operador de punto fijo y η es el neutro de s. Por tanto $[\Theta, \eta][f, \iota] = [(\Theta)f, \iota]$ de tal forma que

$$[f, \iota][(\Theta)f, \iota] = [(f)(\Theta)f, s(\iota, \iota)] = [(\Theta)f, \iota]$$

y ello sólo es posible si s es max o min.

Por tanto podemos calcular el menor punto fijo de H como $\bigsqcup_k H^k(\bot_{ME})$

Definición 4.26 *Definimos* Do*:*$(\Sigma \to \mathbb{B}_{\mathbb{G}} \times ME) \to ME$ *por:*

$$\begin{aligned} Do(p, (m, s, i)) \;=\; & Y(\lambda(m', s, i').\lambda\sigma.if\ (p(\sigma) = (tt, g)) \\ & then\ Comp((m, s, s(i, g)), (m', s, i'))(\sigma) \\ & else\ \{\sigma\}) \\ & donde\ (m', s, i') \in ME\ y\ \sigma \in \Sigma \end{aligned}$$

En efecto podemos definir las funciones parciales $Do_k(p, (m, s, i)) = H^k(\bot_{ME})$ *y por tanto* $Do(p, (m, s, i)) = \bigsqcup Do_k(p, (m, s, i))$.

Definición 4.27 *Definimos* $Guarda : (\Sigma \to \mathbb{B}_{\mathbb{G}} \times ME)^2 \to ME$ *por:*

$$\begin{aligned} & Guarda(\langle p, (m, s, i)\rangle, \langle (m', s, i')\rangle) = \langle p \vee q, \lambda\sigma. \\ & \quad if\ (p(\sigma) = (tt, i_p)) \\ & \quad\quad then\ (if\ (q(\sigma) = (tt, i_q)) \\ & \quad\quad\quad\quad then\ (m, s, s(i, i_p))(\sigma) \cup (m', s, s(i', i_q))(\sigma)) \\ & \quad\quad\quad\quad else\ (m, s, s(i, i_p))(\sigma) \\ & \quad\quad else\ (if\ (q(\sigma) = (tt, i_q)) \\ & \quad\quad\quad\quad then\ (m', s, s(i', i_q))(\sigma)) \\ & \quad\quad\quad\quad else\ \bot)\ \rangle \end{aligned}$$

Las anteriores definiciones 4.22, 4.23, 4.24, 4.25, 4.26 y 4.27 se han llevado a cabo utilizando funciones continuas, por tanto todas ellas son continuas.

4.4.4. Semántica denotacional dinámica

Vamos a describir la semántica denotacional dinámica. En dicha semántica es irrelevante el asignar un símbolo para representar los errores, pues suponemos que los programas que los contengan ha sido rechazados por ilegales. Por ello reservamos $\perp$ para representar la *no terminación* y no se introducirá ningún símbolo para representar errores en todos los dominios e incluso no especificaremos el tratamiento de dichos errores. Esto se hará más adelante.

I Literales
Los literales (también llamados "literales constantes", o justamente "constantes") son símbolos (o frases) que, independientemente de donde aparezcan, siempre remiten al mismo conjunto de datos. Su denotación semántica es sencilla pero tediosa de especificar, por ello omitiremos bastantes detalles. Por ejemplo para la denotación de "true" y "false", se puede utilizar los valores **true** y **false** del dominio **T**. Sin embargo, para la denotación de los restantes literales, se debe tener en cuenta que, diferentes implementaciones, imponen:

Cotas distintas a la magnitud de los números "enteros"

Precisiones distintas para los números "reales"

Por ello, en la descripción semántica, se dejan tales parámetros como variables sin especificar. A continuación mostramos un esquema para la sintaxis abstracta de los literales:

(Literal)
L ::= B | N | G

Booleano
B ::= true | false

(Numeral)
N ::= *sin especificar*

Grado
G ::= *sin especificar*

Sus dominios serían:

a) Dominio **Bool** = **T**
Operaciones
- **true, false**: **Bool**
- **or, and**: **(Bool** $\otimes$ **Bool)** $\circ\!\!\rightarrow$ **Bool**
- **not**: **Bool** $\circ\!\!\rightarrow$ **Bool**

b) Dominio **Num** = *sin especificar*
Operaciones

- **cero, uno, ...**: **Num**
- **sum, dif, prod, div**: **(Num** $\otimes$ **Num)** $\circ\!\!\rightarrow$ **Num**

c) Dominio **Grd** $= [0, 1]$
Operaciones

- cerog, unog, cerocincog, ...: **Grd**
- Operador_triangular: **(Grd** $\otimes$ **Grd)** $\rightarrow$ **Grd**
  - *Dependientes estrategia*

d) Dominio **V** $=$ **Bool** $\oplus$ **Num** $\oplus$ **Grd**

La denotaciones para los literales serían:

$\mathcal{L}$: Literal $\rightarrow$ **V**

$\mathcal{B}$: Booleano $\rightarrow$ **Bool**

$\mathcal{N}$: Numeral $\rightarrow$ **Num**

$\mathcal{G}$: Grado $\rightarrow$ **Grd**

Los dominios para la denotación de literales. excepto para **Grad** que es totalmente ordenado, son planos y todos ellos numerables.

II Identificadores
Se supone que existe un dominio plano **Ide** que se corresponde con una clase llamada Identificador.

III Booleanos Borrosos
Dominio $\mathbb{B}_{\mathbb{G}}$ $=$ **Bool** $\otimes$ **Grd**

IV Números Borrosos
Dominio $\mathbb{N}_{\mathbb{G}}$ $=$ **Num** $\otimes$ **Grd**

V Valores Expresables
Para representar el resultado de la evaluación de expresiones se considera el
Dominio **VE** $= \mathbb{B}_{\mathbb{G}} \oplus \mathbb{N}_{\mathbb{G}}$.

VI Valores Denotables
Para representar el conjunto de valores que pueden ser "denotados" por identificadores se utiliza el
Dominio **VD** $= \mathbb{N}_{\mathbb{G}}$

VII Valores Almacenables
Para representar el conjunto de valores que pueden ser "almacenados" se utiliza el
Dominio **VA** $=$ **VD**

VIII ALMACÉN

Dominio **Alm** = **Ide** $\to$ **VA**

Operaciones

· acc_alm: **Ide** $\to$ **Alm** $\to$ **VA**

acc_alm = $\lambda \mathrm{i}_{\in \mathbf{Ide}}.\lambda\rho_{\in(\mathbf{Ide} \to \mathbf{VA})}.\ \rho(\mathrm{i})$

· mod_alm: **Ide** $\to$ **VA** $\to$ **Ind_Fuz** $\to$ **Alm** $\to$ **Alm**

mod_alm = $\lambda \mathrm{i}_{\in \mathbf{Ide}}.\lambda(v,g)_{\in \mathbf{VA}}.\lambda\iota_{\in \mathbf{Ind\_Fuz}}.\lambda\rho_{\in(\mathbf{Ide} \to \mathbf{VA})}.\ \rho[\mathrm{i} \mapsto (v, s(g,\iota)]$

4.4.5. Funciones de valuación

$\mathcal{S} : (S, Ind\_Fuz) \to \Sigma \to (\mathbf{O} \oplus \Sigma^{\natural})$

$[\mathcal{S}[[\mathrm{S}_1;\mathrm{S}_2]],\iota] = \mathbf{ext}([\mathcal{S}[[\mathrm{S}_2]],\iota])[\mathcal{S}[[\mathrm{S}_1]],\iota]$

$[\mathcal{S}[[\underline{\mathrm{skip}}]],\iota] = \lambda\rho.\{\!|\rho|\!\}$

$[\mathcal{S}[[\underline{\mathrm{abort}}]],\iota] = \lambda\rho.\{\!|\delta|\!\}$

$[\mathcal{S}[[\mathrm{v}:=\mathrm{E}]],\iota] = \lambda\rho.\{\!|(\mathrm{mod\_alm}\ [[\mathrm{v}]]\ (\mathcal{E}[[\mathrm{E}]]\rho)\ \iota\ \rho)|\!\}$

$[\mathcal{S}[[\underline{\mathrm{case}}\ \mathrm{G}\ \underline{\mathrm{esac}}]],\iota] = \lambda\rho.[\lambda x_{\in \mathbf{O}}.\{\!|\delta|\!\}, \mathbf{id}_{\Sigma^{\natural}}][\mathcal{G}[[\mathrm{G}]],\iota]\rho$

$[\mathcal{S}[[\underline{\mathrm{do}}\ \mathrm{G}\ \underline{\mathrm{od}}]],\iota] = \mathbf{fix}([\lambda f\lambda\rho.[\lambda x_{\in \mathbf{O}}.\{\!|\rho|\!\}, \mathbf{ext}(f)\ ]([\mathcal{G}[[\mathrm{G}]],\iota]\rho), \eta])$

$\mathcal{G} : (G, Ind\_Fuz) \to \Sigma \to (\mathbf{O} \oplus \Sigma^{\natural})$

$[\mathcal{G}[[\mathrm{G}_1 \ \square\ \mathrm{G}_2]],\iota] = \lambda\rho.[\mathcal{G}[[\mathrm{G}_1]],\iota]\rho \uplus [\mathcal{G}[[\mathrm{G}_2]],\iota]\rho$

$[\mathcal{G}[[\mathrm{B} \to \mathrm{S}]],\iota] = \lambda\rho.(\lambda[t,g].\mathbf{if}\ \ t\ \mathbf{then}\ \mathbf{in}_2([\mathcal{S}[[\mathrm{S}]],s(\iota,g)])\ \mathbf{else}\ \mathbf{in}_1\top)\mathcal{B}[[\mathrm{B}]]\rho$

$\mathcal{T} : G \to \Sigma \to \mathbb{B}_{\mathbb{G}}$

$\mathcal{T}[[\mathrm{G}_1 \ \square\ \mathrm{G}_2]] = \lambda\rho.\ (\mathcal{T}[[\mathrm{G}_1]]\rho \wedge \mathcal{T}[[\mathrm{G}_2]]\rho)$

$\mathcal{T}[[\mathrm{B} \to \mathrm{S}]] = \lambda\rho.\mathcal{B}[[\mathrm{B}]]\rho$

$\mathcal{E} : Expr \to \Sigma \to \mathbb{N}_{\mathbb{G}}$

$\mathcal{E}[[\mathrm{E}_1\ \mathbf{op}\ \mathrm{E}_2]] = \lambda\rho.\ \mathrm{let}\ [n_1,d_1] = \mathcal{E}[[\mathrm{E}_1]]\rho\ \mathrm{in\ let}\ [n_2,d_2] = \mathcal{E}[[\mathrm{E}_2]]\rho\ \mathrm{in}$
$[n_1\ \mathbf{op}\ n_2, s(d_1,d_2)]$

$\mathcal{E}[[\mathrm{I}]] = \lambda\rho.\ \mathrm{acc\_alm}\ [[\mathrm{I}]]\ \rho$

$\mathcal{E}[[(\mathrm{n},\mathrm{d})]] = \lambda\rho.\ [\mathrm{n},\mathrm{d}]$

$\mathcal{B} : BExpr \to \Sigma \to \mathbb{B}_{\mathbb{G}}$

$\mathcal{B}[[(\mathrm{true},\mathrm{d})]] = \lambda\rho.[\mathrm{true},\mathrm{d}]$

$\mathcal{B}[[(\mathrm{false},\mathrm{d})]] = \lambda\rho.[\mathrm{false},\mathrm{d}]$

$$\mathcal{B}[[\mathrm{E}_1 \textbf{ rel } \mathrm{E}_2]] = \lambda\rho.\ \text{let } [n_1, d_1] = \mathcal{E}[[\mathrm{E}_1]]\rho \text{ in let } [n_2, d_2] = \mathcal{E}[[\mathrm{E}_2]]\rho \text{ in } [n_1 \textbf{ rel } n_2, s(d_1, d_2)]$$

$$\mathcal{B}[[\mathrm{B}_1 \textbf{ Bop } \mathrm{B}_2]] = \lambda\rho.\ \text{let } [b_1, d_1] = \mathcal{B}[[\mathrm{B}_1]]\rho \text{ in let } [b_2, d_2] = \mathcal{B}[[\mathrm{B}_2]]\rho \text{ in } [b_1 \textbf{ Bop } b_2, s(d_1, d_2)]$$

4.5. Resumen

El lenguaje que hemos propuesto es un lenguaje con asignación al que le añadimos la sentencia condicional no determinística y un bucle multi-prueba no determinístico, el cual iterará mientras alguna prueba sea cierta. Ambas sentencias, la condicional y el bucle materializan la **∧-Ramificación**. Por ello el dominio de los posibles resultados de un cálculo es $\mathcal{P}(\Sigma)$. El significado de la secuenciación de dos sentencias $(S_1; S_2)$ sobre un almacén σ es el siguiente: S_1 obrará sobre σ produciendo un conjunto de almacenes. Cada uno de los almacenes de este conjunto se pasará a S_2 para producir a su vez otro conjunto de almacenes. Finalmente reuniremos todos estos conjuntos para producir un elemento de $\mathcal{P}(\Sigma)$. Por medio de ella hemos materializado la **∀-Asignación**.

Hemos utilizado un operador triangular s, como global y constante, por ello no lo hemos materializado más que en el momento de su utilización. La introducción del concepto de bloques, o mejor de abstracciones, funciones y procedimientos, nos permitirá elegir distintas normas triangulares. Con ello podremos superar la limitación encontrada en la proposición 4.8

Para este sencillo lenguaje hemos dado su semántica denotacional. Su sencillez es ideal para mostrar la aplicación de los conceptos básicos de la semántica denotacional borrosa. Pero esa misma sencillez lo hace poco adecuado para programar problemas reales, por ello en el próximo capítulo lo ampliaremos.

Capítulo 5

Ampliaciones

5.1. Introducción

En este capítulo vamos a mostrar que las propuestas hechas en los capítulos anteriores son realizables. Para ello partiremos del lenguaje propuesto en el capítulo anterior, y lo ampliaremos desde dos puntos de vista: bien enriqueciendo su estructuración, bien considerando la necesidades del cálculo borrosos. Con respecto a la primera, sería interesante ampliarlo para, de alguna forma, dar cuenta de:

1. Dotarlo de estructura de bloques
2. Dotarlo funciones y procedimientos
3. Incrementar el conjunto de sus sentencias
4. Incrementar sus tipos de datos.

Con respecto a la segunda, es interesante:

1. Elegir la manera de representar los conjuntos borrosos y las operaciones definidas sobre ellos
2. Poder definir variables lingüísticas

5.1.1. Bloques y Abstracciones

En el lenguaje del capítulo anterior existían tres elementos necesarios para la evaluación de una sentencia:

- Lo que hemos llamado el índice borroso **Ind_Fuz**. Se modificaba en las ramificaciones, como consecuencia del resultado de la evaluación booleana de la prueba.
- Alguna de las funciones s obtenida al elaborar el lambda cálculo_b. Es decir los operadores triangulares **T_op** necesarios para la evaluación de expresiones.

- Los ámbitos, que no se explicitaban claramente, porque dicho lenguaje tenía un único ámbito.

Al ampliar el lenguaje dotándolo con bloques, y en especial de abstracciones (funciones y procedimientos), hay que definir una estrategia para elegir estos elementos al invocar las abstracciones. Primero veremos los ámbitos, lo que hagamos con ellos nos servirá de guía para el **Ind_Fuz** y **T_op**.

Lo ámbitos

Virtualmente todos los lenguajes cuentan con alguna noción de contexto. El contexto en el que se utiliza una frase influye en su significado. En un lenguaje de programación, los contextos son los encargados de atribuir significado a los identificadores. En semántica denotacional, el contexto de una frase se modela por un mecanismo llamado *ámbito (environment)*. En el lenguaje del capítulo anterior este ámbito iba unido al almacén, dando lugar a una aplicación de los identificadores en los valores almacenables. En este capítulo, ese modelo tan simple se dividirá en dos componentes, el ámbito y el almacén.

Los ámbitos se utilizan como argumentos en las funciones de valuación. El significado de una sentencia se determinará, en principio, por la función:

$$\mathcal{S} : \mathbf{Sentencias} \rightarrow \mathbf{Amb} \rightarrow \mathbf{Ind\_Fuz} \rightarrow \mathbf{T\_op} \rightarrow \mathbf{Alm} \rightarrow \mathcal{P}(\mathbf{Alm})$$

de forma que para un determinado índice $\in$ **Ind_Fuz** y unos determinados operadores triangulares $\in$ **T_op**, el significado de una sentencia es una función **Alm** $\rightarrow$ $\mathcal{P}$(**Alm**) que es determinada una vez que el ámbito establece el contexto para la sentencia. Por tanto el ámbito pertenecerá al dominio

$$\mathbf{Ambito} = \mathbf{Identificadores} \rightarrow \mathbf{Valores\_Denotables}$$

donde **Valores_Denotables** es el dominio de todos los valores que un identificador pueden representar. Para un programa (de un lenguaje determinista) existirá un único almacén y varios ámbitos, los que sean necesarios para establecer los contextos de los diversos bloques, tales como funciones y procedimientos. Esto hace que existan dos posibilidades cuando se invoca una abstacción:

- Utilizar el ámbito activo en el momento de la definición de la abstracción.
- Utilizar el ámbito activo en el momento de la invocación de la abstracción.

Índice borroso y Operadores triangulares

El índice borroso y el operador triangular necesarios para la evaluación de cada sentencia son globales para cada bloque, con lo cual se tienen varias opciones pues, por ejemplo para el índice, podemos considerar:

1. Que las sentencias que componen el cuerpo de la abstracción toman como índice, el índice existente en el momento de la invocación (podríamos decir que tiene alcance dinámico).

2. Que las sentencias que componen el cuerpo de la abstracción toman como índice, el índice que se define en el momento de la declaración de la abstracción (podríamos decir que tiene alcance estático).

3. Incluso existe una tercera vía: que el índice para evaluar las sentencias del cuerpo de la abstracción, sea el resultado de operar, por medio de una T-norma o S-conorma, el índice que existe en el momento de la invocación con el índice que se define en el momento de la declaración de la abstracción.

Para los operadores triangulares ocurre algo parecido:

1. Que el operador triangular sea el utilizado en el momento de la invocación de la abstracción.

2. Que el operador triangular sea el establecido en el momento de la declaración de la abstracción.

5.1.2. Control

Vamos a introducir las sentencias **if** E **then** S **el** S **end if** y **while** E **do** S **end do**. Las sentencias **case** G **esac** y **do** G **od** son una generalización de ambas. Para no perder el carácter ortogonal de nuestra construcción haremos que las sentencias **if** E **then** S **el** S **end if** y **while** E **do** S **end do** sean sentencias de control de flujo exclusivamente. Es decir, que el grado con el que se evalúa E no se transmita al índice con el cual evaluar S.

5.1.3. Tipos

Vamos a introducir una representación compacta para los conjuntos borrosos, para ello utilizaremos los números trapezoidales, y como deseamos tratarlos directamente, es decir queremos nombrarlos, guardarlos y que puedan ser el resultado de una operación o de la llamada a una función, deberán formar parte tanto de los valores almacenables, como de los denotables y los expresables. Además queremos dar la posibilidad de crear nuevos tipos, sobre todo para tratar con lo que Zadeh llama *valores lingüísticos* y *variables lingüísticas* [87]. Con ellos se da significado a las frases coloquiales como *poco adecuado*, *adecuado*, *muy adecuado* etc., referidas a algunas característica de un determinado objeto. El objeto, en su forma más simple, vendrá definido a partir de unas caracteriticas observables en determinadas escalas. Cada una de estas escalas podemos dividirla, de forma borrosa, en distintos tramos que etiquetaremos con un valor lingüístico. El universo del discurso para el objeto será el producto cartesiano de las características. Los subconjuntos borrosos de

dicho producto, construidos a partir de operadores lógicos y valores lingüísticos de las caracterícas formaran la base a partir de la cual podremos establecer valores lingüísticos para el objeto.

Para la declaración de estos nuevos tipo haremos uso del *principio de cualificación* de Tennet [75]. Según dicho principio todo dominio sintáctico puede tener un bloque para admitir declaraciones locales. En particular esto lo aplicaremos a la ampliación de los "record". Ya que el cuerpo de un record es una declaración vamos a permitir que ella aparezcan también funciones, lo cual es semánticamente correcto al ser cada record una especie de ámbito en el que cada identificador está ligado a un valor denotable, por tanto este puede una función. Con lo cual obtendremos una especie de *clase*. Además si adjuntamos a esta estructura una lista de pares de identificadores, podremos definir *antínomos*

5.2. Sintaxis abstracta

Notación

P ∈ Programa

K ∈ Bloque

D ∈ Declaraciones

D_c ∈ Definición de constantes

D_t ∈ Definición de tipos

D_v ∈ Declaración de variables

S ∈ Sentencia

E ∈ Expresión

G ∈ Comando guardado

T ∈ Tipo

I ∈ Identificador

Ω ∈ Operadores binarios

ϒ ∈ Operadores unarios

Π ∈ Parámetros

V_l ∈ Valor lingüístico

$G_r \in$ Grd

$T_o \in$ T_op

N $\in$ Numeral

B $\in$ Booleano

Gramática

P ::= K.

K ::= D **begin** S **end**

D ::= **const** D_c^*
| **var** D_v^*
| **type** D_t^*
| **function** I (Π^*) T ; G_r T_o K
| **procedure** I (Π^*) ; G_r T_o K

D_c ::= I = E

D_v ::= I T

D_t ::= I = **tipo_li** Π^* **va_li** $(V_l)^*$ anti $(I\ I)^*$ **end**

Π ::= I T

T ::= **entero** | **booleano** | **real** | **c_fuzzy** | **borroso** | I

V_l ::= I (Π^*) T ; G_r T_o K
| **as** I

S ::= I **:=** E
| I(E^*)
| **if** E **then** S **else** S **end if**
| **while** E **do** S **end do**
| **print**(E^*)
| **case** G **esac**
| **do** G **od**
| K
| **skip**
| **return**
| I_1 **<-** I_2
| S ; S

G ::= E -> S | G □ G

E ::= I | N | B | E Ω E | ϒ E | I(E^*) | I.E

5.3. Álgebras semánticas

Como hemos señalado vamos a describir la semántica denotacional dinámica. En dicha semántica es irrelevante el asignar un símbolo para representar los errores, pues suponemos que los programas que los contengan ha sido rechazados por ilegales. Por ello reservamos $\perp$ para representar la *no terminación* y no se introducirá ningún símbolo para representar errores en todos los dominios e incluso no especificaremos el tratamiento de dichos errores. Esto se hará más adelante.

I Literales

Ampliamos los dados en página 114, de manera que su sintaxis abstracta sea:

(Literal)
L ::= B | N | R | G | C | CC

Booleano
B ::= true | false

(Numeral)
N ::= *sin especificar*

Numeral Real
R ::= *sin especificar*

Grado
Grd ::= *sin especificar*

(Carácter)
C ::= *sin especificar*

(Cadena-Caracteres)
CC ::= *sin especificar*

a fin de poder disponer de reales y "letreros". Para ello se amplían en los siguientes dominios:

a) Dominio **Bool**. Igual que en página 114

b) Dominio **Num**. Igual que en página 114

c) Dominio **Grd**. Igual que en página 114

d) Dominio **Real** = *sin especificar*
Operaciones
- **ceror, unor,** ...: **Real**
- **sumr, difr, prodr. divr**: **(Real** $\otimes$ **Real)** $\circ\!\!\rightarrow$ **Real**

e) Dominio **Car** = *sin especificar*
Operaciones
- **ord**: **Car** $\circ\!\!\rightarrow$ **Num**

· **chr**: **Num** $\circ\!\!\rightarrow$ **Car**

f) Dominio **Cadena** = *sin especificar*
Operaciones

· **str**: **Car*** $\circ\!\!\rightarrow$ **Cadena**

· **chrs**: **Cadena** $\circ\!\!\rightarrow$ **Car***

g) El dominio **V** queda ampliado de la siguiente forma:
Dominio **V** = **Bool** $\oplus$ **Num** $\oplus$ **Real** $\oplus$ **Grd** $\oplus$ **Car** $\oplus$ **Cadena**

Y finalmente la denotaciones para los literales serian:

$\mathcal{L}$: Literal $\rightarrow$ **V**

$\mathcal{B}$: Booleano $\rightarrow$ **Bool**

$\mathcal{N}$: Numeral $\rightarrow$ **Num**

$\mathcal{R}$: Numeral Real $\rightarrow$ **Real**

$\mathcal{G}$: Grado $\rightarrow$ **Grd**

$\mathcal{CAR}$: Carácter $\rightarrow$ **Car**

$\mathcal{CC}$: Cadena-Caracteres $\rightarrow$ **Cadena**

II Identificadores
Igual que en página 114

III Booleanos Borrosos
Igual que en página 114

IV Números Borrosos
Igual que en página 114

V Reales Borrosos
Dominio **RealB = Real** $\times$ **Grd**

VI Números Trapezoidales
Dominio **NumT** = **Real** $\times$ **Real** $\times$ **Real** $\times$ **Real** $\times$ **Grd**
Operaciones

· sumt, dift, multt, divt : **NumT** $\otimes$ **NumT** $\circ\!\!\rightarrow$ **NumT**

VII Conjuntos Borrosos
Dominio **C_fuzzy** = $\mathcal{P}$(**Num** $\times$ **Grd**)
Operaciones

· unión, intersección, complementario: **C_fuzzy** $\otimes$ **C_fuzzy** $\circ\!\!\rightarrow$ **C_fuzzy**

Para la definición de estos dominios ver página 186

VIII Posiciones de memoria

Dominio **Loc**

Operaciones

- primera_locn: **Loc**
  - **primera_locn**= *Parámetro*
- siguiente_locn: **Loc** → **Loc**
  - **siguiente_locn**= *Sin especificar*
- igual_locn, menorque_locn: **Loc** × **Loc** → **T**

Consultar páginas 197 y 201 para definición e implementación de sus operaciones

IX Valores expresables

Deseamos que **NumT** sea de "primera categoría", es decir que pueda pasarse como parámetro a una función o ser devuelto por ella, etc. Por tanto ampliamos **VE**, el dominio del resultado de la evaluación de expresiones, definido en página 114, quedando:

Dominio **VE** =
BoolB ⊕ **NumB** ⊕ **RealB** ⊕ **NumT** ⊗ **C_fuzzy**.

Ver página 199 para su definición.

X Valores Denotables

Enriqucemos nuestro lenguaje permitiendo la existencia de constantes, funciones, tipos, etc. Por tanto, el conjunto de valores que pueden ser "denotados" por identificadores queda de la siguiente forma:

Dominio **VD** = **Const** ⊕ **Loc** ⊕ **Abst** ⊕ **VaL** ⊕ **TiL**
donde
Const = **VE**
y
Abst = **Func** ⊕ **Proc**

Para su definición ver página 199.

XI Funciones

Func = **Param** $\circ\!\!\rightarrow$ **Ind_Fuz** $\circ\!\!\rightarrow$ **T_op** $\circ\!\!\rightarrow$ (**Alm** ⊗ **Sal**) $\circ\!\!\rightarrow$ (**VE** × ((**Alm** ⊗ **Sal**)$_\perp$ ⊕ δ))$^\natural$

XII Procedimientos

$\mathbf{Proc} = \mathbf{Param} \ \circ\!\!\rightarrow \mathbf{Ind\_Fuz} \ \circ\!\!\rightarrow \mathbf{T\_op} \circ\!\!\rightarrow (\mathbf{Alm} \otimes \mathbf{Sal}) \ \circ\!\!\rightarrow ((\mathbf{Alm} \otimes \mathbf{Sal})_\perp \oplus \delta)^\natural$
donde $\mathbf{Param} = \mathbf{VE}$

XIII Variables lingüísticas

$\mathbf{VaL} = \mathbf{Ide} \rightarrow (\mathbf{Loc} \oplus \mathbf{Func} \oplus \mathbf{VaL})$

XIV Tipos lingüísticos

$\mathbf{TiL} = \mathbf{Amb} \ \circ\!\!\rightarrow \mathbf{Alm} \ \circ\!\!\rightarrow (\mathbf{VD} \times \mathbf{Alm})$

La definición de **Func, Proc, TiL, Param** *en página 200.*

XV Ámbitos

Al dotar a nuestro lenguaje de estructura de bloques, necesitamos los ámbitos, para representar asociaciones entre los identificadores y los valores denotados. El elemento $\top \in \mathbf{O}$ se utiliza para indicar la ausencia del valor denotado
Dominio $\mathbf{Amb} = \mathbf{Ide} \rightarrow (\mathbf{VD} \oplus \mathbf{O})$
Operaciones

- vac_amb: **Amb**
  $\mathbf{vac\_amb} = \lambda \mathrm{id}_{\in \mathbf{Ide}}.\ \mathbf{in}_2 \top$
- acc_amb: $\mathbf{Ide} \rightarrow \mathbf{Amb} \rightarrow \mathbf{VD}$
  $\mathbf{acc\_amb} = \lambda \mathrm{i}_{\in \mathbf{Ide}}.\lambda \mathrm{a}_{\in \mathbf{Amb}}.[\ \mathbf{id}_{\mathbf{VD}},\ \perp\ ](\mathrm{a}(\mathrm{i}))$
- cre_amb: $\mathbf{Ide} \rightarrow \mathbf{VD} \rightarrow \mathbf{Amb}$
  $\mathbf{cre\_amb} = \lambda \mathrm{i}_{\in \mathbf{Ide}}.\lambda \mathrm{v}_{\in \mathbf{VD}}.\lambda \mathrm{i}'_{\in \mathbf{Ide}}.$ if $\mathrm{i} =_{\mathbf{Ide}} \mathrm{i}'$ then $\mathbf{in}_1(\mathrm{v})$ else $\mathbf{in}_2(\top)$
- sol_amb: $\mathbf{Amb} \times \mathbf{Amb} \rightarrow \mathbf{Amb}$
  $\mathbf{sol\_amb} = \lambda(\mathrm{a}_{\in \mathbf{Amb}}, \mathrm{a}'_{\in \mathbf{Amb}}).\lambda \mathrm{i}_{\in \mathbf{Ide}}.[\ \mathbf{id}_{\mathbf{VD}},\ \lambda \mathrm{x}_{\in \mathbf{O}}.\ \mathrm{a}'(\mathrm{i})\ ](\mathrm{a}(\mathrm{i}))$
- mod_amb: $\mathbf{Ide} \rightarrow \mathbf{VD} \rightarrow \mathbf{Amb} \rightarrow \mathbf{Amb}$
  $\mathbf{mod\_amb} = \lambda \mathrm{i}_{\in \mathbf{Ide}}.\lambda \mathrm{v}_{\in \mathbf{VD}}.\lambda \mathrm{a}_{\in \mathbf{Amb}}.\ \mathbf{sol\_amb}\ \mathrm{a}\ (\mathbf{cre\_amb}\ \mathrm{i}\ \mathrm{d})$
- une_amb: $\mathbf{Amb} \times \mathbf{Amb} \rightarrow \mathbf{Amb}$
  $\mathbf{une\_amb} = \lambda(\mathrm{a}_{\in \mathbf{Amb}}, \mathrm{a}'_{\in \mathbf{Amb}}).\lambda \mathrm{i}_{\in \mathbf{Ide}}.$
  $[\ \lambda \mathrm{v}_{\in \mathbf{VD}}.[\ \lambda \mathrm{x}_{\in \mathbf{O}}.\mathrm{v},\ \perp\ ],\ \lambda \mathrm{x}_{\in \mathbf{O}}.\ \mathbf{id}_{\mathbf{VD} \oplus \mathbf{O}}\ ]\ (\mathrm{a}(\mathrm{i}))(\mathrm{a}'(\mathrm{i}))$

Consultar páginas 200 y 201 para definición e implementación de sus operaciones

XVI Valores almacenables

Para representar el conjunto de valores que pueden ser almacenado en una única localización, para ello se utiliza el

Dominio $\mathbf{VA} = \mathbf{VE}$

XVII ALMACÉN: MEMORIA BASADA EN UNA PILA

Los "almacenes" se utilizan para representar asociaciones entre las variables y sus valores. Las variables son representadas por "localizaciones" en los almacenes y la única propiedad relevante que poseen es la de ser distingibles una de otra. Por tanto el identificador de una variables está ligado a una localización, que a su vez dá acceso al valor almacenado en la variable. Ello hace que a veces se introduzcan los dominios **LV** de todas las variables y **RV** el dominio de valores asignables.

Generalmente a la hora de administrar los almacenes sólo se necesita saber si una determinada localizacion está reservada o no. Lo cual hace que por ejemplo la función **asig_locn** se deje sin especificar y todo el modelo se simplifique. Nosotros imponemos una determinada forma de manejar los almacenes a fin de facilitar el prototipo que presentaremos más adelante.

Dominio $\mathbf{Alm} = \mathbf{Loc} \to \mathbf{(VA} \oplus \mathbf{O)} \times \mathbf{Loc}$

Operaciones

- vac_alm: **Alm**

  $\mathbf{vac\_alm} = (\lambda \mathrm{l}_{\in \mathbf{Loc}}.\mathbf{in}_2(\top)$,first_locn)

- acc_alm: $\mathbf{Loc} \to \mathbf{Alm} \to \mathbf{VA}$

  $\mathbf{acc\_alm} = \lambda \mathrm{l}_{\in \mathbf{Loc}}.\lambda(\mathrm{map}_{\in \mathbf{Loc} \to \mathbf{(VA} \oplus \mathbf{O)}}, \mathrm{l}'_{\in \mathbf{Loc}})$.
  if l menorque_locn l′ then (map(l)) else $\mathbf{in}_2(\top)$

- mod_alm: $\mathbf{Loc} \to \mathbf{VA} \to \mathbf{Alm} \to \mathbf{Alm}$

  $\mathbf{mod\_alm} = \lambda \mathrm{l}_{\in \mathbf{Loc}}.\lambda \mathrm{v}_{\in \mathbf{VA}}.\lambda(\mathrm{map}_{\in \mathbf{Loc} \to \mathbf{(VA} \oplus \mathbf{O)}}, \mathrm{l}'_{\in \mathbf{Loc}})$.
  if l menorque_locn l′ then
  $(\lambda \mathrm{l}''$. if $\mathrm{l} =_{\mathbf{Loc}} \mathrm{l}''$ then v else $((\mathrm{map}, \mathrm{l}')(\mathrm{l}'')), \mathrm{l}')$
  else $(\lambda \mathrm{l}_{\in \mathbf{Loc}}.\mathbf{in}_2(\top), \mathrm{l}')$

- mar_loc: $\mathbf{Alm} \to \mathbf{Loc}$

  $\mathbf{mar\_loc} = \lambda(\mathrm{map}_{\in \mathbf{Loc} \to \mathbf{(VA} \oplus \mathbf{O)}}, \mathrm{l}_{\in \mathbf{Loc}}).\mathrm{l}$

- asi_loc: $\mathbf{Alm} \to \mathbf{Loc} \times \mathbf{Alm}$

  $\mathbf{asi\_loc} = \lambda(\mathrm{map}_{\in \mathbf{Loc} \to \mathbf{(VA} \oplus \mathbf{O)}}, \mathrm{l} \in \mathbf{Loc}).(\mathrm{l}, (\mathrm{map}, \mathrm{siguiente\_locn}(\mathrm{l})))$

- des_loc: $\mathbf{Loc} \to \mathbf{Alm} \to \mathbf{Alm}$

  $\mathbf{des\_loc} = \lambda \mathrm{l}_{\in \mathbf{Loc}}.\lambda(\mathrm{map}_{\in \mathbf{Loc} \to \mathbf{(VA} \oplus \mathbf{O)}}, \mathrm{l}'_{\in \mathbf{Loc}}).(\mathrm{map}, \mathrm{l})$

Consultar páginas 200 y 203 para definición e implementación de sus operaciones

XVIII Salida

Dominio **Sal** = (**VA** $\oplus$ **String**)*
Operaciones

- vac_sal: **Sal**
  vac_sal = $\emptyset$
- esc_val: ((**Const** $\oplus$ **String**) $\times$ **Sal**) $\rightarrow$ **Sal**
  esc_val = $\lambda(v_{\in \mathbf{Const} \oplus \mathbf{String}}, s_{\in \mathbf{Sal}})$. s::v

Consultar páginas 197 y 204 para definición e implementación de sus operaciones

5.4. Funciones de valuación

5.4.1. Programa y bloques

En los leguajes clásicos los programas para ejecutarse necesitan de un único parámetro, la primera dirección de memoria que pueden utilizar. Nuestro lenguaje además necesita que se le pasen como parámetros, lo que globalmente hemos llamado *estrategia*, que estaría compuesta de:

1. La t-norma o conorma elegida para ser utilizada en lugar de la operación s del lambda-cálculo_b
2. El orden que se debe considerar en **Grd**.
3. El *neutro*$\in$**Gdr** de s.
4. El valor inicial en el que hacer las evaluaciones.

Se supone que los puntos 2 y 3 son coherentes con el punto 1 y que el valor inicial es el neutro de la norma elegida. Por ello proponemos introducir los siguientes parámetros:

1. **Ind_Fuz**. Se utiliza en el momento de guardar cualquier valor en el almacén. Se alterara como consecuencia de la ejecuación de un comando guardado, o de la ejecución de una función o procedimiento, esta alteración sólo afectará al cuerpo de la función, procedimiento o comando guardado.
2. **T_op**. Se utiliza en el momento de hacer las evaluación de expresiones. Constará de los operadores triangulares necesarios para llevar a cabo la unión, intersección y complementación de conjuntos borrosos.

Consultar página 188 para su implementación

$\mathcal{P}$: PROGRAMA $\rightarrow$ **Amb** $\rightarrow$ **Ind_Fuz** $\rightarrow$ **T_op** $\rightarrow$ $((\mathbf{Alm} \otimes \mathbf{Sal})_\perp \oplus \delta)^\natural$

Consultar página 240 para su implementación

$\mathcal{P}[[\text{K.}]] = \lambda \text{e}_{\in \mathbf{Amb}}.\lambda \text{g}_{\in \mathbf{Ind\_Fuz}}.\mathcal{K}[[\text{K}]]$

$\mathcal{K}$: BLOQUE $\rightarrow$ **Amb** $\rightarrow$ **Ind_Fuz** $\rightarrow$ **T_op** $\rightarrow$ (**Alm** $\otimes$ **Sal**) $\rightarrow$ $((\mathbf{Alm} \otimes \mathbf{Sal})_\perp \oplus \delta)^\natural$

$\mathcal{K}[[\text{ D } \mathbf{inicio} \text{ S } \mathbf{fin}]] = \lambda \text{e}_{\in \mathbf{Amb}}.\lambda \text{i}_{\in \mathbf{Ind\_Fuz}}.\lambda \text{t}_{\in \mathbf{T\_op}}.\lambda \text{a}_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.\mathbf{strict}\lambda \text{l}_{\in \mathbf{Loc}}.$
$(\mathbf{strict}\lambda(\text{e}_{1 \in \mathbf{Amb}}, \text{s}_{1 \in \mathbf{Alm}}).(\mathbf{strict}\lambda \text{a}_{2 \in \mathbf{Alm} \otimes \mathbf{Sal}}.$
$\mathbf{smash}((\mathbf{des\_loc}\ \text{l})^\natural\ \mathbf{on}_1\text{a}_2,\ \mathbf{on}_2\text{a}_2))\ \mathcal{S}[[\text{S}]]\ \text{e}_1\ \text{i}\ \text{t}\ \mathbf{smash}(\text{s}_1, \mathbf{on}_2\text{a}))$
$(\mathbf{on}_1(\mathcal{D}[[\text{D}]]\text{e}\ \mathbf{on}_1\text{a})\ \mathbf{on}_2(\mathcal{D}[[\text{D}]]\text{e}\ \mathbf{on}_1\text{a}))\ \mathbf{mar\_loc}\ \mathbf{on}_1\text{a}$

Consultar página 232 para su implementación

5.4.2. Declaraciones

En este apartado definimos las funciones de evaluación clásicas para las declaraciones en cualquier lenguaje imperativo.

$\mathcal{D}$: DECLARACIONES $\rightarrow$ **Amb** $\rightarrow$ **Alm** $\rightarrow$ (**Amb** $\times$ **Alm**)

$\mathcal{D}[[\ \mathbf{const}\ \text{I} = \text{E}\ ]] = \lambda \text{e}_{\in \mathbf{Amb}}.\lambda \text{a}_{\in \mathbf{Alm}}.$
$\mathbf{strict}\lambda(\text{d}_{\in \mathbf{VE}}, \text{a}_{1 \in \mathbf{Alm}}).((\mathbf{mod\_amb}\ \text{I}\ \mathbf{in}_1^{\mathbf{PV}}(\text{d})\ \text{e}), \text{a}_1)\ \mathcal{E}[[\ \text{E}\ ]]\ \text{e}\ \text{a}$

Consultar página 227 para su implementación

$\mathcal{D}[[\ \mathbf{var}\ \text{I} : \text{T}\ ]] = \lambda \text{e}_{\in \mathbf{Amb}}.\lambda \text{a}_{\in \mathbf{Alm}}.$
$\mathbf{strict}\lambda(\text{d}_{\in \mathbf{VD}}, \text{a}_{1 \in \mathbf{Alm}}).((\mathbf{mod\_amb}\ \text{I}\ \text{d}\ \text{e}),\ \text{a}_1)\ \mathcal{T}[[\text{T}]]\ \text{e}\ \text{a}$

Consultar página 228 para su implementación

$\mathcal{D}[[\ \mathbf{procedure}\ \text{I}\ (\Pi^*);\ \text{K g}\ ]] = \lambda \text{e}_{\in \mathbf{Amb}}.\lambda \text{a}_{\in \mathbf{Alm}}.$
$((\mathbf{mod\_amb}\ \text{I}\ \mathbf{in}_3^{\mathbf{VD}}(\mathbf{in}_2^{\mathbf{Abst}}(\lambda \text{d}^*_{\in \mathbf{VE}^*}.\lambda \text{c}_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.(\lambda(\text{e}_{1 \in \mathbf{Amb}}, \text{c}_{1 \in \mathbf{Alm} \otimes \mathbf{Sal}}).$
$(\lambda \text{a}_{2 \in \mathbf{Alm} \otimes \mathbf{Sal}}.\lambda.\text{i}_{\in \mathbf{Ind\_Fuz}}\mathcal{K}[[\text{K}]]\ \text{e}_1\ (\text{i}_2(\text{i}_1, \text{g}), \text{i}_2)\ \text{a}_2)$
$(\mathcal{R}^*[[\ \text{I}^*\ ]]\ \text{d}^*)\ \text{e}_1\ \text{c}_1)\mathcal{Q}^*\ [[\ \Pi^*\ ]]\ \text{e}\ \text{c}))\ \text{e}),\ \text{a})$

Consultar página 229 para su implementación

donde

$\mathcal{Q}^*$:PARÁMETROS $\rightarrow$ **Amb** $\rightarrow$ **Alm** $\rightarrow$ (**Amb** $\times$ **Alm**)

$\mathcal{Q}^*\ [[\ \Pi^*\ ]] = \lambda \text{e}_{\in \mathbf{Amb}}.\lambda \text{a}_{\in \mathbf{Alm}}.$
$\lambda(\text{e}_{1 \in \mathbf{Amb}}, \text{a}_{1 \in \mathbf{Alm}})\ \ldots\ \lambda(\text{e}_{n \in \mathbf{Amb}}, \text{a}_{n \in \mathbf{Alm}}).$
$(\ \ldots\ ((\mathcal{D}[[\ \text{I}_n\text{: T}_n\ ]])\mathcal{D}[[\ \text{I}_{n-1}\text{: T}_{n-1}\ ]])\ \ldots\ \mathcal{D}[[\ \text{I}_1\text{: T}_1\ ]])$

y

$\mathcal{R}^*$:PARÁMETROS × VALORES EXPRESABLES → **Amb** → **Alm** → **Alm**

$\mathcal{R}^*$ [[I^*]] d^* = $\lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.$
$\lambda a_{1 \in \mathbf{Alm}}. \ldots \lambda a_{n \in \mathbf{Alm}}.(\ldots ((\mathcal{R}[[I_n]]\ d_n\ e\ a_n)$
$\mathcal{R}[[I_{n-2}\ d_{n-2}\ e\ a_{n-2}]]) \ldots \mathcal{R}[[I_1\ d_1\ e\ a_1]])$

donde

$\mathcal{R}$:PARÁMETRO × VALOR EXPRESABLE → **Amb** → **Alm** → **Alm**

$\mathcal{R}$ [[I]] d = $\lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.$
$[\top,[\lambda l_{\in \mathbf{Loc}}$ **if** $\mathcal{T}'(l) = \mathcal{T}'(d)$ **then mod_alm** l d a **else** $\top],\top,\top,\top]$
(**acc_amb** I e)

Consultar página 206 para su implementación

$\mathcal{D}$[[**function** I (Π^*): T; K g]] = $\lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.$
((**mod_amb** I $\mathbf{in}_3^{\mathbf{VD}}(\mathbf{in}_1^{\mathbf{Abst}}(\lambda d^*{}_{\in \mathbf{VE}}.\lambda c_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.(\lambda(e_{1 \in \mathbf{Amb}}, c_{1 \in \mathbf{Alm} \otimes \mathbf{Sal}}).$
$(\lambda a_{2 \in \mathbf{Alm} \otimes \mathbf{Sal}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\mathcal{E}$ [[_I]] e_1 ($\mathcal{K}$[[K]] e_1 ($i_2(i_1,g),i_2$) a_2))
($\mathcal{R}^*$[[I^*]] d^*) e_1 c_1)$\mathcal{Q}^*$ [[Π^*::(_I:T)]] e c)) e), a)

Consultar página 230 para su implementación

$\mathcal{D}$[[**type** I = **tipo_li** D]] = $\lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.$
((**mod_amb** I $\mathbf{in}_3^{\mathbf{VD}}(\lambda a_{1 \in \mathbf{Alm}}.(\mathcal{T}$[[**til** D]] e a_1)) e), a)

Consultar página 232 para su implementación

$\mathcal{D}$[[D_1; D_2]] = $\lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.$ $\mathcal{D}$[[D_2]] $\mathbf{on}_1(\mathcal{D}$[[$D_1$]] e a) $\mathbf{on}_2(\mathcal{D}$[[$D_1$]] e a)

Consultar página 229 para su implementación

5.4.3. Tipos

$\mathcal{T}$: TIPOS → **Amb** → **Alm** → (**VD** × **Alm**)

$\mathcal{T}$[[**booleano**]] = $\lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.$
strict$\lambda(l_{\in \mathbf{Loc}}, a_{1 \in \mathbf{Alm}}).(\mathbf{in}_2^{\mathbf{VD}}(\mathbf{in}_1^{\mathbf{Loc}}(l)),\ a_1)$ **asi_loc** a

$\mathcal{T}$[[**entero**]] = $\lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.$
strict$\lambda(l_{\in \mathbf{Loc}}, a_{1 \in \mathbf{Alm}}).(\mathbf{in}_2^{\mathbf{VD}}(\mathbf{in}_2^{\mathbf{Loc}}(l)),\ a_1)$ **asi_loc** a

$\mathcal{T}$[[**real**]] = $\lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.$
strict$\lambda(l_{\in \mathbf{Loc}}, a_{1 \in \mathbf{Alm}}).(\mathbf{in}_2^{\mathbf{VD}}(\mathbf{in}_3^{\mathbf{Loc}}(l)),\ a_1)$ **asi_loc** a

$\mathcal{T}$[[**borroso**]] = $\lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.$
strict$\lambda(l_{\in \mathbf{Loc}}, a_{1 \in \mathbf{Alm}}).(\mathbf{in}_2^{\mathbf{VD}}(\mathbf{in}_4^{\mathbf{Loc}}(l)),\ a_1)$ **asi_loc** a

$\mathcal{T}$[[**c_fuzzy**]] = $\lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.$
strict$\lambda(l_{\in \mathbf{Loc}}, a_{1 \in \mathbf{Alm}}).(\mathbf{in}_2^{\mathbf{VD}}(\mathbf{in}_5^{\mathbf{Loc}}(l)),\ a_1)$ **asi_loc** a

$\mathcal{T}$[[**TiL** D]] = $\lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.$
strict$\lambda(e_{1 \in \mathbf{Amb}}, a_{1 \in \mathbf{Alm}}).(\mathbf{in}_4^{\mathbf{VD}}(e_1),\ a_1)\ \mathcal{D}$[[D]] e a

$\mathcal{T}$[[I]] = $\lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.$
$[\top, \top, \top, \top, \lambda f_{\in \mathbf{Alm} \to (\mathbf{VD} \times \mathbf{Amb})}.\ f\ a]$ **acc_amb** I e

$\mathcal{T}'$: TIPOS → **Amb** → (**VE** ⊕ **O**)

$\mathcal{T}'$[[I]] = $\lambda e_{\in \mathbf{Amb}}.[\top, \lambda l_{\in \mathbf{Loc}}.\mathbf{id}_{\mathbf{VE}},\ \top, \top, \top]$ **acc_amb** I e

Consultar página 227 para su implementación

5.4.4. Sentencias

$\mathcal{S}$: SENTENCIAS → **Amb** → **Ind_Fuz** → **T_op** → (**Alm** ⊗ **Sal**) →
((**Alm** ⊗ **Sal**)$_\perp$ ⊕ δ)$^\natural$

$\mathcal{S}$[[I **:=** E]] = $\lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\lambda t_{\in \mathbf{T\_op}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.$
strict$\lambda(v_{\in \mathbf{VE}}, a_{1 \in \mathbf{Alm} \otimes \mathbf{Sal}}).\lambda l_{\in \mathbf{Loc}}.$ ((**mod_alm** l i_2(v,i_1) **on**$_1$ a_1),**on**$_2$ a_1)$^\natural$
(**acc_amb** [[I]] e)($\mathcal{E}$[[E]] e t a)

Consultar página 233 para su implementación

$\mathcal{S}$[[**if** E **then** S **end if**]] = $\lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\lambda t_{\in \mathbf{T\_op}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.$
(**strict**$\lambda(t_{1 \in \mathbf{Bool}}, i_{t \in \mathbf{Grd}})$. **if** t_1 **then** $\mathcal{S}$[[S]] e i t a
else {| a |})($\mathcal{B}$[[E]] e t a)

Consultar página 235 para su implementación

$\mathcal{S}$[[**while** E **do** S **end do**]] = $\lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\lambda t_{\in \mathbf{T\_op=(min/max)}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.$
fix$(\lambda f_{\in \mathbf{Alm} \to (\mathbf{Alm} \otimes \mathbf{Sal})^\natural}.\lambda a_{\in \mathbf{Alm}}.\lambda(t_{1 \in \mathbf{Bool}}, i_{1 \in \mathbf{Grd}})$.
if t_1 **then ext**(f)($\mathcal{S}$[[S]] e i t a)
else {| a |})($\mathcal{B}$[[E]] e t a)

Consultar página 236 para su implementación

$\mathcal{S}$[[I(E*)]] = **strict**$\lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind_Fuz}}.\lambda t_{\in \mathbf{T_op}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.$
$[\top, \top, [\top, \lambda p_{\in \mathbf{Proc}}.(p\ (\mathcal{E}$[[E*]] e t a) i t a)$], \top, \top]$(**acc_amb**[[I]] e a)

Consultar página 234 para su implementación

$\mathcal{S}$[[**case** G **esac**]] = $\lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\lambda t_{\in \mathbf{T\_op}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.$
(**strict**$\lambda(t_{1 \in \mathbf{Bool}}, i_{1 \in \mathbf{Grd}})$. **if** t_1 **then** $\mathcal{G}$[[G]] e i t a
else {| δ |})) $\mathcal{V}$[[G]] e a

Consultar página 235 para su implementación

$$\mathcal{S}[\![\ \textbf{do}\ \mathrm{G}\ \textbf{od}\ ]\!] = \lambda e_{\in \mathbf{Amb}}.\lambda i_{1 \in \mathbf{Ind\_Fuz}}.\lambda t_{\in \mathbf{T\_op=(max/min)}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.$$
$$\textbf{fix}(\textbf{strict}\lambda(t_{1 \in \mathbf{Bool}}, i_{1 \in \mathbf{Grd}}).\lambda f_{\in \mathbf{Alm} \to (\mathbf{Alm} \otimes \mathbf{Sal})^{\natural}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.$$
$$\textbf{if}\ t_1\ \textbf{then}\ \textbf{ext}(f)(\mathcal{G}[\![\mathrm{G}]\!]\ e\ i\ t\ a)$$
$$\textbf{else}\ \{\!|\ a\ |\!\})(\mathcal{V}[\![\mathrm{G}]\!]\ e\ a)$$

Consultar página 236 para su implementación

$$\mathcal{S}[\![\ \mathrm{K}\ ]\!] = \lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\lambda t_{\in \mathbf{T\_op}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.\mathcal{K}[\![\ \mathrm{K}\ ]\!]\ e\ i\ t\ a$$

Consultar página 237 para su implementación

$$\mathcal{S}[\![\ \mathrm{S}_1; \mathrm{S}_2\ ]\!] = \lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\lambda t_{\in \mathbf{T\_op}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.$$
$$\textbf{ext}(\mathcal{S}[\![\ \mathrm{S}_2\ ]\!]\ e\ i\ t)\ \mathcal{S}[\![\ \mathrm{S}_1\ ]\!]\ e\ i\ t\ a$$

Consultar página 238 para su implementación

$$\mathcal{S}[\![\ \textbf{skip}\ ]\!] = \lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\lambda t_{\in \mathbf{T\_op}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.\{\!|\ a\ |\!\}$$

Consultar página 236 para su implementación

$$\mathcal{S}[\![\ \textbf{return}\ ]\!] = \lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\lambda t_{\in \mathbf{T\_op}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.\{\!|\ a\ |\!\}$$

Consultar página 237 para su implementación

$$\mathcal{S}[\![\ \mathrm{I}_1 \textbf{<-}\ \mathrm{I}_2\ ]\!] = \lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\lambda t_{\in \mathbf{T\_op}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.\{\!|\ a\ |\!\}$$

Consultar página 237 para su implementación

$$\mathcal{S}[\![\ \textbf{print}\ \mathrm{E}\ ]\!] = \lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\lambda t_{\in \mathbf{T\_op}}.\lambda(a_{\in \mathbf{Alm}}, s_{\in \mathbf{Sal}}).$$
$$\{\!|(a, \textbf{esc\_valor}\ \mathcal{E}([\![\mathrm{E}]\!]\ e\ t\ a)\ s)\ |\!\}$$

Consultar página 236 para su implementación

5.4.5. Comandos guardados

$\mathcal{V}$: COMANDO GUARDADO $\to$ **Amb** $\to$ **Alm** $\to$ **BoolB**

$$\mathcal{V}[\![\ \mathrm{G}_1 \square \mathrm{G}_2\ ]\!] = \lambda e_{\in \mathbf{Amb}} \lambda a_{\in \mathbf{Alm}}.(\mathcal{V}[\![\mathrm{G}_1]\!]\ e\ a)\ \textit{or}\ (\mathcal{V}[\![\mathrm{G}_2]\!]\ e\ a)$$
$$\mathcal{V}[\![\ \mathrm{E} \to \mathrm{S}\ ]\!] = \lambda e_{\in \mathbf{Amb}}.\lambda a_{\in \mathbf{Alm}}.\ \mathcal{B}[\![\mathrm{E}]\!]\ e\ a$$

Consultar página 238 para su implementación

$\mathcal{G}$: COMANDO GUARDADO $\to$ **Amb** $\to$ **Ind_Fuz** $\to$ **T_op** $\to$ (**Alm** $\otimes$ **Sal**) $\to$ $((\mathbf{Alm} \otimes \mathbf{Sal})_\perp \oplus \delta)^{\natural}$

$$\mathcal{G}[\![\ \mathrm{G}_1 \square \mathrm{G}_2\ ]\!] = \lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\lambda t_{\in \mathbf{T\_op}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.$$
$$(\mathcal{G}[\![\mathrm{G}_1]\!]\ e\ i\ t\ a) \uplus\ (\mathcal{G}[\![\mathrm{G}_2]\!]\ e\ i\ t\ a)$$
$$\mathcal{G}[\![\ \mathrm{E} \to \mathrm{S}\ ]\!] = \lambda e_{\in \mathbf{Amb}}.\lambda i_{\in \mathbf{Ind\_Fuz}}.\lambda.t_{\in \mathbf{T\_op}}.\lambda a_{\in \mathbf{Alm} \otimes \mathbf{Sal}}.$$
$$(\lambda(t_{1 \in \mathbf{Bool}}, i_{t \in \mathbf{Grd}}).\ \textbf{if}\ t_1\ \textbf{then}\ \mathcal{S}[\![\mathrm{S}]\!]\ e\ t(i, i_t)\ t\ a$$
$$\textbf{else}\ \{\!|\ a\ |\!\})(\mathcal{B}[\![\mathrm{E}]\!]\ e\ t\ a)$$

Consultar página 238 para su implementación

5.4.6. Expresiones

$\mathcal{E}$: EXPRESIÓN $\rightarrow$ **Amb** $\rightarrow$ **T_op** $\rightarrow$ (**Alm** $\otimes$ **Sal**) $\rightarrow$
(**VE** $\times$ ((**Alm** $\otimes$ **Sal**)$_\perp$ $\oplus$ δ))$^\natural$

$\mathcal{E}$[[L]] = λe$_{\in\mathbf{Amb}}$.λt$_{\in\mathbf{T_op}}$.λa$_{\in\mathbf{Alm}\otimes\mathbf{Sal}}$.{| ($\mathcal{L}$[[L]],a) |}

Consultar página 221 para su implementación

$\mathcal{E}$[[Υ E]] = λe$_{\in\mathbf{Amb}}$.λ.t$_{\in\mathbf{T_op}}$.λa$_{\in\mathbf{Alm}\otimes\mathbf{Sal}}$.
{| ($\mathcal{U}$[[Υ]] t ($\mathcal{E}$[[E]] e t a),a) |}

$\mathcal{E}$[[E$_1$ Ω E$_2$]] = λe$_{\in\mathbf{Amb}}$.λt$_{\in\mathbf{T_op}}$.λa$_{\in\mathbf{Alm}\otimes\mathbf{Sal}}$.
{|($\mathcal{B}I$[[Ω]] t ($\mathcal{E}$[[E$_1$]] e t a) ($\mathcal{E}$[[E$_2$]] e t a),a) |}

Consultar página 214 para su implementación

$\mathcal{E}$[[I]] = λe$_{\in\mathbf{Amb}}$.λt$_{\in\mathbf{T_op}}$.λa$_{\in\mathbf{Alm}\otimes\mathbf{Sal}}$.
{|([[(**id**$_{\mathbf{VE}}$,a),((**strict**λl.$_{\in\mathbf{Loc}}$.**acc_alm** l a),a),$\top$,$\top$,$\top$] (**acc_amb** I e a),a)|}

Consultar página 222 para su implementación

$\mathcal{E}$[[I.E]] = λe$_{\in\mathbf{Amb}}$.λt$_{\in\mathbf{T_op}}$.λa$_{\in\mathbf{Alm}\otimes\mathbf{Sal}}$.
{| ([$\top$,$\top$,$\top$,$\mathcal{E}$[[E]] e t a,$\top$] (**acc_amb** I e a),a) |}

Consultar página 222 para su implementación

$\mathcal{E}$[[I(E*)]] = λe$_{\in\mathbf{Amb}}$.λt$_{\in\mathbf{T_op}}$.λa$_{\in\mathbf{Alm}\otimes\mathbf{Sal}}$.
[$\top$,$\top$,[λp$_{\in\mathbf{Func}}$.(p($\mathcal{E}$[[E*]] e t a) a),$\top$],$\top$,$\top$](**acc_amb**[[I]] e a)

Consultar página 224 para su implementación

Operadores Unarios

$\mathcal{U}$: OPERADORES UNARIOS $\rightarrow$ **T_op** $\rightarrow$ **VE** $\circ\!\!\rightarrow$ **VE**

$\mathcal{U}$[[**not**]] = λt$_{\in\mathbf{T_op}}$.λ(b$_{\in\mathbf{Bool}}$,g$_{\in\mathbf{Grd}}$).**if** b **then** (**false**,g$'$) **else** (**true**,g$'$)
donde g$'$ es tal que t(g,g$'$) = neutro

Consultar página 220 para su implementación

$\mathcal{U}$[[**complementario**]] = λt$_{\in\mathbf{T_op}}$.λc_f$_{\in\mathbf{C_fuzzy}}$.(**com** c_f,t)

Consultar página 220 para su implementación

$\mathcal{U}$[[**-**]] = λt$_{\in\mathbf{T_op}}$.λ(n$_{\in\mathbf{Num}}$,g$_{\in\mathbf{Grd}}$).(-n,g)

Consultar página 219 para su implementación

Operadores Binarios

$\mathcal{B}I$: OPERADORES BINARIOS $\rightarrow$ **T_op** $\rightarrow$(**VE** $\otimes$ **VE**) $\circ\!\!\rightarrow$ **VE**

$\mathcal{B}I$[[**and**]] = $\lambda t_{\in \mathbf{T\_op}}.\lambda(t_{1\in \mathbf{Bool}}, g_{1\in \mathbf{Grd}}, t_{2\in \mathbf{Bool}}, g_{2\in \mathbf{Grd}})$.
if t_1 **then** $(t_2, t(g_1, g_2))$ **else** (**false**,$t(g_1, g_2)$)

$\mathcal{B}I$[[**or**]] = $\lambda.t_{\in \mathbf{T\_op}}.\lambda(t_{1\in \mathbf{Bool}}, g_{1\in \mathbf{Grd}}, t_{2\in \mathbf{Bool}}, g_{2\in \mathbf{Grd}})$.
if t_1 **then** (**true**,$t'(g_1, g_2)$) **else** $(t_2, t'(g_1, g_2))$

Consultar página 210 para su implementación

$\mathcal{B}I$[[**op_arit**]] = $\lambda.t_{\in \mathbf{T\_op}}.\lambda(n_{1\in \mathbf{Num}}, g_{1\in \mathbf{Grd}}, n_{2\in \mathbf{Num}}, g_{2\in \mathbf{Grd}})$.
(**op_arit**(n_1, n_2),$t(g_1, g_2)$)
donde **op_arit** puede ser: **sum, dif, prod, div**

Consultar página 210 para su implementación

$\mathcal{B}I$[[**op_rel**]] = $\lambda t_{\in \mathbf{T\_op}}.\lambda(n_{1\in \mathbf{Num}}, g_{1\in \mathbf{Grd}}, n_{2\in \mathbf{Num}}, g_{2\in \mathbf{Grd}})$.
(**op_rel**(n_1, n_2),$t(g_1, g_2)$)
donde **op_rel** puede ser: **<, >, <=, >=, =, !=**

Consultar página 212 para su implementación

$\mathcal{B}I$[[$\mathbf{op\_arit}_r$]] = $\lambda t_{\in \mathbf{T\_op}}.\lambda(n_{1\in \mathbf{Real}}, g_{1\in \mathbf{Grd}}, n_{2\in \mathbf{Real}}, g_{2\in \mathbf{Grd}})$.
($\mathbf{op\_arit}_r(n_1, n_2)$,$t(g_1, g_2)$)
donde $\mathbf{op\_arit}_r$ puede ser: **sumr, difr, prodr, divr**

Consultar página 212 para su implementación

$\mathcal{B}I$[[$\mathbf{op\_rel}_r$]] = $\lambda t_{\in \mathbf{T\_op}}.\lambda(r_{1\in \mathbf{Real}}, g_{1\in \mathbf{Grd}}, r_{2\in \mathbf{Real}}, g_{2\in \mathbf{Grd}})$.
($\mathbf{op\_rel}_r(r_1, r_2)$,$t(g_1, g_2)$)
donde **op_rel** puede ser: **<, >, <=, >=, =, !=**

Consultar página 212 para su implementación

$\mathcal{B}I$[[$\mathbf{op\_arit}_t$]] = $\lambda t_{\in \mathbf{T\_op}}.\lambda(n_{1\in \mathbf{Num}}, g_{1\in \mathbf{Grd}}, n_{2\in \mathbf{NumT}}, g_{2\in \mathbf{Grd}})$.
($\mathbf{op\_arit}_t(n_1, n_2)$,$t(g_1, g_2)$)
donde $\mathbf{op\_arit}_r$ puede ser: **sumt, dift, prodt, divt**

Consultar página 213 para su implementación

$\mathcal{B}I$[[$\mathbf{op\_arit}_{tr}$]] = $\lambda t_{\in \mathbf{T\_op}}.\lambda(n_{\in \mathbf{Num}}, g_{1\in \mathbf{Grd}}, r_{\in \mathbf{Real}}, g_{2\in \mathbf{Grd}})$.
($\mathbf{op\_arit}_{tr}(n, r)$,$t(g_1, g_2)$)
donde $\mathbf{op\_arit}_{tr}$ puede ser: **sumtr, diftr, prodtr, divtr**

Consultar página 213 para su implementación

$\mathcal{B}I$[[**op_con**]] = $\lambda t_{\in \mathbf{T\_op}}.\lambda c\_f_{1\in \mathbf{C\_fuzzy}}.\lambda c\_f_{2\in \mathbf{C\_fuzzy}}$.
(**op_con**(c_f_1, c_f_2, t))
donde **op_con** puede ser: **union,interseccion**

Consultar página 212 para su implementación

$\mathcal{B}I$[[**in**]] = $\lambda t_{\in \mathbf{T\_op}}.\lambda n_{\in \mathbf{Num}}.\lambda c\_f_{\in \mathbf{C\_fuzzy}}$.
(**in**(n, c_f))

Consultar página 217 para su implementación

5.5. Resumen

En este capítulo hemos ampliado el lenguaje:

Introduciendo nuevos tipos básicos:

- Numeros reales
- Numeros trapezoidales
- Conjuntos borrosos
- Cadenas de caracteres

Permitiendo la definición de abstracciones

- Procedimientos
- Funciones

- Permitiendo la creación de nuevos tipos

Lo que nos permite escribir programas como el siguiente.(En el apéndice D hay otros ejemplos).

```
\       Programa prueba de errores en el paso de parametros.
const y={0.5,6.,7.,0.8}  ;

function entre(integer a,b):c_fuz

var integer n,i;
    c_fuz c, d;
begin
   n := a-5 ;
   i := 1 ;
   c := [ 0,18]{8|0.1} ;
   while i < 5 do
      d:= (n+i)|((0.2*EXTR(i)));
      c:= c U d ;
      i:= i + 1
   end do ;
   i := a ;
   while i <= b do
      d:= i|1.0 ;
```

```
      c:= c U d ;
      i:= i + 1
   end do ;
   n := b+5 ;
   i := 1 ;
   while i < 5 do
      d:= (n-i)|((0.2*EXTR(i)));
      c:= c U d ;
      i:= i + 1
   end do ;
   entre <- c
end;

var
     c_fuz a1,a2,na1,na2 ;
     boolean b1,b2;
     integer i ;
begin
    i := 8 ;
    a1 := entre(11,14);
    a2 := [ 0,18]{6|0.1,7|0.3,8|0.5,9|1.,10|1.,11|0.5,12|0.2};
    na1 := a1 N a2;
    na2 := C a1;
    print("a1=",a1,nl,"a2=",a2,nl,"na1=",na1,nl,"na2=",na2,nl);
    b1 := i in a1;
    b2 := i in na2 ;
    print("i en a1 ",b1,nl,nl,"i en complementario a1 ",b2)
end.
```

produciendo la siguiente salida

```
a1=[0, 18]{7|0.2, 8|0.4, 9|0.6, 10|0.8, 11|1, 12|1, 13|1, 14|1,
     15|0.8, 16|0.6, 17|0.4, 18|0.2}

a2=[0, 18]{6|0.1, 7|0.3, 8|0.5, 9|1, 10|1, 11|0.5, 12|0.2}

na1=[0, 18]{7|0.2, 8|0.4, 9|0.6, 10|0.8, 11|0.5, 12|0.2}

na2=[0, 18]{0|1, 1|1, 2|1, 3|1, 4|1, 5|1, 6|1, 7|0.8, 8|0.6, 9|0.4,
     10|0.2, 15|0.2, 16|0.4, 17|0.6, 18|0.8}

i en a1 (1)CIERTO(0.4)

i en complementario a1 (1)CIERTO(0.6)
```

Conclusiones y Líneas futuras

Al iniciar esta tesis nos pusimos como objetivo elaborar las herramientas para el diseño y la especificación formal de lenguajes de programación que tuvieran en cuenta el paradigma borroso. Debiamos rellenar dos requisitos:

- Permitir que la definición del lenguaje sea completa.
- Permitir que la definición formal del lenguaje se pueda utilizar en un entorno industrial, para servir:
    1. Como un estándar para el lenguaje. Es el documento de referencia para la validación de implementaciones y como una guía para implementadores.
    2. Como entrada para un generador de compiladores

En este trabajo se han aportados varios resultados entre los cuales destacamos:

- La ampliación del lambda cálculo de manera que cada uno de sus términos está acompañado de un elemento de un conjunto $\mathbb{D}$. Hemos investigado las condiciones mínimas de las que debe estar dotado dicho conjunto para que se cumpla la propiedad de Church-Rosser.
- La estructuración de los sistemas de numeración en el lambda cálculo ampliado, definiendo una relación de equivalencia entre sus términos, para permitir la incorporación de definiciones externas, llegando a la conclusión que para ello es suficiente que sobre el conjunto $\mathbb{D}$ se tenga definida una T-norma o una S-conorma.
- El establecimiento de un modelo de cálculo, que contiene como un caso particular al lambda cálculo clásico.
- El diseño de un lenguaje multivaluado, haciendo uso del modelo anterior, suministrando su sintaxis y su semántica denotacional.
- Hemos ampliado el lenguaje anterior para dar cabida a bloques y abstracciones, aumentar las estructuras de control, permitir la definición de tipos. Con ello hemos mostrado que, tanto el grado de borrosidad con el que se trabaja como la norma que se utiliza, pueden tener alcances estático o dinámico. Además hemos añadido un mecanismo sencillo que permite la definición de variables lingüísticas.

- Hemos esbozado una estrategia para permitir la comparación de programas que manejen datos borrosos. En la programación determinista esto es relativamente fácil. En nuestro caso hay que tener en cuenta las posibles opciones.

Una vez establecidas las bases sobre la que diseñar lenguajes de programación que manejen datos borrosos se abre todo un abanico de posibilidades. La semántica denotacional no se limita a los lenguajes imperativos, lenguajes funcionales o lógicos también pueden ser diseñados con su ayuda. En los apéndices se muestra un intérprete de un lenguaje funcional fundado en el modelo de cálculo aquí establecido.

En el futuro sería deseable proseguir la linea aquí establecida tanto desde el punto de vista teórico como del práctico:

Respecto a la elección de la forma de representar $[0, 1]$ hemos sido excesivamente conservadores, una vía muy interesante sería representar $[0, 1]$ por medio de un conjunto numerable.

Investigar la semántica axiomática de lenguajes de progrmación con datos borrosos y establecer, para un lenguaje común, las relaciones que se dan en el caso nítido entre la semántica axiomática y la denotacional.

Utilizar la semántica denotacional para generar una gramática atribuida para facilitar la traducción a algún código intermedio.

El lenguaje diseñado admite multitud de ampliaciones: punteros, matrices, módulos, etc. Sin embargo más interesantes serían:

Desarrollar en profundidad las clases y objetos como soporte de las variables lingüísticas

Introducir el tiempo, pues en general la correspondencia entre valores lingüísticos y clases borrosas no suele ser estática. Dicha correspondencia a menudo muestra un aspecto dinámico.

Introducir saltos y continuaciones, a partir de ellos se podría dar cuenta de las excepciones y corrutinas

Introducir algunos métodos de inferencia

Introducir el tipo conjunto borroso de forma que sus valores sean almacenables, denotables y expresables, conectando la asignación a una variable de este tipo con las sentencias multivaluadas. Estudiar el problema del complementario.

Estudiar la paralelización del lenguaje, en particular al ejecutar los comandos guardados

Finalmente concluir con las posibilidades que se ofrece el disponer de un lenguaje que sea capaz desde su propia fundamentación con la capacidad de manejar la borrosidad bien en las operaciones bien en la defición de las variables. Dicho lenguaje nos permitirá dentro de los proyectos de investigación que actualmente están siendo desarrollados en nuestro Departamento, no sólo evaluar sus posibles ampliaciones en entornos de razonamiento aproximado, sino su uso como herramienta de razonamiento y de prototipado de sistemas de consultas flexibles.

Apéndice A

Ordenaciones en los dominios potencia

A.1. Introducción

Como ya mencionamos para los dominios potencia no hay una solución que sea la mejor. Esto es debido a que las diferencias existentes entre las propiedades de los subconjuntos del dominio potencia y las propiedades del orden parcial de sus componentes se pueden resolver de distintas formas, consultar página 26. Esta dificultad se ve incrementada cuando el dominio sobre el que se trabaja no es plano. Nuestro interés en los dominios no planos proviene de que, como sabemos, un conjunto borroso F queda definido a partir de un referencial Ω y una aplicación, escrita μ_F, de Ω en $[0,1]$. De forma que $\forall\omega \in \Omega\ \mu_F(\omega)$ se interpreta como el grado de pertenencia de ω al conjunto borroso F. Esto permite modelizar categorías vagas del lenguaje natural definidas sobre un soporte objetivo con el que clasificarlo con la ayuda de dichas categorías. $\mu_F(\omega)$ expresará entonces hasta que punto el valor ω es compatible con el concepto F [23]. Por tanto, aún cuando el referencial Ω sea un dominio plano, el dominio formado por el referencial junto con el grado de pertenencia, formarán un dominio no plano.

A.2. Dominios potencia discretos

Los dominios potencia discretos son los construidos a partir de dominios planos

Definición A.1 *Sea A un dominio plano, el dominio potencia relacional discreto de A, representado por $\mathcal{P}_R(A)$, es la colección de todos los subconjunto propios de A (por tanto no se considera $\perp$), parcialmente ordenados de la siguiente forma: para todo $U, V \in \mathcal{P}_R(A)$ $U \subseteq_R V$ si y solo si $(\forall x \in U)(\exists y \in V)$ tal que $x \subseteq_A y$*

El único elemento del dominio plano que podría haber causado problemas, $\perp$, desaparece

Definición A.2 *Sea A un dominio plano, el dominio potencia Smyth discreto de A, representado por $\mathcal{P}_S(A)$, es la colección de todos los subconjuntos finitos no vacíos de elementos propios de A, junto con el mismo A, ordenados parcialmente de la siguiente forma: para todo $U, V \in \mathcal{P}_S(A)$ $U \subseteq_S V$ si y solo si $(\forall y \in V)(\exists x \in U)$ tal que $x \subseteq_A y$*

Las relaciones $\subseteq_S$ y $\subseteq_R$ son inversas. Aquí el menor elemento es A

Definición A.3 *Sea A un dominio plano, el dominio potencia discreto Egli-Milner, representado por $\mathcal{P}_{EM}(A)$ es la colección de todos los subconjuntos no vacíos de A que o bien son finitos o contienen a $\perp$, parcialmente ordenados de la siguiente forma; para todo $U, V \in \mathcal{P}_{EM}(A)$ $U \subseteq_{EM} V$ si y solo si:*

1. $(\forall x \in U)(\exists y \in V)$ *tal que* $x \subseteq_A y$
2. $(\forall y \in V)(\exists x \in U)$ *tal que* $x \subseteq_A y$

Esta construcción solo opera con dominios planos que contengan $\perp$. Todos los conjuntos del dominio potencia son no vacíos. Si consideramos $\mathcal{P}_{EM}(N_\perp)$ tendríamos que $\{1,\perp\} \subseteq_{EM} \{1,2,3\}$ pero $\{1\} \not\subseteq_{EM} \{1,2,3\}$. En este dominio potencia el conjunto vacio es representado por $\{\perp\}$ y como consecuencia se tiene que $\emptyset \cup a$ no es igual a a.

Ya que $\subseteq_S$ es la inversa de $\subseteq_R$, $U \subseteq_S V$ si y solo si $V \subseteq U$. El orden inverso de los subconjuntos sugiere que un conjunto U está mejor definido que V cuando la información de U es más especifica que la información de V. Utilizar el dominio potencia de Smyth puede considerarse como un proceso para determinar que cosas no pueden ser una respuesta. Un cálculo que no termine no suministra información. Por tanto A es el elemento menos definido de $\mathcal{P}_S(A)$. Al igual que en el dominio potencia de Egli-Milner se tiene que $\emptyset \cup a$ no es igual a a. Aquí $\emptyset \cup a = \emptyset$. Este dominio potencia es apropiado para estudiar la corrección total, es decir, los resultados son válidos solo si el programa examinado siempre termina.

A.3. Dominios potencia en general

Vamos a generalizar la construcción de los dominios potencia discretos a fin de manejar dominio que no sean planos. Empezaremos con la generalización del dominio potencia relacional, para ello definimos el dominio potencia relacional de un dominio arbitrario A de forma similar a la versión discreta: los elementos de $\mathcal{P}_R(A)$ deben ser los subconjuntos de elementos propios de A ordenados por: $U \subseteq_R V$ si y solo si $(\forall x \in U)(\exists y \in V)$ tal que $x \subseteq_A y$. Desgraciadamente, esta forma de ordenar da lugar a problemas pues $\subseteq_R$ no es un orden parcial. Por ejemplo dados los elementos propios $a_1, a_2 \in A$ tales que $a_1 \subseteq_A a_2$, se tiene que $\{a_2\} \subseteq_R \{a_1, a_2\}$ y $\{a_1, a_2\} \subseteq_R \{a_2\}$. La razón de esta equivalencia es que la información total contenida en ambos conjuntos es idéntica. Podríamos esperar una solución agrupando juntos

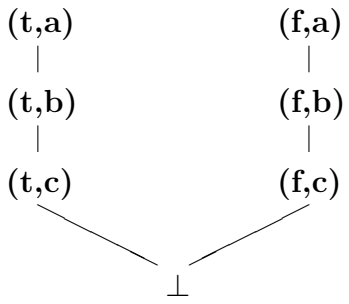
aquellos conjuntos que fueran equivalentes con respecto a la ordenación $\subseteq_R$. Sea pues $\mathcal{P}(A)/\subseteq_R$ el cociente de $\mathcal{P}(A)$ con respecto a $\subseteq_R$, es decir los conjuntos de los elementos propios de A agrupados en colecciones llamadas clases de equivalencia. Dos conjuntos U y V estarían en la misma clase de equivalencia si y solo si $U \subseteq_R V$ y $V \subseteq_R U$. Las clases de equivalencia estarían parcialmente ordenadas por $\subseteq_R$: para las clases de equivalencia $P, Q \in \mathcal{P}(A)/\subseteq_R$, $P \subseteq Q$ si y solo si para todo $U \in P$ y $B \in Q$, $U \subseteq_R V$. Sea $[U]$ el representante de la clase de equivalencia que contiene al conjunto $U \in \mathcal{P}(A)$. Se pueden definir las operaciones $\emptyset$, $\{\!|\cdot|\!\}$ y $\uplus$. La menor de las cotas superiores en el dominio se determinan por la unión de conjuntos: para una cadena $C = \{[X_i] \mid i \in I\}$, $\bigsqcup C = [\bigcup\{X_i \mid i \in I\}]$. Desgraciadamente el dominio cociente también da problemas: la operación $\{\_\}$ no es continua. En efecto, si $c \in A$ es la menor cota superior de la cadena $\{d_i \mid i \in I\}$ entonces $\bigsqcup\{[\{d_i\}] \mid i \in I\} = [\bigcup\{\{d_i\} \mid i \in I\}] = [\{d_i\} \mid i \in I\}]$, pero esta no es la misma clase de equivalencia que $[\{c\}]$. Por ello hemos de recurrir a la construcción del dominio de los ideales sobre el conjunto de los subconjuntos finitos no vacíos de A. Otra opción hubiese sido tratarlos directamente como *sistemas de información* [70], [83]. Nuestro caso es muy particular, pero muy importante para nosotros, nos dará las pautas para temas tan fundamentales como la optimización o la comparación de programas, por ello vamos a estudiarlo con algún detalle.

A.4. El problema

Supongamos que tenemos un referencial plano, finito **T** y que conforme a lo señalado al principio del capítulo 4 aceptamos que $[0, 1]$ esté representado por un conjunto finito lo llamaremos **Bo**. Desde el punto de vista de la teoría de dominios ¿como podríamos ordenar $\mathbf{T}\otimes\mathbf{Bo})_\perp$?. Las tres respuestas posibles las consideramos a continuación. Para visualizarlas vamos a considerar que **T** = {**t**,**f**} y **Bo** = {**a**,**b**,**c**} con la siguiente relación de orden

a
|
b
|
c

Su producto cartesiano puntiagudo, al que llamaremos $\mathbf{TB} = (\mathbf{T}\otimes\mathbf{Bo})_\perp$ será



El conjunto $\mathsf{Idl}(\mathbf{TB}) = \widehat{\mathbf{TB}}$ está formado por los siguientes elementos:

$\widehat{t_a} = \{(\mathbf{t,a}),(\mathbf{t,b}),(\mathbf{t,c}),\bot\}$

$\widehat{t_b} = \{(\mathbf{t,b}),(\mathbf{t,c}),\bot\}$

$\widehat{t_c} = \{(\mathbf{t,c}),\bot\}$

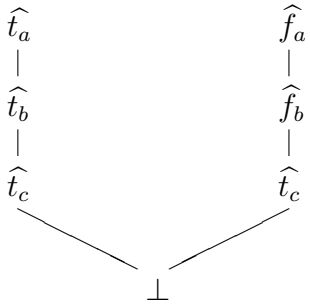
$\widehat{f_a} = \{(\mathbf{f,a}),(\mathbf{f,b}),\mathbf{f,c}),\bot\}$

$\widehat{f_b} = \{(\mathbf{f,b}),(\mathbf{f,c}),\bot\}$

$\widehat{f_c} = \{(\mathbf{f,c}),\bot\}$

$\widehat{\bot} = \{\bot\}$

ordenados de la siguiente forma:



A.5. Dominio Potencia Inferior

Es el conjunto de ideales sobre el preorden $\langle \mathcal{P}^*_f(\widehat{\mathbf{TB}}), \sqsubseteq_1 \rangle$. Para ver como puede ser estos ideales hay que notar que $u \cup \{\bot\} \sqsubseteq_1 u$ y $u \sqsubseteq_1 u \cup \{\bot\}$ para cualquier $u \in \mathcal{P}^*_f(\widehat{\mathbf{TB}})$. O más general si $u_1, u_2 \in \mathcal{P}^*_f(\widehat{\mathbf{TB}})$ son tales que $u_1 \sqsubseteq_1 u_2$ entonces $\{u_2\} \sqsubseteq_1 \{u_1.u_2\}$ y $\{u_1, u_2\} \sqsubseteq_1 \{u_2\}$. Por ello es necesario establecer clases de equivalencia a partir de la relación $\approx_1$ definida por

$A \approx_1 B$ si y solo si $A \sqsubseteq_1 B$ y $B \sqsubseteq_1 A$

A.5.1. Clases

CLASE	Elementos que pertenecen
$[\mathbf{t}_a]_1$	$\{\widehat{t_a}\}$,$\{\widehat{t_a},\widehat{t_b}\}$, $\{\widehat{t_a},\widehat{t_c}\}$, $\{\widehat{t_a},\widehat{t_b},\widehat{t_c}\}$, $\{\widehat{t_a},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{t_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{t_b},\widehat{t_c},\widehat{\bot}\}$
$[\mathbf{f}_a]_1$	$\{\widehat{f_a}\}$,$\{\widehat{f_a},\widehat{f_b}\}$, $\{\widehat{f_a},\widehat{f_c}\}$, $\{\widehat{f_a},\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{f_a},\widehat{\bot}\}$, $\{\widehat{f_a},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{f_a},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{f_a},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_b]_1$	$\{\widehat{t_b}\}$, $\{\widehat{t_b},\widehat{t_c}\}$, $\{\widehat{t_b},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{t_c},\widehat{\bot}\}$
$[\mathbf{f}_b]_1$	$\{\widehat{f_b}\}$, $\{\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_c]_1$	$\{\widehat{t_c}\}$, $\{\widehat{t_c},\widehat{\bot}\}$
$[\mathbf{f}_c]_1$	$\{\widehat{f_c}\}$, $\{\widehat{f_c},\widehat{\bot}\}$
$[\bot]_1$	$\{\widehat{\bot}\}$
$[\mathbf{t}_a\mathbf{f}_a]_1$	$\{\widehat{t_a},\widehat{f_a}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_b}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{f_b}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{f_c}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{f_b}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_b},\widehat{f_c}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_b},\widehat{f_b}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{f_c}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{t_b}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{f_b},\widehat{f_c}\}$ $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{t_b},\widehat{f_b}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{t_b},\widehat{f_c}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{f_c},\widehat{f_b}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_b},\widehat{f_c},\widehat{f_b}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{t_b},\widehat{f_c},\widehat{f_b}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_b},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{t_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{t_b},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{t_b},\widehat{f_c},\widehat{f_b},\widehat{\bot}\}$
$[\mathbf{t}_a\mathbf{f}_b]_1$	$\{\widehat{t_a},\widehat{f_b}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_b}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_c}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{t_c}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{f_c}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_c},\widehat{f_c}\}$,$\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{t_c},\widehat{f_c}\}$,$\{\widehat{t_a},\widehat{f_b},\widehat{\bot}\}$,$\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{\bot}\}$,$\{\widehat{t_a},\widehat{f_b},\widehat{t_c},\widehat{\bot}\}$,$\{\widehat{t_a},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{f_c},\widehat{\bot}\}$,$\{\widehat{t_a},\widehat{f_b},\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_b\mathbf{f}_a]_1$	$\{\widehat{t_b},\widehat{f_a}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{t_c}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{t_c}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{t_c},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{t_c},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_a\mathbf{f}_c]_1$	$\{\widehat{t_a},\widehat{f_c}\}$, $\{\widehat{t_a},\widehat{f_c},\widehat{t_b}\}$, $\{\widehat{t_a},\widehat{f_c},\widehat{t_c}\}$, $\{\widehat{t_a},\widehat{f_c},\widehat{t_b},\widehat{t_c}\}$, $\{\widehat{t_a},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_c},\widehat{t_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_c},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{t_c},\widehat{\bot}\}$
$[\mathbf{t}_b\mathbf{f}_b]_1$	$\{\widehat{t_b},\widehat{f_b}\}$, $\{\widehat{t_b},\widehat{f_b},\widehat{t_c}\}$, $\{\widehat{t_b},\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{f_b},\widehat{t_c},\widehat{f_c}\}$ $\{\widehat{t_b},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_b},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_b},\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_c\mathbf{f}_a]_1$	$\{\widehat{t_c},\widehat{f_a}\}$, $\{\widehat{t_c},\widehat{f_a},\widehat{f_b}\}$, $\{\widehat{t_c},\widehat{f_a},\widehat{f_c}\}$, $\{\widehat{t_c},\widehat{f_a},\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{t_c},\widehat{f_a},\widehat{\bot}\}$, $\{\widehat{t_c},\widehat{f_a},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_c},\widehat{f_a},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_c},\widehat{f_a},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_b\mathbf{f}_c]_1$	$\{\widehat{t_b},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{f_c},\widehat{t_c}\}$, $\{\widehat{t_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_c},\widehat{t_c},\widehat{\bot}\}$
$[\mathbf{t}_c\mathbf{f}_b]_1$	$\{\widehat{t_c},\widehat{f_b}\}$, $\{\widehat{t_c},\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{t_c},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_c},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_c\mathbf{f}_c]_1$	$\{\widehat{t_c},\widehat{f_c}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$

Cuadro A.1: Clases de equivalencia del dominio potencia inferior

A.5.2. Grafo

Con la siguiente relación de orden:

Figura A.1: Relación de orden entre las clases del dominio potencia inferior

A.6. Dominio Potencia Superior

El dominio potencia superior de $\widehat{\mathbf{TB}}$ es el conjunto de ideales sobre el preorden $(\mathcal{P}^*_f(\widehat{\mathbf{TB}}), \sqsubseteq_2)$. Notar que si u y v son subconjuntos no vacíos de $\widehat{\mathbf{TB}}$ y $\widehat{\bot} \in v$, entonces $v \sqsubseteq_2 u$. En particular, cualquier ideal x de $\mathsf{U}(\widehat{\mathbf{TB}})$ contiene a todos los subconjuntos v de $\widehat{\mathbf{TB}}$ con $\widehat{\bot} \in v$. Por ello es necesario establecer clases de equivalencia a partir de la relación $\approx_2$ definida por

$A \approx_2 B$ si y solo si $A \sqsubseteq_2 B$ y $B \sqsubseteq_2 A$

A.6.1. Clases

Tendríamos las siguientes clases:

CLASE	Elementos que pertenecen
$[\bot]_2$	todos los $u \in (\mathcal{P}^*_f(\widehat{\mathbf{TB}}))$ tales que $\widehat{\bot} \in u$
$[\mathbf{t}_a]_2$	$\{\widehat{t_a}\}$
$[\mathbf{f}_a]_2$	$\{\widehat{f_a}\}$
$[\mathbf{t}_b]_2$	$\{\widehat{t_b}\}$, $\{\widehat{t_b},\widehat{t_a}\}$
$[\mathbf{f}_b]_2$	$\{\widehat{f_b}\}$, $\{\widehat{f_b},\widehat{f_a}\}$
$[\mathbf{t}_c]_2$	$\{\widehat{t_c}\}$, $\{\widehat{t_c},\widehat{t_a}\}$, $\{\widehat{t_c},\widehat{t_b}\}$, $\{\widehat{t_c},\widehat{t_b},\widehat{t_a}\}$
$[\mathbf{f}_c]_2$	$\{\widehat{f_c}\}$, $\{\widehat{f_c},\widehat{f_a}\}$, $\{\widehat{f_c},\widehat{f_b}\}$, $\{\widehat{f_c},\widehat{f_b},\widehat{f_a}\}$
$[\mathbf{t}_a\mathbf{f}_a]_2$	$\{\widehat{t_a},\widehat{f_a}\}$
$[\mathbf{t}_a\mathbf{f}_b]_2$	$\{\widehat{t_a},\widehat{f_b}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{f_b}\}$
$[\mathbf{t}_a\mathbf{f}_c]_2$	$\{\widehat{t_a},\widehat{f_c}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{f_c}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{f_b},\widehat{f_c}\}$
$[\mathbf{t}_b\mathbf{f}_a]_2$	$\{\widehat{t_b},\widehat{f_a}\}$, $\{\widehat{t_b},\widehat{t_a},\widehat{f_a}\}$
$[\mathbf{t}_b\mathbf{f}_b]_2$	$\{\widehat{t_b},\widehat{f_b}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b}\}$, $\{\widehat{t_b},\widehat{t_a},\widehat{f_b}\}$, $\{\widehat{t_b},\widehat{t_a},\widehat{f_a},\widehat{f_b}\}$
$[\mathbf{t}_b\mathbf{f}_c]_2$	$\{\widehat{t_b},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{t_a},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{t_a},\widehat{f_a},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{t_a},\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{t_b},\widehat{t_a},\widehat{f_a},\widehat{f_b},\widehat{f_c}\}$
$[\mathbf{t}_c\mathbf{f}_a]_2$	$\{\widehat{f_a},\widehat{t_c}\}$, $\{\widehat{f_a},\widehat{t_a},\widehat{t_c}\}$, $\{\widehat{f_a},\widehat{t_b},\widehat{t_c}\}$, $\{\widehat{f_a},\widehat{t_a},\widehat{t_b},\widehat{t_c}\}$
$[\mathbf{t}_c\mathbf{f}_b]_2$	$\{\widehat{f_b},\widehat{t_c}\}$, $\{\widehat{f_b},\widehat{f_a},\widehat{t_c}\}$, $\{\widehat{f_b},\widehat{t_a},\widehat{t_c}\}$, $\{\widehat{f_b},\widehat{t_b},\widehat{t_c}\}$, $\{\widehat{f_b},\widehat{t_a},\widehat{t_b},\widehat{t_c}\}$, $\{\widehat{f_b},\widehat{f_a},\widehat{t_a},\widehat{t_c}\}$, $\{\widehat{f_b},\widehat{f_a},\widehat{t_b},\widehat{t_c}\}$, $\{\widehat{f_b},\widehat{f_a},\widehat{t_a},\widehat{t_b},\widehat{t_c}\}$
$[\mathbf{t}_c\mathbf{f}_c]_2$	$\{\widehat{t_c},\widehat{f_c}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{f_b}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{t_b}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{f_a}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{t_a}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{t_a},\widehat{f_a}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{t_b},\widehat{f_a}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{t_a},\widehat{t_b}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{f_b},\widehat{f_a}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{t_a},\widehat{f_b}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{t_b},\widehat{b_b}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{t_a},\widehat{f_a},\widehat{t_b}\}$, $\{\widehat{t_c},\widehat{f_c},\widehat{t_a},\widehat{f_a},\widehat{f_b}\}$, $\{\widehat{t_c},\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{t_a},\widehat{t_c},\widehat{t_b},\widehat{f_b},\widehat{f_c}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_b},\widehat{f_b},\widehat{t_c},\widehat{f_c}\}$

Cuadro A.2: Clases de equivalencia en el dominio potencia superior

A.6.2. Grafo

Con la siguiente relación de orden:

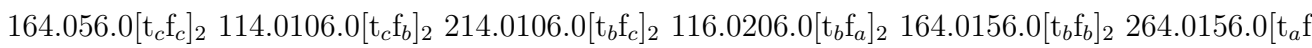


Figura A.2: Relación de orden entre las clases del dominio potencia superior

A.7. Dominio Potencia Convexo

El dominio potencia convexo de $\widehat{\mathbf{TB}}$ es el conjunto de ideales sobre el preorden $(\mathcal{P}^*_f(\widehat{\mathbf{TB}}), \sqsubseteq_3)$. Notar que, en un dominio plano, si u y v son subconjuntos no vacíos de $\widehat{\mathbf{TB}}$, entonces $v \sqsubseteq_3 u$ si y solo si:

1. $\widehat{\perp} \in v$ y $v \subseteq u \cup \{\widehat{\perp}\}$, o

2. $v = u$

Con lo cual el preorden $\sqsubseteq_3$ es fácil de establecer pero debido al gran número de elementos el diagrama resultante es algo farragoso. Para hacernos idea, podemos considerar, en primer lugar, el caso en que el conjunto ordenado **Bo**=$\{$**1,0**$\}$ el diagrama se muestra en la figura A.3

$102.06.0\perp$ $62.046.0t_0\perp$ $142.046.0f_0\perp$ $102.086.0t_0f_0\perp$ $102.0126.0t_0f_0$ 52.0250

Figura A.3: Relación de orden en d.p. convexo (2 elementos)

A.7.1. Clases

Volviendo a nuestro caso, hemos de considerar a los elementos que hacen al dominio no plano, consideremos por ejemplo:

1. Los elementos $\{\widehat{t_c}, \widehat{t_b}\}$ y $\{\widehat{t_b}\}$. Tenemos que en el domonio potencia inferior ambos pertenecen a $[\mathbf{t}_b]_1$, mientras que en dominio potencia superior:

   $\{\widehat{t_c}, \widehat{t_b}\} \in [\mathbf{t}_c]_2$
   $\{\widehat{t_b}\} \in [\mathbf{t}_b]_2$

   por lo que $\{\widehat{t_c}, \widehat{t_b}\} \sqsubseteq_3 \{\widehat{t_b}$

2. Los elementos $\{\widehat{t_c}, \widehat{t_b}\}$ y $\{\widehat{t_c}\}$. Tenemos que en el dominio potencia superior ambos pertenecen a $[\mathbf{t}_c]_2$, mientras que en dominio potencia inferior:

   $\{\widehat{t_c}, \widehat{t_b}\} \in [\mathbf{t}_b]_1$
   $\{\widehat{t_c}\} \in [\mathbf{t}_c]_1$

   por lo que $\{\widehat{t_c}\} \sqsubseteq_3 \{\widehat{t_c},\widehat{t_b}\}$

3. Los elementos $\{\widehat{t_c}, \widehat{t_b}, \widehat{f_c}, \bot\}$ y $\{\widehat{t_b}, \widehat{f_c}, \bot\}$. Tenemos que en el dominio potencia superior ambos pertenecen a $[\bot]_2$, y en el dominio potencia inferior ambos pertenecen a $[\mathbf{t}_b\mathbf{f}_c]_1$. Por tanto:

   $\{\widehat{t_c}, \widehat{t_b}, \widehat{f_c}, \bot\} \sqsubseteq_3 \{\widehat{t_b}, \widehat{f_c}, \bot\}$
   $\{\widehat{t_b}, \widehat{f_c}, \bot\} \sqsubseteq_3 \{\widehat{t_c}, \widehat{t_b}, \widehat{f_c}, \bot\}$

Este último caso muestra que $\sqsubseteq_3$ es tambien un preorden por lo que tenemos que establecer clases de equivalencia. Dichas clases tienen un único elemento salvo en el caso de los elementos que contienen a $\bot$. Para visualizar el diagrama resultante los dividiremos en cuatro subdiagramas:

1. El formado por los elementos que contiene a $\bot$.
2. El formado por los elementos que solo contienen a **t** y no incluidos en el caso anterior.
3. El formado por los elementos que solo contienen a **f** y no incluidos en el caso 1.
4. Los restantes elementos, es decir los que contienen tanto **t** como a **f** y no incluidos en los casos anteriores

Caso 1

Los elementos que contienen a $\bot$ pertenecen a la clase $[\bot]_2$. Por tanto todos los que pertenezcan a la misma clase en el Dominio potencia inferior perteneceran aquí a la misma clase. Ver tabla A.3

CLASE	Elementos que pertenecen
$[\mathbf{t}_a\bot]_3$	$\{\widehat{t_a},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{t_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{t_b},\widehat{t_c},\widehat{\bot}\}$
$[\mathbf{f}_a\bot]_3$	$\{\widehat{f_a},\widehat{\bot}\}$, $\{\widehat{f_a},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{f_a},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{f_a},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_b\bot]_3$	$\{\widehat{t_b},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{t_c},\widehat{\bot}\}$
$[\mathbf{f}_b\bot]_3$	$\{\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_a\mathbf{f}_c\bot]_3$	$\{\widehat{t_a},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_c},\widehat{t_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_c},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{t_c},\widehat{\bot}\}$,
$[\mathbf{t}_c\mathbf{f}_a\bot]_3$	$\{\widehat{t_c},\widehat{f_a},\widehat{\bot}\}$, $\{\widehat{t_c},\widehat{f_a},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_c},\widehat{f_a},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_c},\widehat{f_a},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$,
$[\mathbf{t}_b\mathbf{f}_c\bot]_3$	$\{\widehat{t_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_c},\widehat{t_c},\widehat{\bot}\}$
$[\mathbf{t}_c\mathbf{f}_b\bot]_3$	$\{\widehat{t_c},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_c},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_b\mathbf{f}_b\bot]_3$	$\{\widehat{t_b},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_b},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_b},\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_a\mathbf{f}_b\bot]_3$	$\{\widehat{t_a},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_b},\widehat{t_b},\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_b\mathbf{f}_a\bot]_3$	$\{\widehat{t_b},\widehat{f_a},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_b},\widehat{f_a},\widehat{f_b},\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$
$[\mathbf{t}_a\mathbf{f}_a\bot]_3$	$\{\widehat{t_a},\widehat{f_a},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_b},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{t_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{f_b},\widehat{f_c},\widehat{\bot}\})$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{t_b},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{t_b},\widehat{f_c},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{f_c},\widehat{f_b},\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_b},\widehat{f_c},\widehat{f_b}),\widehat{\bot}\}$, $\{\widehat{t_a},\widehat{f_a},\widehat{t_c},\widehat{t_b},\widehat{f_c},\widehat{f_b},\widehat{\bot}\}$

Cuadro A.3: Clases de equivalencia en d.p. convexo (Caso 1)

A.7.2. Grafo del caso 1

Con la relación de orden que se muestra en la figura A.4

Caso 2 y caso 3

Para el caso 2, tenemos los siguientes elementos:

1. $\{\widehat{t_a}\}$
2. $\{\widehat{t_a},\widehat{t_b}\}$
3. $\{\widehat{t_a},\widehat{t_b},\widehat{t_c}\}$
4. $\{\widehat{t_a},\widehat{t_c}\}$
5. $\{\widehat{t_b}\}$
6. $\{\widehat{t_b},\widehat{t_c}\}$
7. $\{\widehat{t_c}\}$

168.06.0[⊥]$_3$ 118.056.0[t$_c$⊥]$_3$ 218.056.0[f$_c$⊥]$_3$ 68.0106.0[t$_b$⊥]$_3$ 168.0106.0[t$_c$f$_c$⊥]$_3$ 268.0106.0[f$_b$⊥]$_3$ 18

Figura A.4: Relación de orden en d.p. convexo (Caso 1)

Elemento	Inferior	Superior
$\{\widehat{t_a}\}$	$[\mathrm{t}_a]_1$	$[\mathrm{t}_a]_2$
$\{\widehat{t_a},\widehat{t_b}\}$	$[\mathrm{t}_a]_1$	$[\mathrm{t}_b]_2$
$\{\widehat{t_a},\widehat{t_b},\widehat{t_c}\}$	$[\mathrm{t}_a]_1$	$[\mathrm{t}_c]_2$
$\{\widehat{t_a},\widehat{t_c}\}$	$[\mathrm{t}_a]_1$	$[\mathrm{t}_c]_2$
$\{\widehat{t_b}\}$	$[\mathrm{t}_b]_1$	$[\mathrm{t}_b]_2$
$\{\widehat{t_b},\widehat{t_c}\}$	$[\mathrm{t}_b]_1$	$[\mathrm{t}_c]_2$
$\{\widehat{t_c}\}$	$[\mathrm{t}_c]_1$	$[\mathrm{t}_c]_2$

Cuadro A.4: Los elmentos del caso 2) vistos desde el d.p. inferior y superior

En la tabla A.4 mostramos la pertenencia de estos elementos a los dominios potencia superior e inferior:

Por tanto, como podemos observar en la tabla A.5, los elementos 3 y 4, o sea $\{\widehat{t_a},\widehat{t_b},\widehat{t_c}\}$ y $\{\widehat{t_a},\widehat{t_c}\}$, pertenecen a la misma clase de equivalencia tanto en el dominio potencia inferior como en el superior y por tanto también en el dominio potencia convexo,

1 Ele.	2 Ele.	1	2	3	4	5	6
2		$\approx_1$ y $\sqsubseteq_2$					
3		$\approx_1$ y $\sqsubseteq_2$	$\approx_1$ y $\sqsubseteq_2$				
4		$\approx_1$ y $\sqsubseteq_2$	$\approx_1$ y $\sqsubseteq_2$	$\approx_1$ y $\approx_2$			
5		$\sqsubseteq_1$ y $\sqsubseteq_2$	$\sqsubseteq_1$ y $\approx_2$	$\sqsubseteq_1$ y $\sqsupseteq_2$	$\sqsubseteq_1$ y $\sqsupseteq_2$		
6		$\sqsubseteq_1$ y $\sqsubseteq_2$	$\sqsubseteq_1$ y $\sqsubseteq_2$	$\sqsubseteq_1$ y $\approx_2$	$\sqsubseteq_1$ y $\approx_2$	$\approx_1$ y $\sqsubseteq_2$	
7		$\sqsubseteq_1$ y $\sqsubseteq_2$	$\sqsubseteq_1$ y $\sqsubseteq_2$	$\sqsubseteq_1$ y $\approx_2$	$\sqsubseteq_1$ y $\approx_2$	$\sqsubseteq_1$ y $\sqsubseteq_2$	$\sqsubseteq_1$ y $\approx_2$

Cuadro A.5: Relaciones entre los elementos del caso 2)

la llamaremos $t_a t_c$. Además, esta última clase no es comparable con la clase cuyo único elemento es $\{\widehat{t_b}\}$ y al que llamaremos t_b.

Para el caso 3) se puede repetir el razonamiento.

A.7.3. Grafos de los casos 2 y 3

Los grafos de los casos 2 y 3 se muestran en la figura A.5.

66.07148.19t_a 67.68116.8$t_a t_b$ 71.5939.84104.573.18 107.776.37$t_a t_c$ 67.37142.5167.37120.64 67.683

Figura A.5: Relación de orden en d.p. convexo (Casos 2 y 3)

Caso 4

Para esta caso agrupamos los elementos de la forma que se muestra en la figura A.6. Para ello, tomando como base los elementos del caso 2, es decir los elementos

que solo tienen t, le vamos adjuntando los elementos del caso 3 (es evidente que se habrían podido invertir los papeles de caso 2 y 3), obteniendose los siguientes subcasos del caso 4:

1. F_a. Es el subcaso obtenido adjuntando, a cada elemento de caso 2, $\{\widehat{f_a}\}$
2. F_b. Es el subcaso obtenido adjuntando, a cada elemento de caso 2, $\{\widehat{f_b}\}$
3. F_c. Es el subcaso obtenido adjuntando, a cada elemento de caso 2, $\{\widehat{f_c}\}$
4. F_aF_b. Es el subcaso obtenido adjuntando, a cada elemento de caso 2, $\{\widehat{f_a}, \widehat{f_b}\}$
5. F_aF_c. Es el subcaso obtenido adjuntando, a cada elemento de caso 2, $\{\widehat{f_a},\widehat{f_c}\}$ y $\{\widehat{f_a},\widehat{f_b},\widehat{f_c}\}$, ya que ambos pertenecen a la misma clase en el dominio potencia inferior y superior. En efecto:

   $\{\widehat{f_a},\widehat{f_c}\} \approx_1 \{\widehat{f_a},\widehat{f_b},\widehat{f_c}\}$ y $\{\widehat{f_a},\widehat{f_c}\}$,$\{\widehat{f_a},\widehat{f_b},\widehat{f_c}\} \in [\mathbf{f}_a]_1$
   $\{\widehat{f_a},\widehat{f_c}\} \approx_2 \{\widehat{f_a},\widehat{f_b},\widehat{f_c}\}$ y $\{\widehat{f_a},\widehat{f_c}\}$,$\{\widehat{f_a},\widehat{f_b},\widehat{f_c}\} \in [\mathbf{f}_c]_2$
6. F_bF_c. Es el subcaso obtenido adjuntando, a cada elemento de caso 2, $\{\widehat{f_b},\widehat{f_c}\}$

Todos ellos tienen una estructura muy similar, por ejemplo el subcaso 1 estaría formado por los siguientes elementos:

1. $\{\widehat{t_a}, \widehat{f_a}\}$
2. $\{\widehat{t_a},\widehat{t_b},\widehat{f_a}\}$
3. $\{\widehat{t_a},\widehat{t_b},\widehat{t_c},\widehat{f_a}\}$
4. $\{\widehat{t_a},\widehat{t_c},\widehat{f_a}\}$
5. $\{\widehat{t_b},\widehat{f_a}\}$
6. $\{\widehat{t_b},\widehat{t_c},\widehat{f_a}\}$
7. $\{\widehat{t_c},\widehat{f_a}\}$

y si construimos la tabla de las relaciones entre sus elementos, sería idéntica a la tabla A.5. En la figura A.6 se muestran los distintos subcasos del caso 4

62.25479.92$t_a f_a$ 63.87448.53$t_a t_b f_a$ 67.77371.57100.69404.9 103.88408.09$t_a t_c f_a$ 63.56474.2363.5645

Figura A.6: Relaciones de orden en d.p. convexo (Subcasos del caso 4)

A.7.4. Grafo del caso 4

Las relaciones que existen entre sus distintos subcasos del caso 4 se muestra en la figura A.7, en donde las lineas que los unen tienen el significado que los elementos que se encuentran en la misma posición en cada caja están relacionados, por ejemplo:

88.046.0F$_a$F$_c$ 8.066.0F$_b$ 48.06.0F$_b$F$_c$ 86.040.048.010.0 48.010.010.060.0

significa que "entre otras" existen las relaciones relaciones que se muestran en la figura A.8

48.25148.92F$_a$ 49.86117.52F$_a$F$_b$ 53.7640.5786.6973.89 89.8777.08F$_a$F$_c$ 49.5

Figura A.7: Relación de orden en d.p. convexo entre los subcasos (Caso 4)

Además existen relaciones entre los elementos del caso 1, y los de los casos 2, 3 y 4: cada elemento de estos casos está relacionado con la clase de equivalencia del caso 1 cuyo nombre, borrando el $\perp$, está contenido en el nombre del elemento.

Podemos compararla con la anterior

A.8. Resumen

Como vemos los diagramas de orden para los dominios potencia inferior y superior son fáciles de establecer. Aumentar en uno el número de elementos de **Bo**, significa orlar, de forma conveniente, el diagrama con los elementos que resultan al añadir el nuevo elemento.

Sin embargo el diagrama del dominio potencia convexo es algo más complejo. Analizando los dos ejemplos, bastantes simples, que hemos mostrado, observamos que en ambos, el diagrama, se puede descomponer en cuatro subdiagramas:

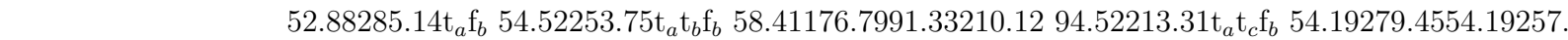


Figura A.8: Relación de orden en d.p. convexo entre algunos subcasos(Caso 4)

- Primer subdiagrama. Similar al del dominio potencia inferior.
- Segundo y tercer subdiagramas. Si el número de elementos de **Bo** es n, el segundo y tercer subdiagrama tendrán tantos nodos como $(n-1)^2+2$, con su orden correspondiente.
- Cuarto subdiagrama. Se descompone a su vez en tantos subdiagramas como nodos existen en los subdiagramas segundo y tercero, es decir en $(n-1)^2+2$. Cada uno de ellos, haciendo abstracción de las etiquetas de los nodos son idénticos a los subdiagramas segundo y tercero. El orden que existe entre ellos, considerándolos como cajas, tal y como hemos hecho para el caso en el que **Bo** tenia tres elementos, repite el mismo patrón de orden.(Observar en la figura A.9 los recuadros etiquetados como: Caja F_0, Caja F_1F_0 y Caja F_1, ellos forman el cuarto subdiagrama. Los nodos que no están en ningún recuadro forman el primer subdiagrama).

Los tres dominios potencia reflejan formas distintas de posibles equivalencias acerca del comportamiento de los programas.

Ejemplo.- Supongamos que tenemos tres programas cuyas entradas son elementos de **TB** y cuyas salidas pertenecen a $\mathcal{P}$(**TB**):

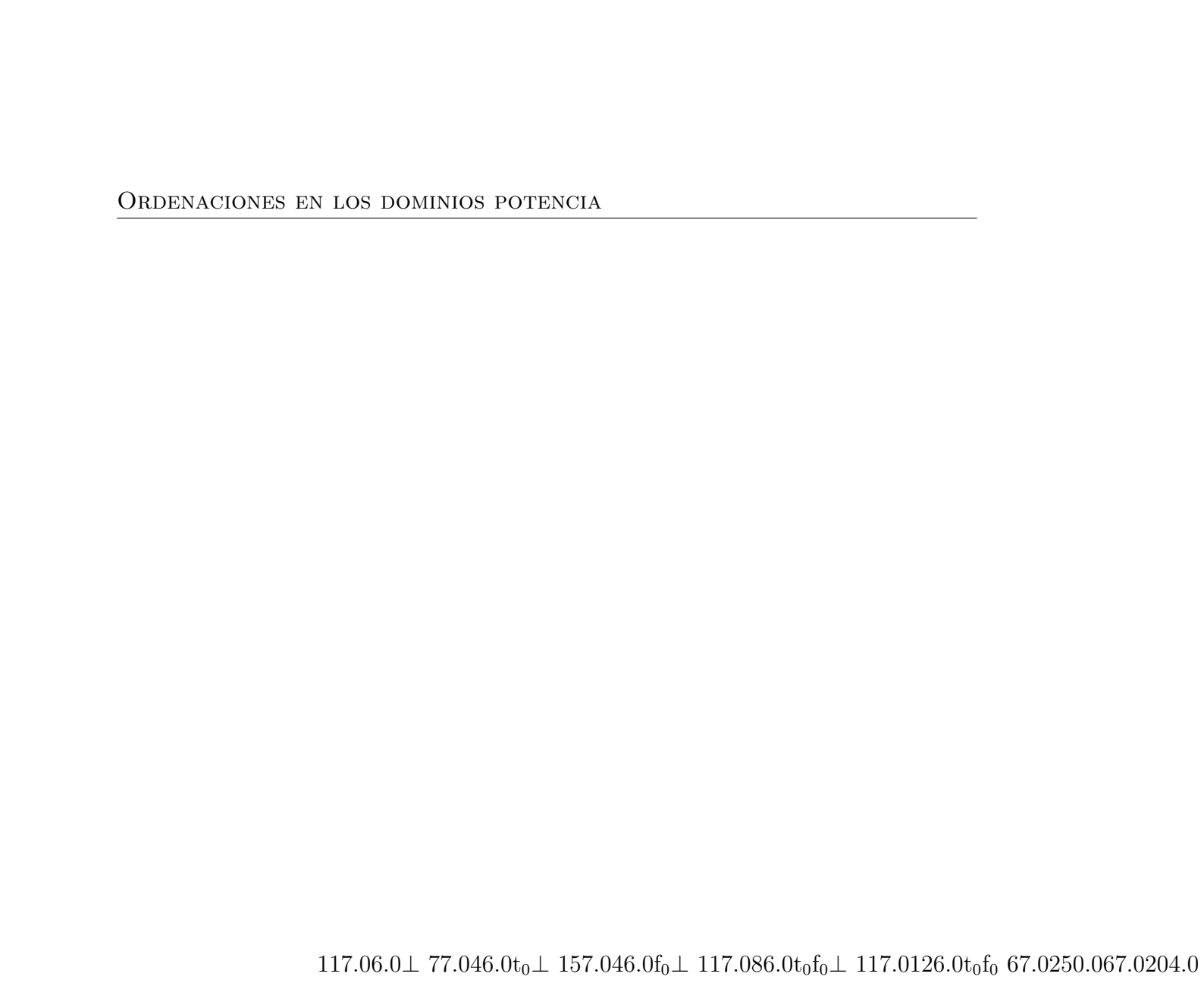


Figura A.9: Descomposición del grafo (2 elementos) en subcasos

- Programa1.- Para cualquier entrada produce $\{\!|(t, b), \bot|\!\}$
- Programa2.- Para cualquier entrada produce $\{\!|\bot|\!\}$
- Programa3.- Para cualquier entrada distinta de $\bot$ produce $\{\!|(t, b)|\!\}$

entonces:

Desde el punto de vista del dominio potencia superior los programas Programa1 y Programa2 son equivalentes.

Desde el punto de vista del dominio potencia inferior los programas Programa1 y Programa3 son equivalentes.

Desde el punto de vista del dominio potencia convexo los tres programas tienen comportamientos distintos.

Por ello es fundamental que, a la hora de realizar optimizaciones, o de discutir equivalencia de programas se considere el dominio potencia utilizado.

Apéndice B

Intérprete de lambda cálculo_b puro

B.1. Introducción

Presentamos a continuación un intérprete del lambda cálculo_b puro, desarrollado en CAML. El adjetivo de puro significa que en su sintaxis sólo son admisibles los elementos introducidos en el capítulo 2. Aunque la función principal de este intérprete ha sido ayudarnos en el estudio de la función de valuación $\mathcal{V}$ del capítulo 3, no por ello dejamos de notar que, a pesar de su sencillez, con él podemos llevar a cabo cualquier cálculo, (consular la página 162 donde se dan algunos operadores y la página 167 para ver algunos ejemplos construidos con ellos).

B.1.1. lambda cálculo_b: sintaxis abstracta

En nuestra definición de la página 46 existen tres constructores de términos, los modificaremos para adecuarlos a un lenguaje de programación:

- $I_\gamma(x) = [x, \gamma]$ lo representaremos por $x : \gamma$
- $F_\gamma(x, T) = [\lambda x.t, \gamma \cdot \ell(T)]$ lo representaremos por $[x : \gamma]\, .t : \ell(T)$.
- $G_\gamma(T, U) = [(t\ u), \gamma \cdot \ell(T) \cdot \ell(U)]$ lo representaremos $(t : \ell(T)\ u : \ell(U))\gamma$

Incluso admitiremos que cuando se presente una expresión de la forma x : neutro de s, la representaremos por x :, el intérprete lo sustituirá por $x : 1$. Utilizaremos esta notación para nuestra *sintaxis concreta*. Además para evitar las dificultades de la $\alpha-$conversión, en los términos las variables no se representaran por medio de identificadores, sino por índices de referencia que identifican canónicamente sus ligaduras en las abstracciones siguiendo la notación llamada los números de Bruijn, por ello el tipo término lo definimos como:

```
type o_num = { Cons: int ; grado: float list} ;;

type termino = Ref of o_num
        | Apl of termino * termino * float list
        | Abs of termino * float list;;
```

B.1.2. La sustitución

La sustitución consiste en reemplazar una posición de algún redex `Apl(Abs(M:m):b,N:n):p` por el resultado de sustituir `N:n` por la primera variable libre de `M:m`, definido como `(subst N:n M:m)` como hacemos a continuación. Esta operación la dividimos en dos pasos. Es necesario explorar `M` recursivamente a fin de encontrar los lugares donde aparece la variable que hemos de sustituir. A continuación, para cada lugar encontrado, necesitamos copiar `N`, haciendo las modificaciones necesarias tanto en cuanto al grado como a que las sus propias variables libres. Esta tarea la realizamos por medio de la función `(eleva n)`, que recalcula las referencias de las variables globales a lo largo de `k` niveles de abstracción

```
let eleva n = eleva_rec 0
where rec eleva_rec k = eleva_k
where rec eleva_k = function
    Ref(i) -> if i.Cons<k then
                  Ref(i) (*variables ligadas no varian*)
              else
                  Ref {Cons =n+i.Cons ; grado = i.grado}
                         (*v.libres reasignadas*)
  | Abs(M,r) -> Abs((eleva_rec(k+1) M),r)
  | Apl(M,N,r) -> Apl((eleva_k M),(eleva_k N),r);;
```

A continuación damos la función de sustitución. Sustituimos `N:n` por `Ref(0)` en `M:m` por medio de `(sust N:n M:m)`

```
let sust N = sust_N 0
where rec sust_N n = function
    Ref(k) -> if k.Cons=n then
                  opS k.grado (eleva n N)
                           (* variable sustituida *)
              else if k.Cons<n then
                  Ref(k)   (* variable ligada *)
              else Ref { Cons = (k.Cons - 1) ; grado = k.grado }
                           (* variable libre *)
  | Abs(M,r) -> Abs((sust_N(n+1) M),r)
  | Apl(M,M1,r) -> Apl((sust_N n M), (sust_N n M1), r);;
```

donde la función `(opS grado)` lleva a cabo la función s definida en 2.1 de la siguiente forma:

```
let opS g = function
          Ref(k) -> Ref { Cons = k.Cons ; grado = (ponS k.grado g) }
        | Abs(M,r) -> Abs(M,(ponS r g))
        | Apl(M1,M2,r) -> Apl(M1,M2,(ponS r g));;
```

B.1.3. La $\beta-$reducción

Como sabemos la principal regla del lambda cálculo_b es la llamada $\beta-$reducción, que consiste en reemplazar un redex, es decir una subexpresión de la forma `([x:a]M:m N:n)`, por `(subst M:m N:n)`. En primer lugar usando el *orden normal* de evaluación construimos la *forma normal en cabeza* de un término por medio de la función:

```
let rec hnf = function
    Ref(n) -> Ref(n)
  | Abs(M,r) -> Abs((hnf M),r)
  | Apl(M1,M2,r) -> (match hnf(M1) with
       Abs(N,r1) -> hnf(opF r r1 (sust (opG r1 M2) N))
     | h -> Apl(h, M2,r));;
```

Finalmente un término se pone en *forma normal*, cuando esta exista, por medio de la función:

```
let rec NF rho = nf
where rec nf = function
    Ref(n) -> Ref(n)
  | Abs(M,r) -> Abs(nf M,r)
  | Apl(M1,M2,r) -> match hnf(M1) with
       Abs(N,r1) -> nf(opF r r1 (sust (opG r1 M2) N))
     | h -> Apl((nf h), (nf M2), r);;
```

donde las funciones `ponF` y `ponG`:

```
let opF n n1 = function
    Ref(v) -> Ref { Cons = v.Cons ; grado = (ponF n n1 v.grado) }
  | Abs(M,r) -> Abs(M,(ponF n n1 r))
  | Apl(M1,M2,r) -> Apl(M1,M2, (ponF n n1 r));;
let opG n = function
    Ref(v) -> Ref { Cons = v.Cons ; grado = (ponG v.grado n) }
  | Abs(M,r) -> Abs(M,(ponG r n))
  | Apl(M1,M2,r) -> Apl(M1,M2, (ponG r n));;
```

implantan las funciones f y g definidas en 2.10.

B.2. Funciones recursivas

Si definimos las funciones recursivas como los elementos del menor conjunto $\mathbb{E}$ que contiene a las funciones base:

- la constante $0 : \mathbb{N} \to \mathbb{N}$
- la función sucesor **S**: $\mathbb{N} \to \mathbb{N}$
- la función proyección $pr_k^i : \mathbb{N}^k \to \mathbb{N}$, para $1 \leq i \leq k$

y es cerrado para:

- la composición, es decir si $g, h_1, \ldots, h_p \in \mathbb{E}$, si f se define por

  $$f(\vec{n}) = g(h_1(\vec{n}), \ldots, h_p(\vec{n})) \text{ entonces } f \in \mathbb{E}$$

- la recursión primitiva, es decir si psra $h, g \in \mathbb{E}$, si f se define por

  $$\begin{aligned} f(0, \vec{n}) &= h(\vec{n}) \\ f(m+1, \vec{n}) &= g(f(m, \vec{n}), m, \vec{n}) \end{aligned}$$
  entonces $f \in \mathbb{E}$

- y la minización total, es decir si para $g \in \mathbb{E}$ tal que para cada $\vec{n}$, existe p tal que $g(\vec{n}, p) = 0$, si f se define por:

  $$f(\vec{n}) = \min \{p \in N | g(\vec{n}, p) = 0\} \text{ entonces } f \in \mathbb{E}$$

entonces con nuestro interprete podemos programar cualquier función recursiva. Para ello, y siendo las funciones base triviales, basta mostrar como se programan la composición, la recursión primitiva y la minimización. Consultar [17] para más detalles. Utilizaremos $Z_0, \ldots$ para referirnos a numerales y $\lceil q \rceil$ par la función que λ-define a q

Previamente vamos a mostrar como programar los pares ordenados y el predecesor.

B.2.1. D (Operador de pares ordenados)

Su propiedad característica es que existen términos **D1** y **D2** tales que (para todo X e Y)

$$\mathbf{D1}(\mathbf{D}XY) = X, \qquad \mathbf{D2}(\mathbf{D}XY) = Y \tag{B.1}$$

Esta propiedad es independiente de la forma en que se hayan definido los numerales, por lo que **D**, **D1** y **D2** no están asociados con una determinada teoría de números.

Algunas definiciones de **D** pueden tener la propiedad de que existan objetos A y B tales que

$$\mathbf{D}XYA = X, \qquad \mathbf{D}XYB = Y \tag{B.2}$$

en este caso B.1 se satisface tomando

$$\mathbf{D1} \equiv \lambda x.xA \qquad \mathbf{D2} \equiv \lambda x.xB \tag{B.3}$$

donde x no aparece libre ni en A ni en B. En la literatura aparecen dos formas de definir **D** con las propiedades B.1, si tomamos la de Bernays

```
DB = [x :][y :][z :]((z : ([x :][u :]x :  y :)1,0)1,0 x :)1,0
```

`DB` satisface B.2 con $A = Z_0$ y $B = Z_1$. Por tanto tomamos

- `DB1 = [x:](x: [x:][y:]y:)1.0`
- `DB2 = [x:](x: [x:][y:](x: y:)1.0)1.0`

Es de notar que para ninguna de las versiones de **D**, **D1** y **D2** se cumple

$$\mathbf{D}(\mathbf{D1}A \quad \mathbf{D2}A) = A$$

En [6] se demuestra que en el λ-cálculo sin tipos no pueden existir operadores **D**, **D1** y **D2** que verifiquen la igualdad anterior

B.2.2. $\lceil\pi\rceil$ (Operador predecesor)

Su propiedad característica es

$$\lceil\pi\rceil\lceil 0\rceil = \lceil 0\rceil, \qquad \lceil\pi\rceil(\lceil\sigma\rceil\lceil n\rceil) = \lceil n\rceil \tag{B.4}$$

donde $\lceil\sigma\rceil = \mathbf{S}$. Una forma de definirlo a partir de **D** es:

```
ChP = [u:](((u: [v:]((&DB(&ChS (v: [x:][y:]y:)1.0)1.0)1.0
      (v:[x:][y:]y:)1.0)1.0)1.0 ((&DB[x:][y:]y:)1.0
      [x:][y:]y:)1.0)1.0 [x:][y:](x: y:)1.0)1.0
```

otra forma, donde no interviene **D**, es:

```
ChP1 = [u:](((u: [p:][u:]((u: (&ChS (p: [x:][y:]x:) 1.0)1.0)1.0
       (p: [x:][y:]x:)1.0)1.0)1.0 [u:]((u: &Ch0)1.0 &Ch0)1.0)1.0
       [x:][y:]y:)1.0
```

B.2.3. Composición

Sea $g : \mathbb{N}^p \to \mathbb{N}$ y $h_i : \mathbb{N}^k \to \mathbb{N}$ $(1 \leq i \leq p)$, las funciones λ-definidas por los términos G y H_i. Si

$$f(n_1, \ldots, n_k) = g(h_1(n_1, \ldots, n_k), \ldots, h_p(n_1, \ldots, n_k))$$

sea

$$F \equiv \lambda x_1 \ldots x_k . G(H_1 x_1 \ldots x_k) \ldots (H_p x_1 \ldots x_k)$$

entonces F λ-define a f

B.2.4. Recursión

La utilización de un operador de punto fijo es una solución elegante y potente pero nuestro interprete no puede utizarla. De hecho los puntos fijos no son ua definición sintáctica de la recursión sino un esquema de computación muy partícular. La idea es reemplazar la recursión por la iteración. Por ello vamos a construir un combinador de recursión primitiva **R** normalizable. Su propiedad característica es que para todo G, H

$$\mathbf{R}GH\lceil 0\rceil = G, \qquad \mathbf{R}GH(\lceil\sigma\rceil\lceil n\rceil) = H\lceil n\rceil(\mathbf{R}GH\lceil n\rceil)$$

Primeramente vamos a construir una definición de **R** debida a Bernays. Si una función f es definida por las ecuaciones

$$f(0) = g$$

$$f(n+1) = h(n, f(n))$$

entonces de puede calcular $f(n)$ de la siguiente forma:

> Empezamos escribiendo el par ordenado $< 0, g >$ y a continuación iteramos n veces la operación φ, la cual cambia el par ordenado $< k, f(k) >$ por el par $< k+1, h(f(k)) >$. El valor de $f(n)$ es el segundo miembro del último par ordenado obtenido de esta manera.

El **R** que vamos a construir es el análogo de este proceso de cálculo. Sea **D** tal que cumple la propiedad B.1, la operación φ, considerada como dependiente tanto de h como del par ordenado, puede representarse por

$$\mathbf{Q} \equiv \lambda yv.\mathbf{D}(\lceil\sigma\rceil(\mathbf{D1v}))(\mathbf{y}(\mathbf{D1v})(\mathbf{D2v}))$$

ya que para cualesquiera obs H, X y cualquier n

$$\mathbf{Q}H(\mathbf{D}\lceil n\rceil X) = \mathbf{D}\lceil n+1\rceil(H\lceil n\rceil X)$$

Además, para todo G, H, n y para cualquier X

$$(\mathbf{Q}H)(\mathbf{D}\lceil 0\rceil)G) = \mathbf{D}\lceil n\rceil X \tag{B.5}$$

en efecto este X representa el valor de $f(n)$ en la discusión anterior si G y H λ-definen a g y h respectivamente. Por tanto definimos

$$\mathbf{R} \equiv \lambda xyu.\mathbf{D2}(\mathbf{Zu}(\mathbf{Qy})(\mathbf{D}\lceil \mathbf{0}\rceil \mathbf{x})) \tag{B.6}$$

entonces para todo G, H, n

$$\begin{aligned} \mathbf{R}GH\lceil n\rceil &= \mathbf{D2}(Z_n(\mathbf{Q}H)(\mathbf{D}\lceil 0\rceil G)) \\ &= \mathbf{D2}((\mathbf{Q}H)(\mathbf{D}\lceil 0\rceil G)) \\ &= \mathbf{D2}(\mathbf{D}\lceil n\rceil X) \\ &= X \end{aligned}$$

En particular se tiene

$$\begin{aligned} \mathbf{R}GH\lceil 0\rceil &= \mathbf{D2}(\mathbf{D}\lceil 0\rceil G) \\ &= G \end{aligned}$$

y

$$\begin{aligned} \mathbf{R}GH\lceil n+1\rceil &= \mathbf{D2}((\mathbf{Q}H)(\mathbf{D}\lceil 0\rceil G)) \\ &= \mathbf{D2}(\mathbf{Q}H((\mathbf{Q}H)(\mathbf{D}\lceil 0\rceil G))) \\ &= \mathbf{D2}(\mathbf{Q}H(\mathbf{D}\lceil n\rceil X)) \\ &= H\lceil n\rceil X \\ &= H\lceil n\rceil(\mathbf{R}GH\lceil n\rceil) \end{aligned}$$

Si en B.5 y B.6 **D** se toma como **DB**, con el apropiado **DB1** y **DB2**, entonces el resultado es la definición de **Q** siguiente:

```
Q = [y:][v:]((&DB(&ChS(&DB1 v:)1.0)1.0)1.0((y: (&DB1 v:)
    1.0)1.0 (&DB2 v:)1.0)1.0) 1.0

R = [x:][y:][u:](&DB2 ((u: (&Q: y:)1.0)1.0((&DB [x:][y:]y:)
    1.0 x:)1.0)1.0)1.0
```

B.2.5. $\lceil$Pe$\rceil$ (p-función de Kleene)

Sirve para definir las funciones generales recursivas. Sus propiedades características son:

$$\begin{aligned} &\lceil \mathbf{Pe}\rceil XY = Y \text{ si } (XY) = \lceil 0\rceil \\ &\lceil \mathbf{Pe}\rceil XY = \lceil \mathbf{Pe}\rceil X(\lceil \sigma\rceil Y) \text{ si } (XY) = \lceil m+1\rceil \text{ para algún } m \end{aligned} \tag{B.7}$$

Para cualquier función numérica g definida por $\lceil g \rceil$ si n es el menor número para el cual g(n) = 0, entonces

$$\lceil \mathbf{Pe} \rceil \lceil g \rceil \lceil 0 \rceil = \lceil n \rceil$$

Para definir **Pe** utilizamos las funciones auxiliares **J** y **P** definidas por:

```
J = [x:][u:][v:](((u: (x: (&ChS v:) 1.0)1.0)1.0 u:)1.0
    (&ChS v:)1.0)1.0

P = [x:]((&DB (&K &I) 1.0)1.0 (&J x:)1.0)1.0
```

con lo cual la definición de **Pe** es:

```
Pe = [x:][y:]((((&P x:)1.0 (x: y:)1.0)1.0(&P x:)1.0)1.0 y:)1.0
```

B.2.6. $\lceil$Gp$\rceil$ (Generalización de la p-función)

Su propiedad característica es:

$$\begin{array}{l} \lceil \mathbf{Gp} \rceil XYZU = XU \text{ si } (ZU) = \lceil 0 \rceil \\ \lceil \mathbf{Gp} \rceil XYZU = Y(\lceil \mathbf{Gp} \rceil XYZ)U \text{ si } (ZU) = \lceil m+1 \rceil \text{ para algún } m \end{array} \tag{B.8}$$

Para definirlo utilizamos el operador **D**, con la ayuda del cual distinguiremos los casos en que $ZU = \lceil 0 \rceil$ y $ZU = \lceil m+1 \rceil$; esto sugiere que se defina

$$\lceil \mathbf{Gp} \rceil \equiv \lambda xyzu.\mathbf{DMN}(\mathbf{zu})\mathbf{L_1}, \ldots, \mathbf{L_k}$$

para unos convenientes $M, N, L_1, \ldots, L_k$. Entonces

$$\begin{array}{l} \lceil \mathbf{Gp} \rceil XYZU = M'L'_1 \ldots L'_k \text{ si } (ZU) = \lceil 0 \rceil \\ \lceil \mathbf{Gp} \rceil XYZU = N'L'_1 \ldots L'_k \text{ si } (ZU) = \lceil m+1 \rceil \text{ para algún } m \end{array} \tag{B.9}$$

donde $M', N', L'_1, \ldots, L'_k$ resultan de sustituir X, Y, Z, U por x, y, z, u en $M, N, L_1, \ldots, L_k$ respectivamente. Para obtener la primera parte de B.8 a partir de aquí se necesita que

$$M'L'_1 \ldots L'_k = XU$$

y esto es cierto si se define

$$M \equiv K^{k-1}x. \qquad L_k \equiv u$$

La segunda parte de B.8 se satisface si

$$N'L'_1 \ldots L'_k = Y(\lceil \mathbf{Gp} \rceil XYZ)U$$

la cual, en vista de las definiciones propuestas anteriormente, es implicada por

$$NL_1 \dots L_k = y(\lambda u.\mathbf{DMN}(\mathbf{zu})\mathbf{L_1} \dots \mathbf{L_{k-1}}\mathbf{u})$$

Pero esta última ecuación es satisfecha sise toma $k = 2$, y

$$L_1 \equiv \mathbf{D}MN, \qquad N \equiv \lambda v.y(\lambda u.v(zu)vu)$$

ya que entonces

$$\begin{aligned} NL_1 &= N(\mathbf{D}MN) \\ &= y(\lambda u.\mathbf{DMN}(\mathbf{zu})(\mathbf{DMN})\mathbf{u}) \end{aligned}$$

Para definir **Gp** utilizaremos la función auxiliar **QGp** definida por :

```
QGp = [x:][y:][z:] ((&DB (&K x:)1.0)1.0 [v:](y: [u:]
      (((v: (z: u:)1.0)1.0 v:)1.0 u:)1.0)1.0)1.0
```

con lo cual la definición de **Gp** es:

```
Gp = [x:][y:][z:][u:] ((((((&QGp x:)1.0 y:)1.0 z:)1.0
      (z: u:)1.0 )1.0 (((&QGp x:)1.0 y:)1.0 z:)1.0)1.0 u:)1.0
```

B.3. Ejemplos

1 La **suma**

```
G1 = [x:]x:
H1 = [x:][y:][z:] (\&ChS z:)1.0
suma = [x:]((&R (&G1 x:)1.0)1.0 (&H1 x:)1.0)1.0
```

2 La función **sumar 2**

```
G2 = &Ch2
H2 = [x:][y:](&ChS y:) 1.0
suma2 = ((&R &G2)1.0 &H2) 1.0
```

3 La función **duplicar**

```
GDUP = &Ch0
HDUP = [x:](&SUMA [z:](&Ch2 z:)1.0)1.0
dup = ((&R &GDUP) 1.0 &HDUP)1.0
```

4 La función **producto**

```
GPR = [x:][y:][z:]z:
HPR = [x:][y:][z:]((&SUMA x:) 1.0 z:) 1.0
pro = [x:] ((&R (&GPR x:)1.0)1.0 (&HPR x:)1.0)1.0
```

5 La **exponenciación**

```
GExP = ((&DB &Ch0) 1.0 &Ch1:) 1.0
HExP = [x:][y:][z:] ((&PROD x:) 1.0 z:) 1.0
exp = [x:] ((&R (&GExP x:) 1.0 )1.0 (&HExP x:) 1.0)1.0
```

6 La **resta**

```
GSUB = [x:]x:
HSUB = [x:][y:][z:] (&ChP1 z:) 1.0
sub = [x:] ((&R (&GSUB x:)1.0)1.0 (&HSUB x:)1.0)1.0
```

7 El **factorial**

```
Gfac = &Ch1
Hfac = [x:][y:] ((&PROD (&ChS x:)1.0) 1.0 y:) 1.0
fact = ((&R &Gfac) 1.0 &Hfac) 1.0
```

8 La función $f(\mathbf{x}) = \mathbf{(x+2)*(x*2)}$
Definimos:

```
Io21 = [g:][h1:][h2:][x:]((g: (h1: x:)1.0)1.0(h2: x:)1.0)1.0
```

y entonces la función que λ-define a f sería

```
[x:]((((&Io21 &PROD) 1.0 &suma2)1.0 &dup:)1.0 x:) 1.0
```

Fichero: analisis.mli

```
(*  analisis.mli *)

#open "stream";;

type o_num = { Cons: int ; grado: string list} ;;

type o_iden = { Iden: string ; gra : string list};;

type termino = Ref of o_num
        | Apl of termino * termino * string list
        | Abs of termino * string list
;;

type token =
    IDENT of string
  | ID_BO of o_iden
  | INT of int
  | AMPER
  | EQUAL
  | LET
  | LPAREN
  | RPAREN
  | LCUA
  | RCUA
  | SEMICOL
  | REAL of string
  | GRADO of string
;;
type top_asl = Decl of string * termino ;;
type 'a option = Nada | Algo of 'a;;
exception Error of string;;
exception No_ligado of string;;

value next_token :char stream -> token stream ;;
value reset_lexer: char stream -> unit ;;

value p_e : int -> int -> int list -> termino -> string stream ;;
value top : token stream -> top_asl;;
value init_env : o_iden list;;
value global_env : o_iden list ref;;
value init_sem : termino list ;;
value global_sem : termino list ref;;
```

Fichero: analisis.ml

```
(* analisis.ml *)

#open "sys" ;;
#open "stream";;

let I x = x;;

let keywords =
  let t = hashtbl__new 13 in
  hashtbl__add t "let" LET;
  t
;;

let ident_buff = make_string 8 ` `;;

let rec igno = function
[< '` `|`\t`|`\n`; igno _ >] -> ()
| [< >] -> ()
;;
let rec ident tam = function
    [< '(`a`..`z` | `A`..`Z` | `0`..`9` | `_` | `'`) as c;
    (
       if tam >= 8 then ident tam
                   else
                     begin
                       set_nth_char ident_buff tam c;
                       ident (succ tam)
                     end
    )
   s >] -> s
  | [< >] -> let str = sub_string ident_buff 0 tam in
    (try hashtbl__find keywords str with _ -> IDENT str)
;;

let rec N_real1 n = function
    [< ''0` .. `9` | `.` as c; (N_real1(n^char_for_read(c))) r >] -> r
  | [< >] ->  if string_length (n) = 1 then
                   GRADO ("1")
              else GRADO(n)
;;

let rec N1_real nn = function
    [< '(`0`..`9` | `.`) as c; (N1_real(nn^char_for_read(c))) r >] -> r
  | [< >] -> nn
;;

let rec grados n = function
    [< '(`0`..`9` | `.`) as c; (grados(n^char_for_read(c))) r >] -> r
  | [< >] ->  if string_length (n) = 0 then
                   "1"
              else n
;;

let rec Numero1 n = function
    [< ''0` .. `9` as c; (Numero1(n^char_for_read(c))) r >] -> r
  | [< ''.` as c;(N1_real(n^char_for_read(c))) r >]->REAL (r)
;;


let rec ident_bo tam = function
```

```
    [< '(`a`..`z` | `A` .. `Z` | `0` .. `9` | `_` | `'`) as c;
    (if tam >= 8 then ident_bo tam
        else begin
                set_nth_char ident_buff tam c;
                ident_bo (succ tam)
end) s >] -> s
  | [< '`:` ; ( grados("")) p >] ->
    let g = (if string_length (p) = 1 then "1" else p) in
             let str = sub_string ident_buff 0 tam in
             (try hashtbl__find keywords str with _ ->
                ID_BO {Iden = str ; gra = g::[] }
            )
;;

let oper = function
    [< 'c >] -> prerr_string "Caracter ilegal: ";
                prerr_endline (char_for_read c);
                raise (Failure "Lexico")
  | [< >] -> prerr_endline "Fin de la entrada";
                raise (End_of_file)
;;

let rec next_token str = igno str ;
   match str with
      [< '(`a`..`z` | `A` .. `Z`) as c;
         (set_nth_char ident_buff 0 c; ident_bo 1 ) tok; igno _ >] ->
             [< 'tok; next_token str >]
    | [< '`0`..`9` as d; (Numero1 (char_for_read(d))) tok; igno _ >] ->
     [< 'tok; next_token str >]
    | [< '`&` >] -> [< 'AMPER   ; next_token str >]
    | [< '`=` >] -> [< 'EQUAL   ; next_token str >]
    | [< '`;` >] -> [< 'SEMICOL ; next_token str >]
    | [< '`(` >] -> [< 'LPAREN  ; next_token str >]
    | [< '`)` >] -> [< 'RPAREN  ; next_token str >]
    | [< '`[` >] -> [< 'LCUA    ; next_token str >]
    | [< '`]` >] -> [< 'RCUA    ; next_token str >]
    | [< '`:`; (N_real1(char_for_read(`0`))) tok; igno _ >] ->
        [< 'tok ; next_token str >]
    | [< '`'`; (ident 0) tok; igno _ >] -> [< 'tok; next_token str >]
    | [< oper tok; igno _ >] -> [< 'tok ; next_token str >]
    | [< 'x >] -> failwith ("Error caracter: "^make_string 1 x)
;;

let rec reset_lexer = function
      [< '`\n` >] -> ()
    | [< '_; reset_lexer _ >] -> ()
    | [< >] -> ()
;;

let init_env =  [{Iden="I";gra="1"::[]}]
;;

let global_env = ref init_env
;;

let caml_function = function
"I" -> Abs(Ref {Cons = 0 ; grado = "1"::[]},"1"::[])
;;

let init_sem =  map (fun x -> caml_function x.Iden) init_env
;;
```

```
let global_sem = ref init_sem
;;

let nth1 n = enesimo n
   where rec enesimo m = function
              []  -> raise (No_ligado "no ligado")
           | x::l -> if m=0 then x else enesimo (m-1) l
;;

let rec list p = function
    [< p x; (list p) l >] -> x::l
  | [< >] -> []
;;

let option p = function
    [< p x >] -> Algo x
  | [< >] -> Nada
;;

let jotaesimo1 s (global_env:o_iden list ref) = enesimo 0 !global_env
   where rec enesimo m = function
     []  -> raise (No_ligado s.Iden)
           | x::l -> if s.Iden=x.Iden then
                         nth1 m !global_sem
             else
                 enesimo (m+1) l
;;

let jotaesimo s = function
            _ -> jotaesimo1 s global_env
;;

let binding_depth s (rho:o_iden list) = lig 0 rho
   where rec lig n = function
    [] -> raise (No_ligado s.Iden)
          | t::l -> if s.Iden=t.Iden then
                        Ref { Cons = n; grado= s.gra}
                    else lig (n+1) l
;;

(* analisis *)

let rec expr = function
    [< 'LCUA ; 'ID_BO s ; 'RCUA ; expr e >] ->
        (function rho -> Abs( e(s::rho),s.gra))

  | [< 'LPAREN ; expr x1 ; expr x2 ; 'RPAREN ; 'REAL r >] ->
(fun rho -> Apl(x1 rho, x2 rho,r::[]))
  | [< expr1 e >] -> e

and expr1 = function
    [< 'ID_BO s >] ->  binding_depth s
  | [< 'CIRCUN ; 'ID_BO s >] -> jotaesimo s
;;

let top = function
    [< 'LET ;'IDENT s;'EQUAL;expr e;'SEMICOL >] -> Decl(s, e !global_env)
  | [< expr e ; 'SEMICOL >] -> Decl(" ", e !global_env)
  | [< '_ >] -> raise Parse_error
;;

(* imprimir el arbol *)
```

```
let rec print_list = function
    [] -> [< >]
  | x::l -> [< 'x;print_list l >]
;;

let rec ver_ari n v = function
    [] -> (-v)
  | h::t -> if v <= h+n then h+n-v
                else ver_ari n v t
;;

(* imprimir lambda termino *)
let rec p_e n m aridad = function
    Abs(a,r) -> (match a with
          Abs(M,r1) -> [< '"[x"; 'string_of_int (n+m);'":";
                   print_list r; '"]" ; p_e (n+1) m aridad a >]
| _ -> [< '"[x"; 'string_of_int (n+m);'":";
                   print_list r ; '"]" ; p_e n (m+1) (m::aridad) a >] )
  | Apl(e1, e2,r) -> [< '"("; p_e n m aridad e1; '" ";
      p_e n m aridad e2; '")"; '":"; print_list r >]
  | Ref v -> if (ver_ari n v.Cons aridad < 0 ) then
                [< '"u";'string_of_int((-(ver_ari n v.Cons aridad))-1);
                   '":"; print_list v.grado >]
             else [< '"x"; 'string_of_int ((ver_ari n v.Cons aridad));
                   '":"; print_list v.grado >]
;;

let p_t s = top(next_token(stream_of_string s));;
```

Fichero: sintesis.mli

```
#open "analisis";;

value cima : top_asl -> termino ;;
value print_expr1 : termino -> string stream ;;
```

Fichero: sintesis.ml

```
(* sintesis.ml *)

#open "analisis";;
#open "sys" ;;

let print_expr1 M = p_e 0 0 [] M
;;

let escri x = do_stream print_string(print_expr1(x));print_newline()
;;

let parse_top s =top(next_token(stream_of_string s));;

exception ERROR of string
;;

let eleva n = eleva_rec 0
    where rec eleva_rec k = eleva_k
       where rec eleva_k = function
          Ref(i) -> if i.Cons<k then
                          Ref(i) (*variables ligadas no varian*)
                    else
                          Ref {Cons =n+i.Cons ;
                               grado = i.grado
                              }       (*v.libres reasignadas*)
        | Abs(M,r) -> Abs((eleva_rec(k+1) M),r)
        | Apl(M,N,r) -> Apl((eleva_k M),(eleva_k N),r)
;;

let ponS n1 n2 = ("S{"::n2) @ ((", "::n1) @ ("}"::[]))
;;

let opS g = function
          Ref(k) -> Ref { Cons = k.Cons ; grado = (ponS k.grado g) }
        | Abs(M,r) -> Abs(M,(ponS r g))
        | Apl(M1,M2,r) -> Apl(M1,M2,(ponS r g))
;;

let sust N = sust_N 0
where rec sust_N n = function
          Ref(k) -> if k.Cons=n then
                       opS k.grado (eleva n N)
                                (* variable sustituida *)
                    else if k.Cons<n then
                       Ref(k)    (* variable ligada     *)
                    else Ref { Cons = (k.Cons - 1) ; grado = k.grado }
                                (* variable libre      *)
        | Abs(M,r) -> Abs((sust_N(n+1) M),r)
        | Apl(M,M1,r) -> Apl((sust_N n M), (sust_N n M1), r)
;;

let ponF m1 m2 m3 = ("F\\" ::m1) @ ((", "::m2) @ ((", "::m3) @ ("/"::[])))
;;

let ponG n1 n2 = ("G<"::n2) @ ((", "::n1) @ (">"::[]))
;;

let opF n n1 = function
          Ref(v) -> Ref { Cons = v.Cons ; grado = (ponF n n1 v.grado) }
        | Abs(M,r) -> Abs(M,(ponF n n1 r))
```

```
        | Apl(M1,M2,r) -> Apl(M1,M2, (ponF n n1 r))
;;

let opG n = function
          Ref(v) -> Ref { Cons = v.Cons ; grado = (ponG v.grado n) }
        | Abs(M,r) -> Abs(M,(ponG r n))
        | Apl(M1,M2,r) -> Apl(M1,M2, (ponG r n))
;;

let rec hnf = function
          Ref(n) -> Ref(n)
        | Abs(M,r) -> Abs((hnf M),r)
        | Apl(M1,M2,r) -> (match hnf(M1) with
                           Abs(N,r1) -> hnf(opF r r1 (sust (opG r1 M2) N))
                         | h -> Apl(h, M2,r))
;;

let rec NF rho = nf
   where rec nf = function
             Ref(n) -> Ref(n)
           | Abs(M,r) -> Abs(nf M,r)
           | Apl(M1,M2,r) -> match hnf(M1) with
                             Abs(N,r1) -> nf(opF r r1 (sust (opG r1 M2) N))
                           | h -> Apl((nf h), (nf M2), r);;

let cima = function Decl(s,e) ->
  let result = NF !global_sem e
  in global_env := {Iden=s;
     gra= "1"::[]}::!global_env;
     global_sem := result::!global_sem;
     match result with
       _ -> if s = " " then
                begin
                print_string":- ";
                result
                end
            else
                begin
                  print_string"Declarado: ";
                  print_string s;
                  print_string " = ";
                  print_newline();
                  result
                end
;;
let nada rho s = s;;

let alta = function Decl(s,e) ->
  let result = nada !global_sem e
  in global_env := { Iden = s ; gra = "1"::[]}::!global_env;
     global_sem := result::!global_sem        ;
                print_string"\"";
                print_string s;print_string"\", ";
                flush std_out
;;

let escribe s = do_stream print_string(print_expr1(cima(parse_top s)));;
let da_alta s = alta(parse_top s);;

da_alta "'let 'K = [x:][y:] x: ;";;

da_alta "'let 'S = [x:][y:][z:] ((x: z:) 1.0 (y: z:) 1.0 ) 1.0 ;";;
```

```
da_alta "'let 'A = [f:][x:] (f: x:) 1.0 ;";;
da_alta "'let 'True = [x:][y:]x: ;";;
da_alta "'let 'False = [x:][y:]y: ;";;

da_alta "'let 'Ch0 = [f:][x:]x: ;";;
da_alta "'let 'ChS = [n:][f:][x:](f: ((n: f:) 1.0 x:) 1.0 ) 1.0 ;";;
da_alta "'let 'Ba0 = [x:]x: ;";;
da_alta "'let 'BaS = [x:][z:]((z: [u:][y:] y:) 1.0 x:) 1.0 ;";;

da_alta "'let 'D = [x:][y:][z:]((z: x:) 1.0 y:) 1.0 ;";;
da_alta "'let 'D1 = [x:](x: [x:][y:]x:) 1.0 ;" ;;
da_alta "'let 'D2 = [x:]((x: [x:][y:] x:) 1.0 [x:]x:) 1.0 ;" ;;
da_alta "'let 'DB = [x:][y:][z:]((z: ([x:][u:]x:  y:) 1.0 ) 1.0 x:) 1.0 ;";;
da_alta "'let 'DB1 = [x:](x: [x:][y:]y:) 1.0 ;" ;;
da_alta "'let 'DB2 = [x:](x: [x:][y:](x: y:) 1.0 ) 1.0 ;" ;;

da_alta "'let 'Ch0 = [x:][y:]y: ;" ;;
(* ZeroC ver Bar pag 1.041.0 *)

da_alta "'let 'Smas = [x:][z:]((z: [u:][v:] v:) 1.0 x:)1.0 ;";;
da_alta "'let 'Cero = [x:](x: [x:][y:]x:) 1.0 ;";;
da_alta "'let 'ZeroC = [n:](&Cero: ((n: &Smas:)1.0 [x:]x:)1.0 )1.0 ;";;

da_alta "'let 'ZeroB = [x:](x: [h:][t:][u:][y:] y: ) 1.0 ;";;

da_alta "'let 'Ch1 = (&ChS: &Ch0:) 1.0 ; ";;
da_alta "'let 'Ch2 = (&ChS: &Ch1:) 1.0 ; ";;
da_alta "'let 'Ch3 = (&ChS: &Ch2:) 1.0 ; ";;

da_alta "'let 'Ch4 = (&ChS: &Ch3:) 1.0 ; ";;
da_alta "'let 'Ch5 = (&ChS: &Ch4:) 1.0 ; ";;
da_alta "'let 'Ch6 = (&ChS: &Ch5:) 1.0 ; ";;
da_alta "'let 'Ch7 = (&ChS: &Ch6:) 1.0 ; ";;
da_alta "'let 'Ch8 = (&ChS: &Ch7:) 1.0 ; ";;
da_alta "'let 'Ch9 = (&ChS: &Ch8:) 1.0 ; ";;
da_alta "'let 'Ch10 = (&ChS: &Ch9:) 1.0 ; ";;
da_alta "'let 'Ba1 = (&BaS: &Ba0:) 1.0 ; ";;
da_alta "'let 'Ba2 = (&BaS: &Ba1:) 1.0 ; ";;

da_alta "'let 'Ba3 = (&BaS: &Ba2:) 1.0 ; ";;
da_alta "'let 'Ba4 = (&BaS: &Ba3:) 1.0 ; ";;
da_alta "'let 'Ba5 = (&BaS: &Ba4:) 1.0 ; ";;
da_alta "'let 'Ba6 = (&BaS: &Ba5:) 1.0 ; ";;
da_alta "'let 'Ba7 = (&BaS: &Ba6:) 1.0 ; ";;
da_alta "'let 'Ba8 = (&BaS: &Ba7:) 1.0 ; ";;
da_alta "'let 'Ba9 = (&BaS: &Ba8:) 1.0 ; ";;
da_alta "'let 'Ba10 = (&BaS: &Ba9:) 1.0 ; ";;


da_alta "'let 'Q = [y:][v:] ((&DB: (&ChS: (&DB1: v:)1.0)1.0)1.0
                  ((y: (&DB1: v:)1.0)1.0 (&DB2: v:)1.0)1.0) 1.0 ;";;
da_alta "'let 'R = [x:][y:][u:] (&DB2: ((u: (&Q: y:)1.0)1.0
                  ((&DB: [x:][y:]y:)1.0 x:)1.0) 1.0)1.0;";;

da_alta "'let 'G1 = [x:]x:;";;
da_alta "'let 'H1 = [x:][y:][z:] (&ChS: z:)1.0;";;
da_alta "'let 'suma = [x:] ((&R: (&G1: x:)1.0)1.0 (&H1: x:)1.0)1.0;";;
da_alta "'let 'SUMA =[m:][n:][g:][z:]((m: g:) 1.0 ((n: g:) 1.0 z:) 1.0 ) 1.0 ;";;

da_alta "'let 'G2 = &Ch2:;";;
da_alta "'let 'H2 = [x:][y:]( &ChS: y:) 1.0;";;
da_alta "'let 'suma2 = ((&R: &G2:)1.0 &H2:) 1.0 ;";;
```

```
da_alta "'let 'GDUP = &Ch0: ;";;
da_alta "'let 'HDUP = [x:] (&suma: [z:](&Ch2: z:) 1.0) 1.0 ;";;
da_alta "'let 'dup = ((&R: &GDUP:) 1.0 &HDUP:) 1.0 ;";;


da_alta "'let 'GPR = [x:][y:][z:]z: ;";;
da_alta "'let 'HPR = [x:][y:][z:]((&SUMA: x:) 1.0 z:) 1.0 ;";;
da_alta "'let 'pro = [x:] ((&R: (&GPR: x:) 1.0 ) 1.0 (&HPR: x:) 1.0 ) 1.0;";;
da_alta "'let 'PROD =[m:][n:][g:](m: (n: g:) 1.0 ) 1.0 ;";;

da_alta "'let 'ChP = [u:](((u: [v:]((&DB: (&ChS: (v: [x:][y:]y:)1.0)1.0)
                     1.0 (v: [x:][y:]y:)1.0)1.0)1.0 ((&DB: [x:][y:]y:)1.0
                     [x:][y:]y:)1.0)1.0 [x:][y:](x: y:)1.0)1.0 ;";;

da_alta "'let 'GExP = ((&DB: &Ch0:) 1.0 &Ch1:) 1.0 ;";;
da_alta "'let 'HExP = [x:][y:][z:] ((&PROD: x:) 1.0 z:) 1.0 ;";;
da_alta "'let 'exp = [x:] ((&R: (&GExP: x:) 1.0 ) 1.0 (&HExP: x:) 1.0 ) 1.0  ;";;

da_alta "'let 'HIN1 = [x:][y:] (&BaS: y:) 1.0 ;";;

da_alta "'let 'ZeroB = [x:](x: [x:][y:]x:) 1.0 ;";;
da_alta "'let 'PME1 = [x:] (x: [x:][y:]y:) 1.0 ;";;
da_alta "'let 'BaP = [x:] (x: [x:][y:]y:)1.0;";;

da_alta "'let 'Pepe = [x:](x: (&ChS: [x:][y:]y:) 1.0 ) 1.0 ;";;
da_alta "'let 'HNO = [x:] (((&D: [x:][y:]y:) 1.0 (&Pepe: x:) 1.0 ) 1.0
                     (&ZeroB: x:) 1.0 ) 1.0;";;
da_alta "'let 'HIN = ((&R: [x:]x:) 1.0 &HIN1:) 1.0 ;";;

da_alta "'let 'C1 = [x:][y:][z:] (x:(y: z:) 1.0 ) 1.0;";;
da_alta "'let 'C2 = [z:] (&ZeroB: (&HIN: z:) 1.0 ) 1.0 ;";;
da_alta "'let 'pepe1 = ((&C1: &BaP:) 1.0 &HIN:) 1.0 ;";;
da_alta "'let 'Pr1 = ((&C1: &HNO:) 1.0 &pepe1:) 1.0 ;";;

da_alta "'let 'Cond = [p:][q:][r:] ((p: q:) 1.0 r:) 1.0 ;";;

da_alta "'let 'GSUB = [x:]x:;";;
da_alta "'let 'HSUB = [x:][y:][z:] (&ChP: z:) 1.0 ;";;
da_alta "'let 'sub = [x:] ((&R: (&GSUB: x:) 1.0 ) 1.0 (&HSUB: x:) 1.0 ) 1.0;";;

da_alta "'let 'GMAx = [x:]x:;";;
da_alta "'let 'HMAx = [x:][y:] (((&SUMA: x:) 1.0 (&sub: y:) 1.0) 1.0 x: ) 1.0 ;";;
da_alta "'let 'max =  [x:][y:] (((&DB: (&GMAx: y:) 1.0 ) 1.0
                     ((&HMAx: x:) 1.0 y:) 1.0) 1.0 x:) 1.0 ;";;

da_alta "'let 'GMIN = [x:]x:;";;
da_alta "'let 'HMIN = [x:][y:] (((&sub: y:) 1.0 (&sub: y:) 1.0) 1.0 x: ) 1.0  ;";;
da_alta "'let 'min =  [x:][y:] (((&DB: (&GMIN: x:) 1.0 ) 1.0
                     ((&HMIN: x:) 1.0 y:) 1.0) 1.0 x:) 1.0 ;";;


da_alta "'let 'GFAC = [x:][y:] (x: y:) 1.0 ;";;
da_alta "'let 'HFAC = [x:][y:] ((&suma: y:) 1.0 y:) 1.0 ;";;
da_alta "'let 'pot2 = ((&R: &GFAC:) 1.0 &HFAC:) 1.0 ;";;

da_alta "'let 'Sig = [f:][x:][y:][z:]((f: (x: z:) 1.0 ) 1.0 (y: z:) 1.0 ) 1.0;";;
da_alta "'let 'Dani = (((&Sig: &PROD:) 1.0 &dup:) 1.0 &suma2:) 1.0 ;";;
```

Fichero: principa.mli

```
(* principa.mli *)

value read_fun : unit -> char;;
value go : unit -> unit;;
value input_stream : in_channel ref;;
value trace_parsing : bool ref;;
```

Fichero: principa.ml

```
(* principa.ml $ *)

#open "stream";;
#open "analisis";;
#open "sintesis";;
#open "sys";;

let input_stream = ref std_in;;
let trace_parsing = ref false;;

let fichero =
for i = 1 to vect_length command_line -1 do
  match command_line.(i) with
    "-p" ->
      trace_parsing := true
  | _ -> try input_stream := open_in command_line.(i) with
sys__Sys_error msg -> prerr_endline("Fichero: "^command_line.(i)^" no existe ")
done
;;

let print_prompt() =
  print_newline();print_string ">> "; flush std_out
;;

let read_fun =
  let bol = ref true in
fun() ->
  if !bol then print_prompt();
  let c = input_char !input_stream in
  if !input_stream != std_in then print_char c;
  bol := c == `\n`;
  c
;;

let except_ene = except_l_n
  where rec except_l_n = fun
    [] _ -> []
  | (elem::l) n -> if n = 0 then l else elem::except_l_n l (n-1)
;;

exception Break;;

let rec go() =
  try
    let cstrm =  stream_from  read_fun in
    let strm =  next_token cstrm in
    while true do
      try
let ta = top strm in
print_newline();
let (Decl(s,_)) = ta in
do_stream print_string(print_expr1(cima (ta)))
      with
Parse_failure ->
  print_newline();
  raise Break
      | End_of_file ->
  input_stream := std_in ;
  go()
```

```
      | Parse_error ->
  print_newline();
  print_string "*** Error sintactico."; print_newline();
  reset_lexer cstrm; stream_next strm; ()
      | Error s ->
  print_newline();
  print_string "*** Error: "; print_string s; print_newline();
  raise Break
    done
  with
    Break ->
      ()
  | Failure s ->
      print_string "*** Fallo: "; print_string s; print_newline()
;;
fichero ;;
go();;
```

Apéndice C

Prototipo

El prototipo que presentamos a continuación está programado en Objective Caml (ocaml) (1.07) [42], un lenguaje de programación funcional orientado a objetos, y es un desarrollo del lenguaje diseñado en el capítulo 5. Existe, por tanto, una dependencia entre ambos, puesta de maniesto por las referencias cruzadas que hemos establecido entre el código que mostramos a continuación y dicho capítulo. Sólo hemos añadido algún parámetro, para facilitar la puesta a punto del programa, y algunas funciones, para llevar a cabo las comprobaciones propias de la semántica estática.

La función de cada uno de los ficheros es la siguiente:

gramat.mli es la pieza clave del programa, contiene la descripción abstracta del lenguaje.

estrat.mli y estrat.ml contienen la definición de los operadores triangulares

lexer.mll y parser.mly contienen las especificaciones para la construcción, por medio de *ocamllex* y *ocamlyacc*, del analizador de léxico y sintáctico respectivamente

algseman.mli y algseman.ml contienen las operaciones que se llevan a cabo en las álgebras semánticas. Es de notar que se han implantado dos versiones de la función modificar almacén: `mod_alm` y `mod_alm_g` ya que en las funciones y procedimientos el paso de parámetros y la devolución de un valor se realizan modificando el almacén, pero dicha modificación debe llevarse a cabo al margen del índice de borrosidad

exvaluac.mli y exvaluac.ml contienen todo lo relativo a la evaluación de expresiones

ffvaluac.mli y ffvaluac.ml contienen las funciones de valuación correspondientes a las declaraciones y sentencias del lenguaje

main.ml contiene las funciones iniciales cuando se invoca el prototipo.

Makefile contiene las órdenes para que el sistema produzca el ejecutable

Fichero: gramat.mli

```
open Estrat;;
type programa =
    { texto: bloque }
and bloque=
    { decls : declaraciones;
      cuerpo : sentencia }
and declaraciones = declaracion list
and declaracion =
    | LDeclConsts of decl_consts list
    | LDeclVars   of decl_vars list
    | LDeclReco   of decl_reco list
    | LDeclAbst   of declAbst list
and declAbst =
    | DeclFunc    of decl_func
    | DeclProc    of decl_proc
    | DeclPre     of decl_pre
and sentencia =
    | Asignacion  of identifi * expresion * int
    | Asignacion2 of identif * expresion * int
    | Asignacion3 of identif * int * expresion * int
    | Llamada     of identif * argumentos * int
    | Condicional of expresion * sentencia * sentencia * int
    | Bucle       of expresion * sentencia * int
    | Print       of expr_imprime list * int
    | Ret_expr    of identif * expresion * int
    | Bloq        of bloque * int
    | Retorna     of int
    | Compuesta   of sentencia * sentencia
    | Sentencia_nula
    | Alternativa of guard list * int
    | Repeticion  of guard list * int
and decl_consts = (identif * constante) list
and guard = (expresion * sentencia)
and decl_vars = (identif * tipador) list
and decl_reco = {
    id_reco     : identif;
    cuerpo_reco : tipador }
and decl_pre = identif * identifi
and decl_func = {
    id_func     : identif;
    gr_func     : float;
    es_func     : int ;
    params_func : decl_vars;
    tipo_rdo    : tipador;
    cuerpo_func : bloque }

and decl_proc = {
    id_proc     : identif;
    gr_proc     : float;
    es_proc     : int ;
    params_proc : decl_vars;
    cuerpo_proc : bloque }

and argumentos = expresiones

and expr_imprime =
    | Expres of expresion
    | Cadena of string
    | Escape of expr_escape
and expr_escape =
```

```
    | NuevaLine
    | Tabulador

and identifi =
    | Variable    of identif
    | VariableC   of identif * int
    | VariableR   of identif list

and expresiones = expresion list
and expresion =
    | Constante   of constante
    | Id          of identifi
    | Aplicacion  of identifi * argumentos
    | Aplicacion1 of identifi * argumentos
    | Op_unaria   of identif  * expresion
    | Op_binaria  of identif  * expresion * expresion
    | Op_r        of expresion
    | Op_i        of expresion
    | Grado       of expresion
    | C_fuzzy     of expresion * expresion
    | Default

and tipador =
    | Tipo  of identif
    | TipoC of declaraciones * pares list

and pares = identif * identif * float

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | Literales,Identificadores y Constantes
 | ====================================
 | - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Diseño en página 123 *)

and identif   = string
and identifs  = identif list
and grado     = float
and borr      = float * float * float * float
and constante =
    | NumBc  of int   * grado
    | BoolBc of bool  * grado
    | RealBc of float * grado
    | NumTc  of borr  * grado
    | CBorc  of int * int * (int * grado) list
    | Textc  of string * grado
;;
```

Fichero: estrat.mli

```
type num_lin = {mutable line:int }

val ini_lin    : int -> num_lin
val mas_lin    : num_lin -> unit
val lin_act    : num_lin -> int
val emp_lin    : num_lin -> unit

type estra     = {neutro:float; opera:int }

val estrategia : int -> estra
val k_est      : estra
val l_est      : int -> estra
val tnorma     : float * float -> estra -> float
val tconorma   : float * float -> estra -> float
val nega       : float -> estra -> float
```

Fichero: estrat.ml

```
open Sys;;

type num_lin = { mutable line: int}
type estra = {neutro:float;opera:int }

let ini_lin numero = { line = numero } ;;
let mas_lin l      = l.line <- l.line +1 ;;
let lin_act l      = l.line ;;
let emp_lin l      = l.line <- 1;;

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - --
 | Estrategia
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - --*)


(* Diseño en página 127 *)

let estrategia = (function n ->
  match n with
| 1 -> { neutro = 1.; opera = 1}
| 2 -> { neutro = 1.; opera = 2}
| 3 -> { neutro = 1.; opera = 3}
| 4 -> { neutro = 0.; opera = 4}
| 5 -> { neutro = 0.; opera = 5}
| 6 -> { neutro = 0.; opera = 6}
| _ -> { neutro = 1.; opera = 1}
  )
;;

let k_est = estrategia (int_of_string(argv.(2)));;
let l_est(x) = estrategia (x);;

let tnorma(g,g1) = function estrate ->
  match estrate.opera with
    | 1 -> if g < g1 then g else g1
    | 2 -> g *. g1
    | 3 -> let w = g +. g1 -. 1. in
      if 0.0 < w then w else 0.0
    | 4 -> if g < g1 then g1 else g
    | 5 -> g +. g1 -. (g *. g1)
    | 6 -> let w = g +. g1 in
      if 1. < w then 1. else w
;;

let tconorma(g,g1) = function k_est ->
  match k_est.opera with
    | 1 -> if g < g1 then g1 else g
    | 2 -> g +. g1 -. (g *. g1)
    | 3 -> let w = g +. g1 in
      if 1. < w then 1. else w
    | 4 -> if g < g1 then g else g1
    | 5 -> g *. g1
    | 6 -> let w = g +. g1 -. 1. in
      if 0.0 < w then w else 0.0
;;
let nega g = function k_est ->
  match k_est.opera with
    | 1 -> (1. -. g)
    | 2 -> (1. -. g)
    | 3 -> (1. -. g)
    | 4 -> (1. -. g)
```

```
    | 5 -> (1. -. g)
    | 6 -> (1. -. g)
;;
```

Fichero: lexer.mll

```
{
open Parser;;
open Gramat;;
open Estrat;;
exception Eof;;
let lin = ini_lin 1;;
}
rule token = parse
  [' ' '\t']       { token lexbuf }
  | '\n'           { (mas_lin) lin ; token lexbuf }
  | '\\' [^'\n']*  { token lexbuf }
  | ['0'-'9']+     { ENT(int_of_string (Lexing.lexeme lexbuf)) }
  | ['0'-'9']+'.'['0'-'9']*
                   { REAL(float_of_string (Lexing.lexeme lexbuf)) }
  | '-'+['0'-'9']+'.'['0'-'9']*
                   { REAL(float_of_string (Lexing.lexeme lexbuf)) }
  | "true"         { BOOL(true)  }
  | "false"        { BOOL(false) }
  | '"' [^'"']* '"'
                   { let str=(Lexing.lexeme lexbuf) in
                     let lstr=(String.length str)-2
                     in CADENA(String.sub str 1 lstr) }
  | "function"     { FUNC     }
  | "procedure"    { PROC     }
  | "begin"        { INICIO   }
  | "end"          { FIN      }
  | "var"          { VARS     }
  | "if"           { SI       }
  | "then"         { ENTONCES }
  | "else"         { SINO     }
  | "while"        { MIENTRAS }
  | "do"           { HAZ      }
  | "print"        { PRINT    }
  | "RETORNA"      { RETORNA  }
  | "tab"          { TAB      }
  | "nl"           { NEWLINE  }
  | "type"         { DEFINIR  }
  | "as"           { COMO     }
  | "const"        { CONSTS   }
  | "and"          { AND      }
  | "or"           { OR       }
  | "not"          { NOT      }
  | "default"      { DEFAULT  }
  | "DO"           { DO       }
  | "OD"           { OD       }
  | "CASE"         { CASE     }
  | "ESAC"         { ESAC     }
  | "GRADO"        { GRADO    }
  | "EXTR"         { EXTR     }
  | "EXTI"         { EXTI     }
  | "NEUTRO"       { NEUT     }
  | "VA_LI"        { VA_LI    }
  | "TIPO_LI"      { TI_LI    }
  | "ANTI"         { ANTI     }
  | "INCLUDE"      { (emp_lin) lin ; INCLUDE    }
  | "U"            { UNION    }
  | "N"            { INTER    }
  | "C"            { COMPLE   }
  | "P"            { PRODUC   }
  | "in"           { IN       }
```

```
| '|'            { BARRA    }
| '['            { CORCi    }
| ']'            { CORCd    }
| "->"           { FLECHA   }
| '+'            { MAS      }
| '-'            { MENOS    }
| '*'            { POR      }
| '/'            { DIV      }
| '.'            { PUNTO    }
| ':'            { DOSPTOS  }
| ','            { COMA     }
| ';'            { PTOCOMA  }
| '('            { PARENi   }
| ')'            { PARENd   }
| '{'            { LLAVEi   }
| '}'            { LLAVEd   }
| '>'            { MAYOR    }
| '<'            { MENOR    }
| ">="           { MAYORIG  }
| "<="           { MENORIG  }
| '='            { IGUAL    }
| "!="           { NOIGUAL  }
| ":="           { ASIGVAR  }
| "<-"           { ARROW    }
| ['A'-'z'] ['_' 'A'-'z' '0'-'9']*
                 { ID(Lexing.lexeme lexbuf) }
| eof            { raise Eof}
| _              { print_string ("Caracter ilegal "
                   ^ (Lexing.lexeme lexbuf));
                   print_newline();
                   flush stdout;
                   token lexbuf
                 }
```

Fichero: parser.mly

```
%{
open Gramat;;
open Estrat;;
let neutro = k_est.neutro ;;
%}

%token <int>              ENT
%token <bool>             BOOL
%token <float>            REAL
%token <Gramat.identif>   ID
%token <string>           CADENA

%token FUNC PROC INICIO FIN CONSTS VARS ASIGVAR
%token SI ENTONCES SINO MIENTRAS HAZ PRINT RETORNA
%token CASE ESAC DO OD FLECHA ARROW GRADO EXTR EXTI DEFAULT
%token TAB NEWLINE
%token ENT BOOL REAL ID NEUT
%token VA_LI DEFINIR TI_LI ANTI BARRA COMO INCLUDE
%token PUNTO DOSPTOS COMA PTOCOMA PARENi PARENd
%token CORCi CORCd LLAVEi LLAVEd
%token MAYOR MENOR MAYORIG MENORIG IGUAL NOIGUAL
%token MAS MENOS POR DIV AND OR NOT UNION INTER COMPLE PRODUC IN
%left AND               /* operador con la menor precedencia */
%left OR
%left UNION
%left INTER
%left PRODUC
%nonassoc NOT IN COMPLE
%nonassoc MAYOR MENOR MAYORIG MENORIG IGUAL NOIGUAL
%left MAS MENOS
%left POR DIV
%nonassoc UMENOS        /* operador con la mayor precedencia */

%start pROGRAMA
%type <Gramat.programa>      pROGRAMA
%type <Gramat.bloque>        Bloque
%type <Gramat.bloque>        Bloque1
%type <Gramat.decl_func>     Decl_func
%type <Gramat.decl_proc>     Decl_proc
%type <Gramat.decl_consts>   Decl_consts
%type <Gramat.decl_vars>     Decl_vars
%type <Gramat.sentencia>     Sentencia
%type <Gramat.sentencia>     SentenciaSimple
%type <Gramat.expr_imprime>  Expr_imprimible
%type <Gramat.expr_escape>   Expr_escape
%type <Gramat.sentencia>     Rama_else_o_fin
%type <Gramat.identifs>      Lista_identifs
%type <Gramat.identifi>      Identifi
%type <Gramat.argumentos>    Lista_Arguments
%type <Gramat.decl_vars>     Lista_decl_params
%type <Gramat.expresiones>   Lista_Expresiones
%type <Gramat.expresion>     Expresion
%type <Gramat.tipador>       Tipador
%type <Gramat.constante>     Constante
%type <Gramat.pares list>    Lista_pares

%%
pROGRAMA: Bloque PUNTO
          { { texto  = $1 } }
```

```
  | Bloque1
          { { texto  = $1 } }
;
Bloque: Declaraciones INICIO Sentencia FIN
          { { decls  = $1 ;
              cuerpo = $3 } }
;
Bloque1: Declaraciones FIN INCLUDE
          { { decls  =$1;
              cuerpo =Sentencia_nula } }
;
Declaraciones:
  /* nada */                        { []        }
  | Declaraciones Opt_constantes     { $1 @ [$2] }
  | Declaraciones Opt_P_o_F          { $1 @ $2   }
  | Declaraciones Opt_types          { $1 @ [$2] }
  | Declaraciones Opt_variables      { $1 @ [$2] }

;
Opt_P_o_F:
  Procedures_o_Funcion              { [LDeclAbst($1)]      }
  | Opt_P_o_F Procedures_o_Funcion { $1 @ [LDeclAbst($2)] }
;
Opt_types:
   DEFINIR Decl_reco PTOCOMA        { LDeclReco($2)        }
;
Opt_constantes:
  CONSTS Lista_decl_consts PTOCOMA { LDeclConsts([$2])    }
;
Opt_variables:
  VARS Lista_decl_vars PTOCOMA     { LDeclVars[$2]        }
;

Procedures_o_Funcion:
  FUNC Decl_func PTOCOMA           { [DeclFunc($2)]       }
  | PROC Decl_proc PTOCOMA         { [DeclProc($2)]       }
;
Lista_decl_consts:
  Decl_consts                      { $1                   }
;
Lista_decl_vars:
  Decl_vars                        { $1                   }
  | Lista_decl_vars PTOCOMA Decl_vars { ($1 @ $3)         }
;
Decl_func: ID PARENi Lista_decl_params PARENd G_Opt E_Opt DOSPTOS Tipador Bloque
            { { id_func     = $1;
                gr_func     = $5;
                es_func     = $6;
                params_func = $3;
                tipo_rdo    = $8;
                cuerpo_func = $9 }
            }
;
Decl_proc: ID PARENi Lista_decl_params PARENd G_Opt E_Opt Bloque
            { { id_proc     = $1;
                gr_proc     = $5;
                es_proc     = $6;
                params_proc = $3;
                cuerpo_proc = $7 }
            }
;
Decl_reco: ID IGUAL Tipador
```

```
            { [({ id_reco = $1;
              cuerpo_reco = $3 })]
            }

| Decl_reco COMA ID IGUAL Tipador
            { $1 @ [({ id_reco = $3;
              cuerpo_reco = $5 })]
            }
;
Decl_consts:
  ID IGUAL Constante                    { [($1,$3)]               }
  | Decl_consts COMA ID IGUAL Constante { $1 @ [($3,$5)]          }
;
Decl_vars: Tipador Lista_identifs
            { List.map (function id -> (id,$1)) $2 }
;
Sentencia:
  SentenciaSimple                       { $1 }
  | Sentencia PTOCOMA SentenciaSimple   { Compuesta       ($1, $3) }
;
SentenciaSimple:
  | ID LLAVEi ENT LLAVEd ASIGVAR Expresion
                                  { Asignacion3($1,$3,$6,lin_act Lexer.lin)}
  | Identifi ASIGVAR Expresion     {Asignacion($1,$3,lin_act Lexer.lin)}
  | GRADO PARENi ID PARENd ASIGVAR Expresion
                                    { Asignacion2($3,$6,lin_act Lexer.lin)}
  | ID PARENi Lista_Arguments PARENd    { Llamada  ($1,$3,lin_act Lexer.lin)}
  | MIENTRAS Expresion HAZ Sentencia FIN HAZ
                                    { Bucle    ($2,$4,lin_act Lexer.lin)}
  | PRINT PARENi Exprs_imprimibles PARENd
                                    { Print      ($3,lin_act Lexer.lin) }
  | ID ARROW Expresion              { Ret_expr ($1,$3,lin_act Lexer.lin)}
  | RETORNA                         { Retorna    (lin_act Lexer.lin)    }
  | Bloque                          { Bloq       ($1,lin_act Lexer.lin) }
  | SI Expresion ENTONCES Sentencia Rama_else_o_fin
                                  { Condicional($2,$4,$5,lin_act Lexer.lin)}
  | CASE Lista_Guardado ESAC        { Alternativa($2,lin_act Lexer.lin) }
  | DO Lista_Guardado OD            { Repeticion ($2,lin_act Lexer.lin) }
;
Lista_Guardado:
  Guardado                          { [$1]                    }
  | Lista_Guardado CORCi CORCd Guardado { $1 @ [$4]            }
;
Guardado: Expresion FLECHA Sentencia    { ($1,$3)               }
;
Exprs_imprimibles:
  Expr_imprimible                   { [$1]                    }
  | Exprs_imprimibles COMA Expr_imprimible
                                    { $1 @ [$3]               }
;
Expr_imprimible:
  Expresion                         { Expres($1)              }
/*  | CADENA                          { Cadena($1)              }
*/
  | Expr_escape                     { Escape($1)              }
;
Expr_escape:
  TAB                               { Tabulador               }
  | NEWLINE                         { NuevaLine               }
;
Rama_else_o_fin:
  FIN SI                            { Sentencia_nula          }
```

```
  | SINO Sentencia FIN SI              { $2                        }
;
Lista_identifs:
  ID                                  { [$1]                      }
  | Lista_identifs COMA ID            { $1 @ [$3]                 }
;
Lista_identifs1:
  ID                                  { [$1]                      }
  | Lista_identifs1 PUNTO ID          { $1 @ [$3]                 }
;
Lista_Arguments:
  /* nada */                          { []                        }
  | Lista_Expresiones                 { $1                        }
;
Lista_decl_params:
  /* nada */                          { []                        }
  | Lista_decl_vars                   { $1                        }
;
Lista_Expresiones:
  Expresion                           { [$1]                      }
  | Lista_Expresiones COMA Expresion  { $1 @ [$3]                 }
;
Identifi:
  | ID                                { Variable            ($1) }
  | ID LLAVEi ENT LLAVEd              { VariableC        ($1,$3) }
  | ID PUNTO Lista_identifs1          { VariableR    ([$1] @ $3) }
;
Expresion:
  | Identifi                          { Id($1)                    }
  | Constante                         { Constante           ($1) }
  | NEUT                              { Constante (RealBc(neutro,neutro)) }
  | GRADO PARENi Expresion PARENd     { Grado               ($3) }
  | Identifi PARENi Lista_Arguments PARENd
                                      { Aplicacion       ($1,$3) }
  | ANTI PARENi Identifi PARENi Lista_Arguments PARENd PARENd
                                      { Aplicacion1      ($3,$5) }
  | PARENi Expresion PARENd           { $2                        }
  | EXTR PARENi Expresion PARENd      { Op_r ($3)                 }
  | EXTI PARENi Expresion PARENd      { Op_i ($3)                 }
  | Expresion MAS Expresion           { Op_binaria   ("+",$1,$3) }
  | Expresion MENOS Expresion         { Op_binaria   ("-",$1,$3) }
  | Expresion POR Expresion           { Op_binaria   ("*",$1,$3) }
  | Expresion DIV Expresion           { Op_binaria   ("/",$1,$3) }
  | MENOS Expresion %prec UMENOS      { Op_unaria       ("-",$2) }
  | Expresion AND Expresion           { Op_binaria ("AND",$1,$3) }
  | Expresion OR Expresion            { Op_binaria  ("OR",$1,$3) }
  | NOT Expresion                     { Op_unaria     ("NOT",$2) }
  | COMPLE Expresion                  { Op_unaria     ("COM",$2) }
  | Expresion MAYOR Expresion         { Op_binaria   (">",$1,$3) }
  | Expresion MENOR Expresion         { Op_binaria   ("<",$1,$3) }
  | Expresion MAYORIG Expresion       { Op_binaria  (">=",$1,$3) }
  | Expresion MENORIG Expresion       { Op_binaria  ("<=",$1,$3) }
  | Expresion IGUAL Expresion         { Op_binaria   ("=",$1,$3) }
  | Expresion NOIGUAL Expresion       { Op_binaria  ("!=",$1,$3) }
  | Expresion UNION Expresion         { Op_binaria ("UNI",$1,$3) }
  | Expresion INTER Expresion         { Op_binaria ("INT",$1,$3) }
  | Expresion PRODUC Expresion        { Op_binaria ("PRO",$1,$3) }
  | Expresion IN Expresion            { Op_binaria  ("IN",$1,$3) }
  | Expresion BARRA Expresion         { C_fuzzy          ($1,$3) }
  | DEFAULT                           { Default                   }
;
Tipador:
```

```
  ID                                    { Tipo                ($1) }
  | TI_LI Lista_decl_vars VA_LI L_pred Opt_Anti FIN
                                        {TipoC([(LDeclVars[$2])]@$4,$5)}
;
L_pred:
  | Pred                    { $1 }
  | ID COMO Identifi        {([LDeclAbst[DeclPre($1,$3)]]) }
  | L_pred BARRA Pred       { $1 @ $3 }
  | L_pred BARRA ID COMO Identifi
                            { $1 @ ([LDeclAbst[DeclPre($3,$5)]])    }
;
Pred:
  ID DOSPTOS PARENi Lista_decl_params PARENd G_Opt E_Opt T_Opt Bloque
                            { [LDeclAbst[DeclFunc({ id_func = $1;
                               gr_func     = $6;
                               es_func     = $7;
                               params_func = $4;
                               tipo_rdo    = $8;
                               cuerpo_func = $9 })]]
                            }
;
Constante:
  ENT G_Opt                             { NumBc            ($1,$2) }
  | BOOL G_Opt                          { BoolBc           ($1,$2) }
  | REAL G_Opt                          { RealBc           ($1,$2) }
  | LLAVEi REAL COMA REAL COMA REAL COMA REAL LLAVEd G_Opt
                                        { NumTc(($2,$4,$6,$8),$10) }
  | CORCi ENT COMA ENT CORCd LLAVEi Lista_ent_gra LLAVEd
                                        { CBorc         ($2,$4,$7) }
  | CADENA G_Opt                        { Textc            ($1,$2) }
;
Lista_ent_gra:
  ENT BARRA REAL                        { [($1,$3)]                }
  | Lista_ent_gra COMA ENT BARRA REAL   { $1 @ [($3,$5)]           }
;

T_Opt:
  Tipador                               { $1                       }
  | /* nada */                          { Tipo("BOOLEANO");        }
;
E_Opt:
  LLAVEi ENT LLAVEd                     { $2                       }
  | /* nada */                          { 0                        }
;
G_Opt:
  PARENi REAL PARENd                    { $2                       }
  | /* nada */                          { neutro                   }
;
Opt_Anti:
  ANTI Lista_pares                      { $2                       }
  | /* nada */                          { []                       }
;
Lista_pares:
  ID COMA ID DOSPTOS G_Opt              { [($1,$3,$5)]             }
  | Lista_pares ID COMA ID DOSPTOS G_Opt{ $1 @ [($2,$4,$6)]        }
;
%%
```

Fichero: algseman.mli

```
open Gramat;;
open Estrat;;
exception Not_found1 of string;;
type ind_borr = float * estra

type location = int
type tipo =
    | NumBt
    | BoolBt
    | RealBt
    | NumTt
    | CBort
    | Textt
and valor_Expresable =
    | Errexpr
    | Exprval of valor_Almacenable

and abstraccion =
    | Funcd of func
    | Procd of proc

and valor_Denotable =
    | Const  of constante
    | Abst   of abstraccion
    | Loc    of location
    | Record of amb * pares list
    | Tipod  of tipolin
    | Errvalue
and func = lista_argums -> ind_borr -> state ->
           (valor_Expresable * poststate) list
and proc = lista_argums -> ind_borr -> state -> respuesta
and tipolin = state -> valor_Denotable * poststate
and alm = (location * valor_Almacenable) list * location
and acc_alm_value =
    | OKacces of valor_Almacenable
    | Erracces
and valor_Almacenable =
    | NumB  of int   * grado
    | BoolB of bool  * grado
    | RealB of float * grado
    | NumT  of borr  * grado
    | CBor  of int * int * (int * grado) list
    | Text  of string * grado
    | Nulo  of tipo
and amb =  (identif * valor_Denotable) list
and lista_argums = valor_Expresable list
and state = alm * sal
and poststate =
    | OK   of state
    | Err  of state
    | Abo  of state
and respuesta = poststate list

(* Diseño en página 127 *)

and sal = string list
type outesc_val  = constante
and lista_params = identifs

(* Diseño en página 124 *)
```

```
val primera_loc    : location
val siguiente_loc  : location -> location
val igual_loc      : location -> location -> bool
val menorque_loc   : location -> location -> bool
val vac_alm        : alm
val acc_alm        : location -> alm -> acc_alm_value
val mod_alm        : location -> valor_Almacenable -> state -> poststate
val mod_alm_g      : location -> valor_Almacenable -> ind_borr -> state
                      -> poststate
val mar_loc        : alm -> location
val asi_loc        : state -> location * poststate
val des_loc        : location -> state -> poststate
val des_loc0       : location -> respuesta -> respuesta
val tipo_stv       : valor_Almacenable -> tipo
val vac_amb        : amb
val acc_amb        : identif -> amb -> valor_Denotable
val mod_amb        : identif -> valor_Denotable -> amb -> amb
val check_expr     : (state -> valor_Expresable * poststate)
                      -> (poststate -> valor_Expresable * poststate)
val ext_Expr       : (state -> (valor_Expresable * poststate))
                      -> (respuesta -> (valor_Expresable * poststate) list)
val check_errexpr  : (state -> valor_Expresable * poststate) ->
                      (valor_Almacenable -> state ->
                      valor_Expresable * poststate) ->
                      (state -> valor_Expresable * poststate)
val vac_sal        : sal
val esc_val        : outesc_val * sal -> sal
val put_message    : string * sal -> sal
val put_menserr    : string * sal -> sal
val return         : state -> poststate
val abort          : state -> poststate
val signalerr      : state -> poststate
val ext            : (state -> respuesta) -> (respuesta -> respuesta)
val check1         : (state -> poststate) -> (poststate -> poststate)
val check_alloc    : (state -> amb * poststate) ->
                      (poststate -> amb * poststate)
val state_of_post  : poststate -> state
val verifica_args  : identif -> lista_argums -> state -> poststate
val liga_params    : identif -> lista_params -> lista_argums -> amb ->
                      state -> poststate
val copy_state     : state -> state
val compru         : poststate -> bool
;;
```

Fichero: algseman.ml

```
open Gramat;;
open Estrat;;

exception Not_found1 of string;;
type ind_borr = float * estra

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |    DEFINICION ALGEBRAS SEMANTICAS
 |    ============================
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |  VIII.- Posiciones de Memoria
 |  ============================
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

type location = int
type tipo =
    | NumBt
    | BoolBt
    | RealBt
    | NumTt
    | CBort
    | Textt

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |  IX.- Valores Expresables
 |  ========================
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Diseño en página 124 *)

and valor_Expresable =
    | Errexpr
    | Exprval of valor_Almacenable

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |  X.- Valores Denotables
 |  ======================
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Diseño en página 124 *)

and abstraccion =
    | Funcd of func
    | Procd of proc

and valor_Denotable =
    | Const  of constante
    | Abst   of abstraccion
    | Loc    of location
    | Record of amb * pares list
    | Tipod  of tipolin
    | Errvalue

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |  XI.- Funciones
 |  ==============
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
```

```
(* Diseño en página 125 *)

and func = lista_argums -> ind_borr -> state ->
           (valor_Expresable * poststate) list

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |  XII.- Procedimientos
 |  ====================
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

and proc = lista_argums -> ind_borr -> state -> respuesta
(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |  XIV.- Valores Linguisticos
 |  ==========================
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

and tipolin = state -> valor_Denotable * poststate

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |  XVII.- Almacen
 |  ==============
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Diseño en página 126 *)

and alm = (location * valor_Almacenable) list * location
and acc_alm_value =
    | OKacces of valor_Almacenable
    | Erracces

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |  XVI.- Valores Almacenables
 |  ==========================
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
and valor_Almacenable =
    | NumB  of int   * grado
    | BoolB of bool  * grado
    | RealB of float * grado
    | NumT  of borr  * grado
    | CBor  of int * int * (int * grado) list
    | Text  of string * grado
    | Nulo  of tipo

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |  XV.- Ambitos
 |  ============
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Diseño en página 125 *)

and amb =  (identif * valor_Denotable) list

and lista_argums = valor_Expresable list
and state = alm * sal
and poststate =
    | OK   of state
    | Err  of state
    | Abo  of state
and lstate = state list
and respuesta = poststate list
and sal = string list
```

```
type outesc_val  = constante
and lista_params = identifs

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |    DEFINICION DE LAS OPERACIONES DE LAS ALGEBRAS SEMANTICAS
 |    ======================================================
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)


(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |    VIII.- Operaciones sobre posiciones de memoria
 |    ============================================
 |    - primera_loc
 |    - siguiente_loc
 |    - menorque_loc
 |    - igual_loc
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Diseño en página 124 *)

let primera_loc  = (0 : location) ;;
let siguiente_loc l = (((l:location)+1):location) ;;
let menorque_loc l1 l2 = (l1<l2) ;;
let igual_loc l1 l2 = (l1=l2) ;;

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |    XV.- Operaciones sobre el ambito
 |    ================================
 |    - vac_amb
 |    - acc_amb
 |    - mod_amb
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Diseño en página 125 *)

(*   vac_amb: amb                                                        *)
let vac_amb  = (([]): amb)

(*   acc_amb: identif -> amb -> valor_Denotable                          *)
and acc_amb = fun id m ->
           try
            List.assoc id m
           with
          Not_found -> raise(Not_found1 id)

(*  mod_amb: identif -> valor_Denotable -> amb -> amb                    *)
and mod_amb = fun id d e -> (id,d)::e
;;

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |    Comprobaciones sobre valores que puede denotar una expresion
 |    ===========================================================
 |    - check_expr
 |    - check_errexpr
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(*   check_expr: (state -> valor_Expresable * poststate) ->
                  (poststate -> valor_Expresable * poststate)            *)
let check_expr f = function z ->
  match z with
    | OK(a)   -> (f a)
    | Err(a)  -> (Errexpr,z)
```

```
    | Abo(a)  -> (Errexpr,z)
;;
(* check_errexpr: (state -> valor_Expresable * poststate) ->
            (valor_Almacenable -> state -> valor_Expresable * poststate) ->
            (state -> valor_Expresable * poststate)                          *)
let check_errexpr f1 f2 =
      function (a:state) ->
        match (f1 a) with
           | (Exprval(v),z) -> (check_expr (f2 v))(z)
           | (Errexpr,z)    -> (Errexpr,z)
;;

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |    Operaciones sobre el Poststate
 |    ==============================
 |    - copy
 |    - copy_state
 |    - return
 |    - signalerr
 |    - abort
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

let rec copy = function
  | [] -> []
  | x::l -> x:: copy l
;;
let copy_state  = function (s,o) -> let (st,lo) = s in
    ((copy st,lo),copy o)
;;

(*   return: state -> poststate                                           *)
let return = function a -> OK(a)

(*   abort: state -> poststate                                            *)
and  abort= function a -> Abo(a)

(*   signalerr: state -> poststate                                        *)
and signalerr = function a -> Err(a)

(*   check1: (state -> respuesta) -> (respuesta -> poststate)             *)
and check1 f = function z ->
  match z with
    | OK(a) -> (f a)
    | Err(a) -> z
    | Abo(a) -> z
;;

let compru = function z ->
  match z with
    | OK(_) -> true
    | Err(_) -> false
    | Abo(_) -> false
;;

let rec ext f = function (z:respuesta) ->
  match z with
    | [] -> ([]:respuesta)
    | x :: l ->
       (match x with
          | OK(a) -> (f a) @ (ext f l)
          | Err(a) -> [Err(a)] @ (ext f l)
          | Abo(a) -> [Abo(a)] @ (ext f l)
```

```
      )

and ext_Expr f = function (z:respuesta) ->
  match z with
  | [] -> []
  | x :: l ->
     (match x with
        | OK(a) -> [(check_expr f) (return a)] @ ext_Expr f l
        | b -> [ (check_expr f) (b) ] @ (ext_Expr f l)
     )


(*   check_alloc: (state -> amb*poststate) -> (poststate -> amb*poststate) *)
and check_alloc (f:(state -> amb * poststate)) =
        function z ->
            match z with
                OK(a) -> (f a)
                | Err(a) -> (vac_amb,Err(a))
                | Abo(a) -> (vac_amb,Abo(a))
(*   state_of_post: poststate -> state                                      *)
and state_of_post = function z ->
            match z with
                | OK(a)   -> a
                | Err(a)  -> a
                | Abo(a)  -> a
;;

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |    XVII.- Operaciones sobre el almacen
 |    ==================================
 |    - vac_alm
 |    - acc_alm
 |    - mar_loc
 |    - asi_loc
 |    - des_loc
 |    - mod_alm
 |    - mod_alm_g
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Diseño en página 126 *)

let vac_alm = ([],primera_loc)

(*   acc_alm : location -> alm -> acc_alm_value                             *)
and acc_alm = fun (l:location) ((map_s,top):alm) ->
           try
            OKacces(List.assoc l map_s)
           with
          Not_found -> raise(Not_found1 " en almacen ")
;;
let mar_loc = function (m,top) -> top
;;
let asi_loc = function ((m,top),o) ->
      (top, (return ((m,siguiente_loc(top)),o)))
;;
let des_loc = fun l ((m,top),o) ->
      return ((m,l),o)
;;

let rec des_loc0 l = function
  [] -> []
  | a :: b ->
```

```
 (match a with
       | OK(x) -> let k = des_loc l x in
           [ k ] @ (des_loc0 l b)
       | x  -> [x] @ (des_loc0 l b)
     )
;;
(*   mod_alm: location -> valor_Almacenable -> state -> poststate          *)
let mod_alm = fun (l:location) v (((m,ci),o)) ->
  if ((menorque_loc (l) (ci))) then
      return((((l,v)::m),ci),o)
  else
      signalerr ((m,ci),o)
;;
(* mod_alm_g: location -> valor_Almacenable -> state -> poststate          *)
let mod_alm_g = fun (l:location) (v:valor_Almacenable) i
                (((m,ci),o):state) ->
   if ((menorque_loc (l) (ci))) then
       let (g,est) = i in
       let v1 =
         (match v with
            | NumB(v2,g1)  -> NumB(v2,(tnorma(g,g1) est))
            | BoolB(v2,g1) -> BoolB(v2,(tnorma(g,g1) est))
            | RealB(v2,g1) -> RealB(v2,(tnorma(g,g1) est))
            | NumT(v2,g1)  -> NumT(v2,(tnorma(g,g1) est))
            | Text(v2,g1)  -> Text(v2,(tnorma(g,g1) est))
            | CBor(a1,a2,b) -> CBor(a1,a2,b)
         ) in return((((l,v1)::m),ci),o)
   else
      signalerr ((m,ci),o)
;;
(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |    XVIII.- Operaciones sobre buffer de salida
 |    ========================================
 |    - vac_sal
 |    - esc_val
 |    - put_message
 |    - put_menserr
 |    - put_Message
 |    - put_Menserr
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Diseño en página 127 *)

let rec plista = function
   [] -> ""
   | (x,a):: l -> let v= (if l==[] then "" else ", ") in
           string_of_int(x)^"|"^string_of_float(a)^v^(plista l)
;;
(*   vac_sal: sal                                                          *)
let vac_sal = ([]:sal)
;;
(*   esc_val: outesc_val * sal -> sal                                      *)
let esc_val = function (w,o) ->
      ( match w with
          | NumBc(n,r) ->
            let h=string_of_int(n)^"(" in
            let h1=h^string_of_float(r)^")" in h1::o
          | BoolBc(b,r) ->
            let str=(if b then "CIERTO" else "FALSO") in
            let str1=str^"("^string_of_float(r)^")" in str1::o
          | NumTc((b1,b2,b3,b4),r) ->
            let h1 = "{"^string_of_float(b1) in
```

```
            let h2 = ", "^string_of_float(b2) in
            let h3 = ", "^string_of_float(b3) in
            let h4 = h1^h2^h3^", "^string_of_float(b4)^"}(" in
            let h5 = h4^string_of_float(r)^")" in h5::o
          | RealBc(b,r) ->
            let h = string_of_float(b) in
            let h1 = h^"("^string_of_float(r)^")" in h1::o
          | CBorc(r1,r2,l) ->
            let h1 = "["^string_of_int(r1)^", " in
            let h2 = h1^string_of_int(r2)^"]" in
            let h3 = h2^"{"^(plista l)^"}" in h3::o
          | Textc(n,r) ->
            let h=n^"(" in
            let h1=h^string_of_float(r)^")" in h1::o
     )

(*   put_message: string * sal -> sal                                    *)
and put_message = function ((str:string),(o:sal)) -> str::o
;;
(*   put_menserr: string * sal -> sal                                    *)
let put_menserr = function (str,o) ->
      put_message ((">> Error "^str), o)
;;
let rec put_Message st = function
  | [] -> ([]:lstate)
  | x::y ->
    let (s,o) = x in
    let o1 = put_message(st,o) in
    [(s,o1)] @ (put_Message st) y
;;
let put_Menserr st = function a -> put_Message ("> Error "^st) a
;;

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |    Operaciones sobre los tipos
 |    ==========================
 |    - tipo_stv
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(*   tipo_stv: valor_Almacenable ->tipo                                  *)
let tipo_stv = function
            | NumB(_)  -> NumBt
            | BoolB(_) -> BoolBt
            | RealB(_) -> RealBt
            | NumT(_)  -> NumTt
            | CBor(_)  -> CBort
            | Text(_)  -> Textt
            | Nulo(t)  -> t

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |    Operaciones sobre listas de argumentos
 |    ====================================
 |    - verifica_args
 |    - busca_argm_mal
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(*   verifica_args: identif -> lista_argums -> state -> poststate        *)
(*   busca_argum_mal: int -> lista_argums -> state -> poststate          *)
let verifica_args i =
   let rec busca_argum_mal n g (s,o) =
      if g=[] then
         return (s,o)
```

```
      else
         if List.hd(g) = Errexpr then
           let o1 = put_menserr("en argumento ",o) in
           let o2 = put_menserr(string_of_int(n),o1) in
           let o3 = put_message(" aplicado a `"^i^"`\n",o2)
           in signalerr (s,o3)
         else
           busca_argum_mal (n + 1) (List.tl g) (s,o)
   in busca_argum_mal(1)
;;

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |    Operaciones sobre listas de parametros
 |    ====================================
 |    - liga_params
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Diseño en página 129 *)

(*   liga_params: identif -> lista_params -> lista_argums
                  -> amb -> state -> poststate                              *)
let liga_params i =
let rec liga_params_rec n p g e (s,o) =
  if (p=[] or g=[]) then
      ( if (p=[] & g=[]) then
          return (s,o)
        else
          let h="del numero de argumentos aplicados a `"^i^"`\n"
          in signalerr (s,put_menserr(h,o))
      )
  else
    check1(liga_params_rec (n+1) (List.tl p) (List.tl g) e)
    ( match (acc_amb (List.hd p) e) with
      | Loc(l) ->
        ( match (acc_alm l s) with
           | OKacces(v) ->
             let t = tipo_stv(v) in
             let x = List.hd(g) in
             ( match x with
                | Exprval(v1) ->
                   if t = tipo_stv(v1) then
                     (mod_alm l v1 (s,o))
                   else
                     let t1 = tipo_stv(v1) in
                     let o1 = put_menserr(": en el tipo del argumento ",o) in
                     let o2 = put_menserr(string_of_int(n),o1) in
                     let h = " aplicado a `"^i^"` " in
                     let o3 = put_message(h,o2)
                     in signalerr (s,o3)
                | Errexpr -> signalerr (s,put_menserr("en argumento\n",o))
             )
           | Erracces -> signalerr (s,put_menserr("de memoria\n",o))
        )
      | _ -> signalerr (s,put_menserr(": parametro no variable\n",o))
    )
  in liga_params_rec 1
;;
```

Fichero: exvaluac.mli

```
val oPval_int : string -> int -> int -> int
val oPval_inflo : float -> Algseman.valor_Almacenable -> float
val oPval_bool : string -> bool -> bool -> bool
val maxr : 'a -> 'a -> 'a
val minr : 'a -> 'a -> 'a
val resta : float -> float -> float
val abs : float -> float
val eli_com : 'a * float -> ('a * float) list -> ('a * float) list
val aux1_elimina : ('a * float) list -> ('a * float) list
val eli_int : 'a * 'b -> ('a * 'b) list -> ('a * 'b) list
val aux_elimina : ('a * 'b) list -> ('a * 'b) list
val eli_pro : 'a * float -> ('a * float) list -> ('a * float) list
val aux2_elimina : ('a * float) list -> ('a * float) list
val eli_uni : 'a * 'b -> ('a * 'b) list -> ('a * 'b) list
val inserta1 :
  ('a -> 'a -> bool) -> 'a * 'b -> ('a * 'b) list -> ('a * 'b) list
val ord_inser1 : ('a -> 'a -> bool) -> ('a * 'b) list -> ('a * 'b) list
val llama_eliuni : ('a * 'b) list -> ('a * 'b) list
val llama_eliint : ('a * 'b) list -> ('a * 'b) list
val llama_elipro : ('a * float) list -> ('a * float) list
val procar1 : 'a -> ('b * 'a) list -> 'a
val procar : int -> ('a * 'b) list -> ('c * 'b) list -> (int * 'b) list
val oPval_cbor :
  string -> (int * 'a) list -> (int * 'a) list -> (int * 'a) list
val oPval_cborpro :
  ('a * float) list -> ('a * float) list -> ('a * float) list
val llama_elicom : ('a * float) list -> ('a * float) list
val construye : int -> int -> (int * float) list
val comple : int -> int -> (int * float) list -> (int * float) list
val oPval_rel : string -> 'a -> 'a -> bool
val oPval_tex : string -> 'a -> 'a -> bool
val oPval_rel_flo : string -> 'a -> 'a -> bool
val oPval_flo : string -> float -> float -> float
val oPval_in : 'a -> ('a * float) list -> bool * float
val oPval_bor :
  string ->
  float * float * float * float ->
  float * float * float * float -> float * float * float * float
val oPval_borint :
  string ->
  float * float * float * float -> float -> float * float * float * float
val eval :
  Gramat.expresion ->
  Algseman.amb ->
  float * Estrat.estra ->
  Algseman.state -> Algseman.valor_Expresable * Algseman.poststate
val eval_bin :
  Gramat.identif * Gramat.expresion * Gramat.expresion ->
  Algseman.amb ->
  float * Estrat.estra ->
  Algseman.state -> Algseman.valor_Expresable * Algseman.poststate
val eval_error :
  Algseman.amb ->
  float * Estrat.estra ->
  Algseman.state -> Algseman.valor_Expresable * Algseman.poststate
val eval_una :
  Gramat.identif * Gramat.expresion ->
  Algseman.amb ->
  float * Estrat.estra ->
  Algseman.state -> Algseman.valor_Expresable * Algseman.poststate
```

```
val eval_grad :
  Gramat.expresion ->
  Algseman.amb ->
  float * Estrat.estra ->
  Algseman.state -> Algseman.valor_Expresable * Algseman.poststate
val eval_r :
  Gramat.expresion ->
  Algseman.amb ->
  float * Estrat.estra ->
  Algseman.state -> Algseman.valor_Expresable * Algseman.poststate
val eval_i :
  Gramat.expresion ->
  Algseman.amb ->
  float * Estrat.estra ->
  Algseman.state -> Algseman.valor_Expresable * Algseman.poststate
val eval_cte :
  Gramat.constante ->
  Algseman.amb ->
  float * Estrat.estra ->
  Algseman.state -> Algseman.valor_Expresable * Algseman.poststate
val eval_reco :
  Gramat.identif list ->
  Algseman.amb ->
  float * Estrat.estra ->
  Algseman.state -> Algseman.valor_Expresable * Algseman.poststate
val eval_var :
  Gramat.identif ->
  Algseman.amb ->
  float * Estrat.estra ->
  Algseman.state -> Algseman.valor_Expresable * Algseman.poststate
val eval_varC :
  Gramat.identif * int ->
  Algseman.amb ->
  float * Estrat.estra ->
  Algseman.state -> Algseman.valor_Expresable * Algseman.poststate
val aSval :
  Gramat.argumentos ->
  Algseman.amb ->
  float * Estrat.estra ->
  Algseman.state -> Algseman.lista_argums * Algseman.poststate
val eval_apl :
  Gramat.identif * Gramat.argumentos ->
  Algseman.amb ->
  Algseman.ind_borr ->
  Algseman.alm * Algseman.sal ->
  (Algseman.valor_Expresable * Algseman.poststate) list
val busca_env :
  Algseman.amb -> Gramat.identif list -> Gramat.identif * Algseman.amb
val bus_env :
  Algseman.amb -> Gramat.identifi -> Gramat.identif * Algseman.amb
val busca_a :
  'a -> string -> (string * string * float) list -> string * 'a * float
val busca_env2 :
  Algseman.amb ->
  Gramat.pares list ->
  Gramat.identif list -> Gramat.identif * Algseman.amb * float
val bus_env2 :
  Algseman.amb -> Gramat.identifi -> Gramat.identif * Algseman.amb * float
val eval_apl2 :
  Gramat.identifi * Gramat.argumentos ->
  Algseman.amb ->
  float * Estrat.estra ->
```

```
  Algseman.alm * Algseman.sal ->
  (Algseman.valor_Expresable * Algseman.poststate) list
val eval_apl1 :
  Gramat.identifi * Gramat.argumentos ->
  Algseman.amb ->
  Algseman.ind_borr ->
  Algseman.alm * Algseman.sal ->
  (Algseman.valor_Expresable * Algseman.poststate) list
```

Fichero: exvaluac.ml

```
open Gramat;;
open Estrat;;
open Algseman;;


(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | Funciones de Valuacion
 | ======================
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
let oPval_int = function

(* Especificación formal en página 133 *)

  | "+"  -> ( + )
  | "-"  -> ( - )
  | "*"  -> ( * )
  | "/"  -> ( / )
;;

let oPval_inflo = fun nr v ->
let NumT((a1,a2,a3,a4),g) = v in
if nr <= a1 then 0.0
else if ((a1 < nr) & (nr < a2)) then (nr-.a1) /. (a2-.a1)
else if ((a2 <= nr) & (nr <= a3)) then 1.0
else if ((a3 < nr) & (nr < a4)) then (a4-.nr) /. (a4-.a3)
else 0.0
;;

let oPval_bool=function

(* Especificación formal en página 133 *)

  | "AND" -> (fun b1 b2 -> if b1 then b2 else false)
  | "OR"  -> (fun b1 b2 -> if b1 then true else b2 )
;;

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | Conjuntos borrosos union e interseccion
 + - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
let maxr = fun a b ->  if (a <= b) then b else a;;
let minr = fun a b ->  if (a <= b) then a else b;;
let resta = fun a b -> a -. b;;
let abs x = if x > 0.0 then x else -. x;;

let rec eli_com (ele1,ele2) = function
     [] -> []
   | (x1,x2):: resto as l ->
      if ele1<>x1
      then  (ele1,ele2) :: eli_com (x1,x2) resto
      else  if (abs(resta ele2 x2)) > 0.0
            then (ele1,(abs(resta ele2 x2))):: (aux1_elimina resto)
            else aux1_elimina resto

and aux1_elimina = function
     [] -> []
   | (x1,x2) :: l -> eli_com (x1,x2) l
;;
let rec eli_int (ele1,ele2) = function
     [] -> []
   | (x1,x2):: resto as l ->
```

```
      if ele1<>x1
      then  eli_int (x1,x2) resto
      else  (ele1,minr ele2 x2):: (aux_elimina resto)
and aux_elimina = function
     [] -> []
   | (x1,x2) :: l -> eli_int (x1,x2) l
;;
let rec eli_pro (ele1,ele2) = function
     [] -> []
   | (x1,x2):: resto as l ->
      if ele1<>x1
      then  eli_pro (x1,x2) resto
      else  (ele1, ele2 *. x2):: (aux2_elimina resto)
and aux2_elimina = function
     [] -> []
   | (x1,x2) :: l -> eli_pro (x1,x2) l
;;

let rec eli_uni (ele1,ele2) = function
     [] -> [(ele1,ele2)]
   | (x1,x2) :: resto as l ->
      if ele1=x1
      then eli_uni (ele1,maxr ele2 x2) resto
      else (ele1,ele2) :: (eli_uni (x1,x2) resto)
;;

let rec inserta1 f_ord (ele1, ele2) = function
     [] -> [(ele1,ele2)]
   | (x1,x2) :: resto as l ->
      if f_ord ele1 x1
      then (ele1,ele2) :: l
      else (x1,x2) :: (inserta1 f_ord (ele1,ele2) resto)
;;
let rec ord_inser1 f_ord1 = function
     [] -> []
   | x ::resto -> inserta1 f_ord1 x (ord_inser1 f_ord1 resto)
;;

let llama_eliuni = function
     [] -> []
   | x :: l -> eli_uni x l
;;

let llama_eliint = function
     [] -> []
   | x :: l -> eli_int x l
;;

let llama_elipro = function
     [] -> []
   | x :: l -> eli_pro x l
;;

let rec procar1 g = function
     [] -> g
   | (_,g1) :: l -> procar1 (min g g1) l
;;

let rec procar n a = function
     [] -> []
   | (_,g) :: l -> (n,(procar1 g a)) :: (procar (n+1) a l)
;;
```

```
let oPval_cbor=function

(* Especificación formal en página 133 *)

  | "UNI" -> (fun b1 b2 -> llama_eliuni (ord_inser1 (function x ->
              function y ->  x <= y) (b1 @ b2)))
  | "INT" -> (fun b1 b2 -> llama_eliint (ord_inser1 (function x ->
              function y ->  x <= y) (b1 @ b2)))
  | "PRO" -> (fun b1 b2 -> procar 1 b1 b2)
;;
let oPval_cborpro=(fun b1 b2 -> llama_elipro (ord_inser1 (function x ->
              function y -> x <= y) (b1 @ b2)))
;;
let llama_elicom = function
     [] -> []
   | x :: l -> eli_com x l
;;
let rec construye a b =
   if (a<=b) then (a,1.) :: construye (a+1) b
   else [(a,1.)]
;;

let comple a b = (function l -> llama_elicom (ord_inser1 (function x ->
              function y -> x <=y ) ((construye a b) @ l)))
;;

let oPval_rel = function

(* Especificación formal en página 133 *)

  | "<"  -> ( (<) )
  | ">"  -> ( (>) )
  | "<=" -> ( (<=))
  | ">=" -> ( (>=))
  | "="  -> ( (=) )
  | "!=" -> ( (<>))
;;

let oPval_tex = function
  | "="  -> (fun b c -> b = c )
  | "!=" -> (fun b c -> b != c )
;;
let oPval_rel_flo = function

(* Especificación formal en página 133 *)

  | "<"   -> (fun b c -> b < c )
  | ">"   -> (fun b c -> b > c )
  | "<="  -> (fun b c -> b <= c )
  | ">="  -> (fun b c -> b >= c )
  | "="   -> (fun b c -> b = c )
  | "!="  -> (fun b c -> b <> c )
;;
let oPval_flo = function

(* Especificación formal en página 133 *)

  | "+"  -> (fun b c -> b +. c )
  | "-"  -> (fun b c -> b -. c )
  | "*"  -> (fun b c -> b *. c )
  | "/"  -> (fun b c -> b /. c )
```

```
;;
let rec oPval_in b = function
    [] -> (false,1.0)
  | (a,x):: l -> if (a=b) then (true,x) else oPval_in b l
;;
let oPval_bor = function

(* Especificación formal en página 133 *)

  | "+"  -> (fun (b1,b2,b3,b4) (c1,c2,c3,c4) ->
    ( (b1 +. c1), (b2 +. c2), (b3 +. c3), (b4 +. c4)) )
  | "-"  -> (fun (b1,b2,b3,b4) (c1,c2,c3,c4) ->
    ( (b1 -. c4), (b2 -. c3), (b3 -. c2), (b4 -. c1)) )
  | "*"  ->(fun (b1,b2,b3,b4) (c1,c2,c3,c4) ->
    ( (b1 *. c1), (b2 *. c2), (b3 *. c3), (b4 *. c4)) )
  | "/"  ->(fun (b1,b2,b3,b4) (c1,c2,c3,c4) ->
    ( (b4 /. c1), (b3 /. c2), (b2 /. c3), (b1 /. c4)) )
;;
let oPval_borint = function

(* Especificación formal en página 133 *)

  | "+"  -> (fun (b1, b2, b3, b4) c ->
    ( (b1 +. c), (b2 +. c), (b3 +. c), (b4 +. c) ) )
  | "-"  -> (fun (b1, b2, b3, b4) c ->
    ( (b1 -. c), (b2 -. c), (b3 -. c), (b4 -. c) ) )
  | "*"  -> (fun (b1, b2, b3, b4) c ->
    ( (b1 *. c), (b2 *. c), (b3 *. c), (b4 *. c) ) )
  | "/"  -> (fun (b1, b2, b3, b4) c ->
    ( (b1 /. c), (b2 /. c), (b3 /. c), (b4 /. c) ) )
;;
(* Eval: expresion -> amb -> t_op -> state -> valor_Expresable * poststate *)
let rec eval = function
  | Id(Variable(i))     -> eval_var i
  | Id(VariableC(i,n))  -> eval_varC (i,n)
  | Constante(c)        -> eval_cte c
  | Op_binaria(i,e1,e2) -> eval_bin (i,e1,e2)
  | Op_unaria(i,e)      -> eval_una (i,e)
  | Id(VariableR(l))    -> eval_reco(l)
  | Grado(e)            -> eval_grad(e)
  | Op_r(e)             -> eval_r(e)
  | Op_i(e)             -> eval_i(e)
  | C_fuzzy(e1,e2)      -> eval_f(e1,e2)
  | _                   -> eval_error

and

eval_f (e1,e2) e (g1, est) =
      (check_errexpr
        ((eval(e1) e (g1,est)))
        (
          (function v ->
            ( if (tipo_stv(v)=NumBt) then
                ( check_errexpr
                  ((eval(e2) e (g1,est)))
                  (
                    (fun v1 (s,o) ->
                      ( if (tipo_stv(v1)=RealBt) then
                          (match v with NumB(n,g) ->
                             (match v1 with RealB(n1,g1) ->
                                (Exprval(CBor(0,0,[(n,n1)])),return (s,o)
                                )
```

```
                              )
                            )
                          else
                            let h=": segundo operando " in
                            let h1=h^" no es un real\n"
                          in
                        ( Errexpr,signalerr (s,put_menserr(h,o)))
                        )
                      )
                    )
                  else
                    (function (s,o) ->
                      let h=": primer operando " in
                      let h1=h^" no es un entero\n"
                      in (Errexpr,signalerr(s,put_menserr(h1,o)))
                    )
                )
              )
            )
          )
      )
and

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | Eval_bin: identif * expresion * expresion -> amb -> t_op -> state ->
 |           valor_Expresable * poststate
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Especificación formal en página 132 *)

eval_bin (i,e1,e2) e (g1,est) =
  (match i with
    | ("+" | "-" | "*") ->
      (check_errexpr
        ((eval(e1) e (g1,est)))
        (
          (function v ->
            (check_errexpr
              ((eval(e2) e (g1,est)))
              (
                (fun v1 (s,o) ->
                  (match v with
                    | NumB(n,g) ->
                      (match v1 with
                        | NumB(n1,g1) ->
                          ( Exprval(NumB(((oPval_int i) n n1),
                            tnorma(g,g1) est)), return (s,o)
                          )
                        | NumT(n1,g1) ->
                          (Exprval(NumT(((oPval_borint i) n1
                           (float(n))),tnorma(g,g1) est)), return (s,o)
                          )
                        | _ ->
                          let h=": operandos diferentes tipos 1 " in
                          (Errexpr,signalerr(s,put_menserr(h,o)))
                      )
                    | CBor(r1,r2,b) ->
                      (match v1 with
                        | CBor(t1,t2,m) ->
                           let mi=(min r1 t1) in let ma=(max r2 t2) in
                             ( Exprval(CBor(mi,ma,(oPval_cborpro
                               b m))), return (s,o)  )
```

```
                    )
                  | NumT(n,g) ->
                    (match v1 with
                      | NumT(n1,g1) ->
                        ( Exprval(NumT(((oPval_bor i) n n1),
                          tnorma(g,g1) est)), return (s,o)
                        )
                      | NumB(n1,g1) ->
                        ( Exprval(NumT(((oPval_borint i) n
                          (float(n1))),tnorma(g,g1) est)), return (s,o)
                        )
                      | RealB(n1,g1) ->
                        ( Exprval(NumT(((oPval_borint i) n n1),
                          tnorma(g,g1)est)), return (s,o)
                        )
                      | _ ->
                        let h=": operados diferentes tipos 2 " in
                        (Errexpr, signalerr(s,put_menserr(h,o)))
                    )
                  | RealB(n,g) ->
                    (match v1 with
                      | RealB(n1,g1) ->
                        ( Exprval(RealB(((oPval_flo i) n n1),
                          tnorma(g,g1)est)), return (s,o)
                        )
                      | NumT(n1,g1) ->
                        ( Exprval(NumT(((oPval_borint i) n1 n),
                          tnorma(g,g1)est)), return (s,o)
                        )
                      | _ ->
                        let h=": operados diferente tipo 3 " in
                        (Errexpr,signalerr(s,put_menserr(h,o)))
                    )
                )
              )
            )
          )
        )
      )
    )
  )
| (">"|"<"|">="|"<="|"="|"!=") ->
  (check_errexpr
    ((eval(e1) e (g1,est)))
    (
      (function v1 ->
        (check_errexpr
          ((eval(e2) e (g1,est)))
          (
            (fun v2 (s,o) ->
              (match v1 with
                | NumB(n1,g1) ->
                  (match v2 with
                    | NumB(n2,g2) ->
                      (Exprval(BoolB(((oPval_rel i) n1 n2),
                        tnorma(g1,g2)est)), return (s,o)
                      )
                    | _ ->
                      let h=": operados diferente tipo 1 " in
                      (Errexpr,signalerr(s,put_menserr(h,o)))
                  )
                | Text(n1,g1) ->
                  (match v2 with
```

```
                    | Text(n2,g2) ->
                      (Exprval(BoolB(((oPval_tex i) n1 n2),
                        tnorma(g1,g2)est)), return (s,o)
                      )
                    | _ ->
                      let h=": operados diferente tipo 1 " in
                      (Errexpr,signalerr(s,put_menserr(h,o)))
                  )
                | RealB(n1,g1) ->
                  (match v2 with
                    | RealB(n2,g2) ->
                      ( Exprval(BoolB(((oPval_rel_flo i) n1 n2),
                        tnorma(g1,g2)est)), return (s,o)
                      )
                    | _ ->
                      let h=": operandos tipos diferentes3 " in
                      (Errexpr,signalerr(s,put_menserr(h,o)))
                   )
                | _ ->
                  let h=": operados diferente tipo 3bis " in
                  ( Errexpr,signalerr (s,put_menserr(h,o)))
              )
        ) ) )
  ) ) )

| ("/") ->
  (check_errexpr
    ((eval(e1) e (g1,est)))
    (
      (function v ->
        ( if (tipo_stv(v)=RealBt) then
            ( check_errexpr
              ((eval(e2) e (g1,est)))
              (
                (fun v1 (s,o) ->
                  ( if (tipo_stv(v1)=RealBt) then
                      (match v with RealB(n,g) ->
                         (match v1 with RealB(n1,g1) ->
                            ( if n1= 0.0 then
                                let h=": division por cero "
                                in ( Errexpr,signalerr (s,put_menserr(h,o)))
                              else
                              (Exprval(RealB(((oPval_flo i)n n1),
                                 tnorma(g,g1)est)), return (s,o)
                              )
                            )
                          )
                       )
                    else
                      let h=": segundo operando " in
                      let h1=h^" no es un real\n"
                    in
                  ( Errexpr,signalerr (s,put_menserr(h,o)))
                  )
                )
              )
            )
          else
            (function (s,o) ->
              let h=": primer operando " in
              let h1=h^" no es un real\n"
              in (Errexpr,signalerr(s,put_menserr(h1,o)))
```

```
                )
              )
            )
          )
        )

(* Especificación formal en página 134 *)

    | ("IN") ->
      (check_errexpr
        ((eval(e1) e (g1,est)))
        (
          (function v ->
            ( if (tipo_stv(v)=NumBt) then
                ( check_errexpr
                  ((eval(e2) e (g1,est)))
                  (
                    (fun v1 (s,o) ->
                      ( if (tipo_stv(v1)=CBort) then
                          match v with NumB(nn,gn) ->
                            match v1 with CBor(r1,r2,l) ->
                            let (boo2,gr) = (oPval_in nn) l in
                               (Exprval(BoolB(boo2,
                                tnorma(gr,g1)est)), return (s,o))
                        else
                          let h=": segundo operando" in
                          let h1=h^" no es un Conjunto borroso\n"
                          in (Errexpr,signalerr (s,put_menserr(h1,o)))
                      )
                    )
                  )
                )
              else ( if (tipo_stv(v)=RealBt) then
                       ( check_errexpr
                         ((eval(e2) e (g1,est)))
                         (
                           (fun v1 (s,o) ->
                             ( if (tipo_stv(v1)=NumTt) then
                                 match v with RealB(nn,gn) ->
                                   (Exprval(RealB((oPval_inflo nn v1),
                                    tnorma(gn,g1)est)), return (s,o))
                               else
                                 let h=": segundo operando" in
                                 let h1=h^" no es un Conjunto borroso\n"
                                 in (Errexpr,signalerr (s,put_menserr(h1,o)))
                             )
                           )
                         )
                       )
                     else
                       function (s,o) ->
                       let h=": primer operando" in
                       let h1=h^" no es un entero\n"
                       in (Errexpr,signalerr(s,put_menserr(h1,o)))
            )
          )
        )
      )
    | ("AND"|"OR") ->
      (check_errexpr
        ((eval(e1) e (g1,est)))
```

```
      (
        (function v ->
        ( if (tipo_stv(v)=BoolBt or tipo_stv(v)= RealBt) then
          (check_errexpr
            ((eval(e2) e (g1,est)))
            (
              (fun v1 (s,o) ->
                ( if (tipo_stv(v1)=BoolBt or tipo_stv(v)= RealBt)  then
                    (if (tipo_stv(v)=tipo_stv(v1)) then
                      (if tipo_stv(v1)=BoolBt then
                        (match v with BoolB(b,g) ->
                          (match v1 with BoolB(b1,g1) ->
                            (if i = "OR" then
                              ( Exprval(BoolB(((oPval_bool i) b b1),
                                tconorma(g,g1)est)), return (s,o)
                              )
                             else
                              ( Exprval(BoolB(((oPval_bool i) b b1),
                                tnorma(g,g1)est)), return (s,o)
                              )
                            )
                          )
                         )
                        else
                         (match v with RealB(b,g) ->
                           (match v1 with RealB(b1,g1) ->
                             (if i = "OR" then
                               (Exprval(RealB((minr b b1),g)),return(s,o))
                              else
                               (Exprval(RealB((maxr b b1),g)),return(s,o))
                             )
                           )
                         )
                       )
                      else
                       let h=": segundo operando " in
                       let h1=h^" no es un booleano\n"
                       in ( Errexpr,signalerr (s,put_menserr(h1,o)))
                    )
                   else
                     let h=": Operandos " in
                     let h1=h^" booleanos de tipo distinto\n"
                     in ( Errexpr,signalerr (s,put_menserr(h1,o)))
                )
              )
            )
          )
          else
            (function (s,o) ->
              let h=": primer operando " in
              let h1=h^" no es un booleano\n"
              in ( Errexpr,(signalerr (s,put_menserr(h1,o))) )
            )
        )
))

 | ("UNI"|"INT"|"PRO") ->
   (check_errexpr
     ((eval(e1) e (g1,est)))
     (
       (function v ->
       ( if tipo_stv(v)=CBort then
```

```
            (check_errexpr
              ((eval(e2) e (g1,est)))
              (
                (fun v1 (s,o) ->
                  ( if tipo_stv(v1)=CBort then
                      (match v with CBor(r1,r2,b) ->
                        (match v1 with CBor(t1,t2,b1) ->
                           let mi=(min r1 t1) in let ma=(max r2 t2) in
                         (if i = "UNI" then
                          ( Exprval(CBor(mi,ma,((oPval_cbor i)
                             b b1))), return (s,o)
                          )
                          else
                           (if i = "PRO" then
                            ( Exprval(CBor(1,(t2-t1)*(r2-r1),((oPval_cbor i)
                                b b1))), return (s,o)
                            )
                            else
                            ( Exprval(CBor(mi,ma,((oPval_cbor i)
                               b b1))), return (s,o)
                            )
                           )
                          )
                        )
                      )
                    else
                      let h=": segundo operando " in
                      let h1=h^" no es Conjunto borroso\n"
                      in ( Errexpr,signalerr (s,put_menserr(h1,o)))
                  )
                )
              )
              else
                (function (s,o) ->
                  let h=": primer operando " in
                  let h1=h^" no es un conjunto borroso\n"
                  in ( Errexpr,(signalerr (s,put_menserr(h1,o))) )
                )
          )
   )))

)
and
eval_error = (fun e0 (g1,est) (s,o) ->(print_string("ERROR 0"));
                         let h=": _{x} no asinable funcion " in
                         (Errexpr,signalerr(s,put_menserr(h,o))))
and
(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | Eval_una: identif * expresion -> amb -> t_op -> state ->
 |           valor_Expresable * poststate
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Especificación formal en página 132 *)

eval_una (i,e) e0  (g1,est)=
    match i with
    ("-") ->
        (check_errexpr
        ((eval(e) e0 (g1,est)))
        ((function v ->
          (if tipo_stv(v)=NumBt then
```

```
            (match v with NumB(n,g) ->
               (function a -> ( Exprval(NumB(-1*n,nega(g) est)), return a ))
            )
          else
            (function (s,o) ->
               let h=": signo menos aplicado expresion no entera"
               in ( Errexpr,signalerr (s,put_menserr(h,o)))
            )
        )
       )))

(* Especificación formal en página 132 *)

  | ("NOT") ->
       (check_errexpr
       ((eval(e) e0 (g1,est)))
       ((function v ->
         (if tipo_stv(v)=BoolBt then
            (match v with BoolB(b,g) ->   (* OJO OJO LA negacion *)
               (function a -> (Exprval(BoolB(not b, nega(g) est)),return a)
               )
            )
          else
           (if tipo_stv(v)=RealBt then
              (match v with RealB(b,g) ->   (* OJO OJO LA negacion *)
                 (function a -> (Exprval(RealB((1.-.b), g)),return a)
                 )
              )
            else
              (function (s,o) ->
                 let h=": operador 'not' aplicado a no booleano"
                 in ( Errexpr,signalerr (s,put_menserr(h,o)))
              )
           )
         )
       )))


(* Especificación formal en página 132 *)

  | ("COM") ->
       (check_errexpr
       ((eval(e) e0 (g1,est)))
       ((function v ->
         (if tipo_stv(v)=CBort then
            (match v with CBor(ref1,ref2,b) ->
               (function a -> (Exprval(CBor(ref1,ref2,
                comple ref1 ref2 b)),return a)
               )
            )
          else
            (function (s,o) ->
               let h=": operador 'C' aplicado a no Conjunto borroso"
               in ( Errexpr,signalerr (s,put_menserr(h,o)))
            )
         )
       )))

and
eval_grad (e) e0 (g1, est) =
    (check_errexpr
    ((eval(e) e0 (g1,est)))
```

```
    ((function v ->
      (match tipo_stv(v) with
        NumBt ->
         (match v with NumB(n,g) ->
            (function a -> (Exprval(RealB(g,est.neutro)),return a))
         )
      | BoolBt ->
         (match v with BoolB(b,g) ->
            (function a -> (Exprval(RealB(g,est.neutro)),return a))
         )
      | RealBt ->
         (match v with RealB(b,g) ->
            (function a -> (Exprval(RealB(g,est.neutro)),return a))
         )
      | NumTt ->
         (match v with NumT(b,g) ->
            (function a -> (Exprval(RealB(g,est.neutro)),return a))
         )
      )
    )))

and
eval_r (e) e0  (g1,est)=
    (check_errexpr
    ((eval(e) e0 (g1,est)))
    ((function v ->
      (match tipo_stv(v) with
        NumBt ->
         (match v with NumB(n,g) ->
            (function a -> (Exprval(RealB(float(n),g)),return a))
         )
      | _ ->let h=" tipo operando incorrecto 1" in
            (function (s,o) ->
               ( Errexpr, signalerr(s,put_menserr(h,o)) ))
            )
    )))

and
eval_i (e) e0 (g1,est) =
    (check_errexpr
    ((eval(e) e0 (g1,est)))
    ((function v ->
      (match tipo_stv(v) with
        RealBt ->
         (match v with RealB(n,g) ->
            (function a -> (Exprval(NumB(truncate(n),g)),return a))
         )
      | _ ->let h=" tipo operando incorrecto 2" in
            (function (s,o) ->
              ( Errexpr, signalerr(s,put_menserr(h,o)) ))
            )
    )))
and

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | Eval_cte: constante -> amb -> t_op -> state ->valor_Expresable*poststate
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Especificación formal en página 132 *)

eval_cte c e  (g1,est)a =
   match c with
```

```
    | NumBc((x0,x1))  -> ( Exprval(NumB((x0,x1))), return a )
    | BoolBc((x0,x1)) -> ( Exprval(BoolB((x0,x1))), return a )
    | RealBc((x0,x1)) -> ( Exprval(RealB((x0,x1))), return a )
    | NumTc((x0,x1))  -> ( Exprval(NumT((x0,x1))), return a )
    | CBorc(x0,x1,b)  -> ( Exprval(CBor((x0,x1,b))), return a )
    | Textc(x0,x1)    -> ( Exprval(Text((x0,x1))), return a)
and
(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | Eval_I.E: identif -> amb -> t_op -> state ->valor_Expresable*poststate
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Especificación formal en página 132 *)

eval_reco lI e ind a =
    match lI with
    [] ->   let (s,o) = a in
        let h=": Algo va mal en ExpreR"
        in ( Errexpr, signalerr (s,put_menserr(h,o)))
    | c:: b -> (match (acc_amb c e) with
        | Loc(l) -> eval_var c e ind a
        | Record(d,p) -> eval_reco b d ind a
        | _ -> let (s,o) = a in
        let h=": Algo va mal en ExpreR"
        in ( Errexpr, signalerr (s,put_menserr(h,o)))
           )
and
(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | Eval_var: identif -> amb -> t_op -> state ->valor_Expresable*poststate
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
eval_var i e (g1,est) (s,o) =

(* Especificación formal en página 132 *)

   match (acc_amb i e) with
     Loc(l) ->
        (match (acc_alm l s) with
           OKacces(v) ->
            (match v with
               Nulo(t) ->
                 let h=": variable `"^i^"` sin inicializar\n"
                 in ( Errexpr, signalerr (s,put_menserr(h,o)))
             | _ -> ( Exprval(v), return (s,o))
            )
         | Erracces ->
            let h="de memoria en el acceso a `"^i^"`\n"
            in ( Errexpr, signalerr (s,put_menserr(h,o)))
        )
    | Const(c) ->
        (match c with
         | NumBc((x0,x1))   ->
             ( Exprval(NumB((x0,x1))), return (s,o))
         | BoolBc((x0,x1))  ->
             ( Exprval(BoolB((x0,x1))), return (s,o))
         | RealBc((x0,x1))  ->
             ( Exprval(RealB((x0,x1))), return (s,o))
         | Textc((x0,x1))  ->
             ( Exprval(Text((x0,x1))), return (s,o))
         | NumTc((x0,x1))  ->
             ( Exprval(NumT((x0,x1))), return (s,o))
         | CBorc((x0,x1,b))  ->
             ( Exprval(CBor((x0,x1,b))), return (s,o))
        )
```

```
    | Errvalue ->
        let h=": variable `"^i^"` no existe\n"
        in ( Errexpr,signalerr (s,put_menserr(h,o)))
    | Abst(k)  ->
        let h=": identificador `"^i^"` es abstraccion\n"
        in ( Errexpr,signalerr (s,put_menserr(h,o)))

and
eval_varC (i, n) e (g1,est) (s,o) =
   match (acc_amb i e) with
     | Loc(l) ->
        (match (acc_alm l s) with
          | OKacces(v) ->
            (match v with
             | Nulo(t) ->
                 let h=": variable `"^i^"` sin inicializar\n"
                 in ( Errexpr,signalerr (s,put_menserr(h,o)))
             | NumT((t1,t2,t3,t4),g) ->
                ( match n with
                  | 1 -> ( Exprval(RealB(t1,g)), return (s,o))
                  | 2 -> ( Exprval(RealB(t2,g)), return (s,o))
                  | 3 -> ( Exprval(RealB(t3,g)), return (s,o))
                  | 4 -> ( Exprval(RealB(t4,g)), return (s,o))
                )
             | _ ->
                  let h=": variable `"^i^"` no adecuada "
                  in ( Errexpr,signalerr (s,put_menserr(h,o)))
            )
          | Erracces ->
              let h="de memoria en el acceso a `"^i^"`\n"
              in ( Errexpr,signalerr (s,put_menserr(h,o)))
        )
     | Const(c) ->
        (match c with
         | NumTc((t1,t2,t3,t4),g) ->
            ( match n with
               | 1 -> ( Exprval(RealB(t1,g)), return (s,o))
               | 2 -> ( Exprval(RealB(t2,g)), return (s,o))
               | 3 -> ( Exprval(RealB(t3,g)), return (s,o))
               | 4 -> ( Exprval(RealB(t4,g)), return (s,o))
            )
         | _ ->
            let h=": constante `"^i^"` no adecuada "
            in ( Errexpr,signalerr (s,put_menserr(h,o)))
        )
     | _ ->
        let h=": variable `"^i^"` no existe\n"
        in ( Errexpr,signalerr (s,put_menserr(h,o)))

(*  ASval: argumentos -> amb -> indb * state ->
          lista_argums * indb * poststate                                  *)
and
aSval (lA:argumentos) (e:amb) ind a =
  let rec aSvalrec (lA:argumentos) (e:amb) a =
    (match lA with
        ([]:argumentos) ->
           ( ([]:lista_argums), return a)
     | ([e0]:argumentos) ->
          let (x,z)=(eval e0 e ind a)
          in ( [x], z )
     | ((e0::lE):argumentos) ->
         (match (eval e0 e ind a) with
```

```
            | (Exprval(v),z) ->
                let resto=(aSval lE e ind (state_of_post z))
                in ( (Exprval(v))::(fst resto), (snd resto) )
            | (Errexpr,z) ->
                ( [Errexpr], z)
          )
   ) in aSvalrec lA e a
;;
(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | Eval_apl: identif * argumentos -> amb -> t_op -> state
 |            -> (valor_Expresable * poststate) list
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

(* Especificación formal en página 132 *)


let rec
eval_apl (i,lA) e ind (s,o) =
   match (acc_amb i e) with
    | Loc(l) -> let h="  `"^i^"` de tipo"
          in [ ( Errexpr,signalerr (s,put_menserr(h,o))) ]
    | Abst(Funcd(f)) ->
        let (argg,z) = (aSval lA e ind (s,o)) in
        let z1 = (verifica_args i argg (state_of_post z)) in
        let z2 = (state_of_post z1) in
        let (s2,o2) = z2 in
        (f argg) ind (s2,o2)
    | Abst(Procd(q)) ->
        let h=": `"^i^"` es procedimiento"
        in [ ( Errexpr,signalerr (s,put_menserr(h,o))) ]
    | Const(c) ->
        let h=": `"^i^"` es una constante "
        in [ ( Errexpr,signalerr (s,put_menserr(h,o))) ]
    | Errvalue ->
        let h="  de valor"
        in [ ( Errexpr,signalerr (s,put_menserr(h,o))) ]
;;

let rec
busca_env e = function
    | [] ->("ERROR 1",e)
    | c:: b ->
      (match (acc_amb c e) with
        | Abst(Funcd(_)) -> (c, e)
        | Loc(_) -> (c, e)
        | Record(d,p) -> busca_env (d  @ e) b
        | _ -> ("ERROR 2",e)
      )
;;

let bus_env e = function
    | Variable(nv) ->
      (match (acc_amb nv e) with
        | Abst(Funcd(_)) -> (nv,e)
        | _ -> ("ERROR 3",e)
      )
    | VariableR(ln) -> busca_env e ln
    | _ -> ("ERROR 4",e)
;;
let rec
busca_a e c = function
    | [] -> ("ERRORi 5",e,0.)
```

```
    | x :: y ->
      let (n1,n2,g) = x in
      ( if n1 = c then
           (n2,e,g)
        else if n2 = c then
                (n1,e,g)
             else busca_a e c y
      )

let rec
busca_env2 e lp = function
    | [] ->("ERROR 6",e,0.)
    | c:: b ->
      (match (acc_amb c e) with
        | Abst(Funcd(_)) ->
             busca_a e c lp
        | Record(d,p) -> busca_env2 (d  @ e) p b
        | _ -> ("ERROR 7",e,0.)
      )
;;

let bus_env2 e = function
    | VariableR(ln) -> busca_env2 e [] ln
    | _ -> ("ERROR 8",e,0.)
;;

let
eval_apl2 (i,lA) e ind (s,o) =
   let (gi,est) = ind in
   let (i1,e1,g) = (bus_env2 e i) in
   eval_apl (i1,lA) e1 ((tnorma(gi,g) est), est) (s,o)
;;

let
eval_apl1 (i,lA) e ind (s,o) =
let (i1,e1) = (bus_env e i) in
eval_apl (i1,lA) e1 ind (s,o)
;;
```

Fichero: ffvaluac.mli

```
open Gramat;;
open Algseman;;
open Exvaluac;;

type declaracion_parametros = decl_vars
type declaracion_constantes = decl_consts
type declaracion_variables  = decl_vars

val p      : programa -> alm -> respuesta
val k      : bloque -> amb -> ind_borr -> state -> respuesta
val dval   : declaraciones -> amb -> state -> amb * poststate
val dFval  : decl_func -> amb -> state -> amb * poststate
val pSval  : declaracion_parametros -> lista_params
val dCSval : declaracion_constantes -> amb -> state -> amb * poststate
val dVSval : declaracion_variables -> amb -> state -> amb * poststate
val dPSval : declaracion_parametros -> amb -> state -> amb * poststate
val sval   : sentencia -> amb -> ind_borr -> state -> respuesta
val t      : tipador -> amb -> state -> valor_Denotable * poststate
;;
```

Fichero: ffvaluac.ml

```
open Gramat;;
open Estrat;;
open Algseman;;
open Exvaluac;;

type declaracion_parametros = decl_vars

type declaracion_constantes = decl_consts

type declaracion_variables = decl_vars

(* PSval: declaracion_parametros -> lista_params                          *)
let rec pSval = function
     | ([]:declaracion_parametros) -> ([]:lista_params)
     | [(i,t)] -> [i]
     | (i,t)::lIT -> i::pSval(lIT)
;;
(* dCSval: declaracion_constantes -> amb -> state -> amb * poststate      *)

(* Especificación formal en página 128 *)

let rec dCSval = function
     | [] ->  (fun e a -> (e, (return a)))
     | (i,c)::dCS2 ->
      (fun e a -> let(e1,z)=((mod_amb i (Const(c)) e),(return a))
       in let a1 = state_of_post(z)
        in  (dCSval dCS2 e1 a1)
      )
;;

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | T:estructura_de_tipo -> enviroment -> state -> valor_Denotable * poststate
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)

(* Especificación formal en página 130 *)

let rec t = function v ->
  (match v with
    | Tipo("integer") ->
          (fun e a -> let (l ,p0) = asi_loc a in
            let a1 = state_of_post(p0) in
            (Loc(l),mod_alm l (Nulo(NumBt)) a1)
          )
    | Tipo("ENTERO") ->
          (fun e a -> let (l ,p0) = asi_loc a in
            let a1 = state_of_post(p0) in
            (Loc(l),mod_alm l (Nulo(NumBt)) a1)
          )
    | Tipo("BOOLEANO") ->
          (fun e a -> let (l ,p0) = asi_loc a in
            let a1 = state_of_post(p0) in
            (Loc(l),(mod_alm l (Nulo(BoolBt)) a1))
          )
    | Tipo("boolean") ->
          (fun e a -> let (l ,p0) = asi_loc a in
            let a1 = state_of_post(p0) in
            (Loc(l),(mod_alm l (Nulo(BoolBt)) a1))
          )
    | Tipo("REAL") ->
```

```
          (fun e a -> let (l ,p0) = asi_loc a in
            let a1 = state_of_post(p0) in
            (Loc(l),(mod_alm l (Nulo(RealBt)) a1))
          )
    | Tipo("text") ->
          (fun e a -> let (l ,p0) = asi_loc a in
            let a1 = state_of_post(p0) in
            (Loc(l),(mod_alm l (Nulo(Textt)) a1))
          )
    | Tipo("real") ->
          (fun e a -> let (l ,p0) = asi_loc a in
            let a1 = state_of_post(p0) in
            (Loc(l),(mod_alm l (Nulo(RealBt)) a1))
          )
    | Tipo("CBORROSO") ->
          (fun e a -> let (l ,p0) = asi_loc a in
            let a1 = state_of_post(p0) in
            (Loc(l),(mod_alm l (Nulo(CBort)) a1))
          )
    | Tipo("c_fuz") ->
          (fun e a -> let (l ,p0) = asi_loc a in
            let a1 = state_of_post(p0) in
            (Loc(l),(mod_alm l (Nulo(CBort)) a1))
          )
    | Tipo("BORROSO") ->
          (fun e a -> let (l ,p0) = asi_loc a in
            let a1 = state_of_post(p0) in
            (Loc(l),(mod_alm l (Nulo(NumTt)) a1))
          )
    | Tipo("tra") ->
          (fun e a -> let (l ,p0) = asi_loc a in
            let a1 = state_of_post(p0) in
            (Loc(l),(mod_alm l (Nulo(NumTt)) a1))
          )
    | TipoC(d,p) ->
          (fun e a -> let(e1,p0) = (dval1(d) (e,vac_amb) a) in
            (Record(e1,p), p0)
          )
    | Tipo(l) ->
          (fun e a ->  match (acc_amb l e) with
             Tipod(f) -> (f a)
          )
  )

(* DVSval: declaracion_variables -> amb -> state -> amb * poststate        *)

(* Especificación formal en página 128 *)

and
dVSval = function
  | [(i,t0)] ->
      (fun e a->let((d:valor_Denotable),(p0:poststate)) =
        (t(t0) e a) in ((mod_amb i d e),p0)
      )
  | (dVS1::dVS2) ->
      (fun e a ->
        let (e1,z)=(dVSval([dVS1]) e a)
        in (check_alloc (dVSval dVS2 e1)) (z)
      )
and
dVSval1 = function
  | [(i,t0)] ->
```

```
      (fun (e_con,e_fab) a->
        let((d:valor_Denotable),(p0:poststate)) =
        (t(t0) e_con a) in ((mod_amb i d e_fab),p0)
      )
  | (dVS1::dVS2) ->
      (fun (e_con,e_fab) a ->
        let (e1,z)=(dVSval1([dVS1]) (e_con,e_fab) a)
        in (check_alloc (dVSval1 dVS2 (e_con,e1))) (z)
      )

(* DPSval: declaracion_parametros -> amb -> state -> amb * poststate       *)
and
dPSval = function
  | [] -> (fun e a -> (e,(return a)))
  | dPS ->
      (fun e a ->
        (dVSval (dPS:declaracion_variables) e a)
      )
(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | Dval: declaraciones -> amb -> state -> amb * poststate
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)

(* Especificación formal en página 129 *)

and
dval = function
  | []                    -> (fun e a -> (e, return a) )
  | [LDeclConsts([])]     -> (fun e a -> (e, return a) )
  | [LDeclConsts([dCS])] -> dCSval(dCS)
  | [LDeclVars([])]       -> (fun e a -> (e, return a) )
  | [LDeclVars([dVS])]    -> dVSval(dVS)
  | [LDeclAbst(dA)]       -> dAval(dA)
  | [LDeclReco(dR)]       -> dRvalRec(dR)
  | d1::d2 -> (fun e a ->
                let (e1,z)=(dval([d1]) e a) in
                (check_alloc (dval(d2) e1))(z)
              )

and
dval1 = function
  | []                    -> (fun (e1,e2) a -> (e2, return a) )
  | [LDeclVars([])]       -> (fun (e1,e2) a -> (e2, return a) )
  | [LDeclVars([dVS])]    -> dVSval1(dVS)
  | [LDeclAbst(dA)]       -> dAval1(dA)
  | d1::d2 -> (fun (e1,e2) a ->
                let (e,z)=(dval1([d1])(e1,e2) a) in
                (check_alloc (dval1(d2)(e1,e)))(z)
              )

(* DPval: decl_proc -> amb -> state -> amb * poststate                     *)

(* Especificación formal en página 128 *)

and
dPval = function
    { id_proc     = i;
      gr_proc     = gr ;
      es_proc     = estra0 ;
      params_proc = dPS;
      cuerpo_proc = k0 } ->
      (fun (e:amb) (a:state) ->
        let q=( fun argg ind (s1,o1) ->
```

```
          let (gi,est) = ind in
          let est1 = (if estra0 != 0 then estra0 else est.opera) in
          let l = (mar_loc s1) in
          let (e1,z)=(dPSval(dPS) e (s1,o1)) in
          let z1=(check1 (liga_params i (pSval dPS) argg e1))(z) in
          let z2=(ext(k(k0)e1((tnorma(gi,gr)est),l_est(est1))))([z1])
          in (des_loc0 l z2)
              )
        in ((mod_amb i (Abst(Procd q)) e),return a)
      )
and
dPreval = function (i1,i2) ->
  (fun (e_con,e_fab) (s,o) ->
    (match i2 with
       | VariableR(c) -> let (l1,e1) = busca_env e_con c in
         (match (acc_amb l1 e1) with
            | Abst(f) -> ((mod_amb i1 (Abst(f)) e_fab),(return (s,o)))
            | _ -> let h= " no es un predicado "
              in (e_fab,signalerr (s,(put_menserr (h,o)) ))
         )
       | Variable(c) ->
         (match (acc_amb c e_con) with
            | Abst(f) -> ((mod_amb i1 (Abst(f)) e_fab),(return (s,o)))
            | _ -> let h=" no predicado"
              in (e_fab,signalerr (s,(put_menserr (h,o)) ))
         )

       | _ -> let h="  no predicado"
         in (e_fab,signalerr (s,(put_menserr (h,o)) ))
    )
  )

(* DFval: decl_func -> amb -> state -> amb * poststate                    *)

(* Especificación formal en página 129 *)


and
dFval = function { id_func     = i;
                   gr_func     = gr;
                   es_func     = estra0;
                   params_func = dPS;
                   tipo_rdo    = t;
                   cuerpo_func = k0 } ->
  (fun e a ->
    let f=
    (fun g ind (s,o) ->
      let (gi,est) = ind in
      let est1 = (if estra0 != 0 then estra0 else est.opera) in
      let l=(mar_loc s) in
      let (e1,z)=(dPSval(dPS) e (s,o)) in
      let j="_"^i in
      let (e2,z1)=(check_alloc (dVSval([(j,t)]) e1))(z) in
      let z2=(check1 (liga_params i (pSval dPS) g e2))(z1) in
      let z3 = (ext (k k0 e2 ((tnorma(gi,gr) est),l_est(est1)))) ([z2]) in
      let y1 =
      (ext_Expr
        (fun (s1,o) -> (* OJO OJO *)
           (match (acc_amb j e2) with
             Loc(lj) ->
               (match (acc_alm lj s1) with
                 OKacces(v) ->
```

```
                    (match v with
                       | Nulo(_) ->
                         let h=": en la funcion `"^i^"` " in
                         let h=h^"falta sentencia `<- `\n"
                         in ( Errexpr,
                              signalerr (s1,put_menserr(h,o))
                            )
                       | _ -> ( Exprval(v), return (s1,o) )
                    )
                  | _ ->  let h=": en la funcion `"^i^"` " in
                          let h=h^"error semantico A `\n"
                          in ( Errexpr,
                              signalerr (s1,put_menserr(h,o))
                            )
              )
            | _ ->  let h=": en la funcion `"^i^"` " in
                    let h=h^"error semantico A `\n"
                    in ( Errexpr,
                        signalerr (s1,put_menserr(h,o))
                       )
          )
        )
      ) (z3) in (auxDFval l y1)
    ) in ((mod_amb i (Abst(Funcd f)) e),(return a))
  )

and
dFval1 = function { id_func     = i;
                    gr_func     = gr;
                    es_func     = estra0;
                    params_func = dPS;
                    tipo_rdo    = t;
                    cuerpo_func = k0 } ->
  (fun (e_con,e_fab) a ->
    let f=
    (fun g ind (s,o) ->
      let (gi,est) = ind in
      let est1 = (if estra0 != 0 then estra0 else est.opera) in
      let l=(mar_loc s) in
      let (e1,z)=(dPSval(dPS) e_fab (s,o)) in
      let j="_"^i in
      let (e2,z1)=(check_alloc (dVSval1([(j,t)])(e_con,e1)))(z) in
      let z2=(check1 (liga_params i (pSval dPS) g e2))(z1) in
      let z3 = (ext (k k0 (e2@e_con) ((tnorma(gi,gr) est),l_est(est1)))) ([z2]) in
      let y1 =
      (ext_Expr
        (fun (s1,o) -> (* OJO OJO *)
           (match (acc_amb j e2) with
             Loc(lj) ->
               (match (acc_alm lj s1) with
                 OKacces(v) ->
                 (match v with
                    | Nulo(_) ->
                      let h=": en la funcion `"^i^"` " in
                      let h=h^"falta sentencia `<- `\n"
                      in ( Errexpr,
                           signalerr (s1,put_menserr(h,o))
                         )
                    | _ -> ( Exprval(v), return (s1,o) )
                 )
               )
           )
```

```
        )
      ) (z3) in (auxDFval l y1)
    ) in ((mod_amb i (Abst(Funcd f)) e_fab),(return a))
  )

and
auxDFval l = function
  | [] -> []
  | (x,z4):: q ->
        let z5 = (check1 (des_loc l))(z4) in
        let t2 = (x,z5)::auxDFval l q in t2

and
dAval = function
  | [] -> (fun e a -> (e, return a) )
  | d1::d2 -> (fun e a ->
     let (e1,z)=(dA1val(d1)e a) in (check_alloc (dAval(d2)e1))(z))

and
dAval1 = function
  | [] -> (fun (e_con,e_fab) a -> (e_fab, return a) )
  | d1::d2 -> (fun (e_con,e_fab) a ->
     let (e1,z)=(dA1val1(d1)(e_con,e_fab) a) in
     (check_alloc (dAval1(d2)(e1,e1)))(z))

and
dA1val = function
  | DeclFunc(dF)    -> dFval(dF)
  | DeclProc(dP)    -> dPval(dP)
(*  | DeclPre(i1,i2)  -> dPreval(i1,i2) *)

and
dA1val1 = function
  | DeclFunc(dF)    -> dFval1(dF)
  | DeclPre(i1,i2)  -> dPreval(i1,i2)

and
dRvalRec = function
  | [] -> (fun e a -> (e, return a) )
  | d1::d2 ->
     (fun e a ->
        let (e1,z)=(dRval d1 e a) in
        (check_alloc (dRvalRec d2 e1))(z)
     )

(* DRval: decl_reco -> amb -> state -> amb * poststate                    *)

(* Especificación formal en página 129 *)

and
dRval = function
  { id_reco     = i;
    cuerpo_reco = k0 } -> (fun e a ->
         let f=( fun  (a1:state) -> (t (k0) e a1 ) )
         in ((mod_amb i (Tipod (f)) e),(return a)))

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | Kval: bloque -> amb -> ind_borr -> t_op -> state -> Respuesta
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)

(* Especificación formal en página 128 *)
```

```
and
k bloq  = fun e (i:ind_borr) ((s,r):state) ->
  let l = (mar_loc s) in
  let (e1,z) = (dval (bloq.decls) e (s,r)) in
  (match z with
    | OK(a) ->
      let z1 = sval (bloq.cuerpo) e1 i a in
      (des_loc0 l) (z1)
    | b -> (* (print_string("NO OK"));*) [ b ]
  )

and
auxAsiRe e1 = function
  | Variable(nv) ->(* (print_string(nv);print_newline()); *)
    (match (acc_amb nv e1) with
      | Loc(l) -> l
      | _ -> (-1:location)
    )
  | VariableR(ln) -> auxA e1 ln
  | _ -> (-1:location)

and
auxA e1 = function
  | [] -> (-1:location)
  | c :: b ->(* (print_string(c);print_newline()); *)
    (match (acc_amb c e1) with
      | Record(d,p) -> auxA d b
      | Loc(l) -> l
      | _ -> (-1:location)
    )

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 | Sval: sentencias -> amb -> ind_bor -> t_op -> state -> respuesta
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
and
sval (s:sentencia) (e:amb) i0 (a:state) =
  let rec svalrec = function

(* Especificación formal en página 130 *)

    | Asignacion2(i,e0,l) ->
      (fun e i0 (s,o) ->
        match (acc_amb i e) with
          | Loc(l) ->
            (match (eval(e0) e i0 a) with
              | (Exprval(v1),z) ->
                (match v1 with
                  | RealB(x0,x1) ->
                    let (v2,r) = (x0,x1) in
                    let z1 = state_of_post(z) in
                    let (s0,o) = z1 in
                    let vA = (acc_alm l s0) in
                    let k0 =
                    (match vA with
                      | OKacces(NumB(vE,gr))->(mod_alm l (NumB(vE,v2)) z1)
                      | OKacces(BoolB(vB,gr))->(mod_alm l (BoolB(vB,v2)) z1)
                      | OKacces(RealB(vR,gr))->(mod_alm l (RealB(vR,v2)) z1)
                      | OKacces(NumT(vb,gr))->(mod_alm l (NumT(vb,v2)) z1)
                    ) in
                    [k0]
                )
              | (Errexpr,z) -> [z]
```

```
          )
        | Const(c) -> let h=": en linea "^string_of_int(l)^
                            "`"^i^"` es una constante"
          in [signalerr (s,put_menserr (h,o))]
        | Abst(k0)  -> let h=": en linea "^string_of_int(l)^
                             "`"^i^"` es una abstraccion "
          in [signalerr (s,put_menserr (h,o))]
        | Errvalue -> let h=":en linea "^string_of_int(l)^
                            " variable `"^i^"` no existe"
          in [signalerr (s,put_menserr (h,o))]
        | _ -> let h=": en linea "^string_of_int(l)^
                     "`"^i^"`  no es una variable"
          in [signalerr (s,put_menserr (h,o))]
      )

    | Asignacion3(i,n,e0,lin) ->
      (fun e i0 (s1,o1) ->
        (match (eval(e0) e i0 (s1,o1)) with
          | (Exprval(v1),z) ->
              (match v1 with
                | RealB(x0,x1) ->
                  let (s0,o) = state_of_post(z) in
                  let l = (acc_amb i e) in
                  (match l with
                     | Loc(v) ->
                       (match (acc_alm v s0) with
                          | OKacces(al) ->
                            (match al with
                               | NumT((t1,t2,t3,t4),g0) ->
                                 let v2 =
                                 (match n with
                                    | 1 -> NumT((x0,t2,t3,t4),g0)
                                    | 2 -> NumT((t1,x0,t3,t4),g0)
                                    | 3 -> NumT((t1,t2,x0,t4),g0)
                                    | 4 -> NumT((t1,t2,t3,x0),g0)
                                 ) in [(mod_alm v v2 (s0,o))]
                               | _ -> let h=" en linea "^string_of_int(lin) in
                                      let h1=h^" `"^i^"` debe ser NumT\n"
                                 in [signalerr (s0,(put_menserr (h1,o)))]
                            )
                          | _ -> let h=" en linea "^string_of_int(lin) in
                                 let h1=h^" sin inicializar `"^i^"`\n"
                            in [signalerr (s0,(put_menserr (h1,o)))]
                       )
                     | _ -> let h=" en linea "^string_of_int(lin)^
                                  " `"^i^"` no es variable\n"
                       in [signalerr (s0,(put_menserr (h,o)))]
                  )
                | _ -> let h=" en linea "^string_of_int(lin)^
                             " solo Real\n"
                  in [signalerr (s1,(put_menserr (h,o1)))]
              )
          | _ -> let h=" en linea "^string_of_int(lin)^
                       " expresion incorrecta\n"
            in [signalerr (s1,(put_menserr (h,o1)))]
        )
      )

(* Especificación formal en página 130 *)

    | Asignacion(i,e0,l) ->
      (fun e i0 (s,o) ->
```

```
        (match (auxAsiRe e i) with
          | -1 -> let h=": en linea "^string_of_int(l) in
                  let h1=h^" asignar a no variable"
                  in [signalerr (s,put_menserr (h1,o))]
          | lo -> let OKacces(ti) = (acc_alm lo s) in
            (match e0 with
              | Aplicacion(i1,lA) ->
                let l_e_p = (eval_apl1(i1,lA) e i0 (s,o)) in
                let kK = auxAsigFun lo (tipo_stv(ti)) i0 l_e_p in
                ( if kK = [] then
                    let h =" en linea "^string_of_int(l)^
                           " No hay nada que asignar" in
                    [signalerr(s,(put_menserr (h,o)))]
                  else
                    kK
                )
              | Id(VariableR(l0)) ->
                (match (eval_reco(l0) e i0 (s,o)) with
                  (Exprval(v1),z) ->
                    let k= state_of_post(z) in [(mod_alm_g lo v1 i0 k)]
                  | _ -> let h=" en linea "^string_of_int(l)^
                                " tipo incorrecto" in
                         [signalerr(s,(put_menserr (h,o)))]
                )
              | _ -> (* (print_string("Ni A ni A1 ni VR")); *)
                (match (eval(e0) e i0 a) with
                  | (Exprval(v1),z) ->
                    let (s0,o) = state_of_post(z) in
                        ( if tipo_stv(ti) = tipo_stv(v1)  then
                            [(mod_alm_g lo v1 i0 (s0,o))]
                          else
                            let h=" en linea "^string_of_int(l)^
                                  " tipos distintos" in
                            [signalerr(s0,(put_menserr (h,o)))]
                        )
                  | (Errexpr,z) -> [z]
                )
            )
        )
      )

(* Especificación formal en página 130 *)

    | Alternativa(lCG,l) ->
      (fun e ind a ->  let (b0,r) = (b e ind a lCG) in
        (if b0 then
            (g e ind a false lCG)
         else
            let (s1,o1) = a in
            let o_prima = put_message("Abort",o1) in
            [abort (s1,o_prima)]
        )
      )

(* Especificación formal en página 130 *)

    | Condicional(e0,s1,s2,l) ->
      (fun e i2 (s,o) ->
        match (eval(e0) e i2 (s,o)) with
         | (Exprval(v),z) ->
           (match v with
             | BoolB((x0,x1)) -> let (g,est) = i2 in
```

```
              let (s10,o1)=state_of_post(z) in
              let (b0,r) = (x0,x1) in
                (if b0 then (sval s1 e i2 (s10,o1) )
                 else (sval s2 e i2 (s10,o1))
                )
            | _ -> let h=" en linea "^string_of_int(l)^
                        " algo va mal 5\n"
                   in [signalerr (s,(put_menserr (h,o)))]
          )
        | _ -> let h=" en linea "^string_of_int(l)^" algo va mal 5b\n"
          in [signalerr (s,(put_menserr (h,o)))]
    )

(* Especificación formal en página 131 *)

    | Sentencia_nula -> (fun e i a -> [return a])

(* Especificación formal en página 130 *)

    | Bucle(e0,c,l) as actual ->
      (fun e i2 a -> let (s2,o2) = a in
        (match (eval(e0) e i2 (s2,o2)) with
          | (Errexpr,z) ->
            let h=" en linea "^string_of_int(l)^" semantico 11\n"
            in [signalerr (s2,(put_menserr (h,o2)))]
          | (Exprval(v),z) ->
            (match v with
              | BoolB((x0,x1)) ->
                let (b0,r) = (x0,x1) in (* OJO a la transimision *)
                let t1=state_of_post(z) in
                ( if b0 then
                    let (g,est) = i2
                    in let t2 = (sval c e i2 t1)
                    in (ext(sval actual e i2))(t2)
                  else
                    [return t1]
                )
              | _ -> let h=" en linea "^string_of_int(l)^" algo va mal 5c\n" in
                let t0=state_of_post(z)
                in [signalerr (s2,(put_menserr (h,o2)))]
            )
        )
      )

(* Especificación formal en página 131 *)

    | Repeticion(lCG,l) as actual ->
      (fun e i a ->
        let (b0,t0)= (b e i a lCG) in
        ( if b0 then
            let z1 = (g e i a false lCG) in
            let t2 = (ext(sval actual e i))(z1) in
            (t2)
          else
            [return a]
        )
      )

(* Especificación formal en página 131 *)
```

```
    | Print(eI,l) ->
      (fun e i a ->
        match eI with
         | [e0] -> ( auxPrint [e0] e i a)
         | eI1::lEI -> let s1=Print([eI1],l) and s2=Print(lEI,l)
             in (ext(sval s2 e i))(sval s1 e i a)
      )

(* Especificación formal en página 131 *)

    | Retorna(l) -> (fun e i a -> [return a])

(* Especificación formal en página 131 *)

    | Ret_expr(i,e0,l)  ->
      (fun e i0 a ->
        let (x,o)= a in
        let t1 = check1 ( fun a1 -> (return a1) )
        (let j=("_"^i) in
          (match (acc_amb j e) with
            | Loc(l) ->
              (match (acc_alm l x) with
                | OKacces(v) ->
                  let t0=tipo_stv(v) in
                  (match (eval(e0) e i0 (x,o)) with
                    | (Errexpr,z) ->
                      let (s1,o1)=state_of_post(z) in
                      let h="en linea "^string_of_int(l)^" en Ret 1"
                      in signalerr (s1,put_menserr(h,o1))
                    | (Exprval(v1),z) ->
                      ( if t0 != tipo_stv(v1) then
                          let (s1,o1)=state_of_post(z) in
                          let h="en linea "^string_of_int(l)^ " en Ret 1"
                          in signalerr (s1,put_menserr(h,o1))
                        else let (j,k) = i0 in
                          (mod_alm l v1 (x,o))
                      )
                  )
                | Erracces ->
                  let h="en linea "^string_of_int(l)^
                        " de memoria en el acceso a `"^j^"`\n"
                  in signalerr (x,put_menserr(h,o))
              )
            | Errvalue ->
              let h=" en linea "^string_of_int(l)^" en Ret 2 con "^i
              in signalerr (x,put_menserr(h,o))
          )
        ) in [t1]
      )

(* Especificación formal en página 131 *)

    | Bloq(k0,l) ->
      (fun e i a -> k k0 e i a )

    | Llamada(i,lA,l)  ->
      (fun e ind (s,o) ->
        match (acc_amb i e) with
         | Abst(Procd(q)) ->
           let (g0,z)=(aSval lA e ind (s,o)) in
           let z1=(verifica_args i g0 (state_of_post z)) in
```

```
          ext(q g0 ind)([z1])
        | _ ->
          let h=" en linea "^string_of_int(l)^
                " en Llamada a `"^i^"`, no es procedimiento\n"
          in [signalerr (s,(put_menserr (h,o)))]
      )

(* Especificación formal en página 131 *)

    | Compuesta(s1,lS) ->
      ( fun e i a -> let j = sval s1 e i a in
          ext (sval lS e i) j
      )
(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |  B: Lista_guardados -> amb -> state -> boolB
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)

(* Especificación formal en página 131 *)

and
b e i a = function
  | [] -> (false,k_est.neutro)
  | (e0,_) :: lcg ->
    (match e0 with
      | Default -> (true,1.)
      | _ -> (match (eval(e0) e i a) with
                | (Exprval(v),z) ->
                  (match v with
                    | BoolB((x0,x1)) ->
                      let (b0,r) = (x0,x1) in
                      ( if b0 then  (x0,x1)
                        else (b e i a lcg)
                      )
                    | _ -> (false,k_est.neutro)
                  )
                | (Errexpr,z) -> (false,k_est.neutro)
              )
    )
(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |  G: Lista_guardados -> amb -> ind_bor -> t_op -> state -> respuesta
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)

(* Especificación formal en página 131 *)

and
g e i (sx,ox) bo = function
  | [] -> (if bo then [] else [return (sx,ox)])
  | (e0,ls1)::lCG ->
    (match e0 with
      | Default -> (if bo then [] else (sval ls1 e i (sx,ox)))
      | _ -> (match (eval(e0) e i (sx,ox)) with
                | (Exprval(v),z) ->
                  (match v with
                    | BoolB((x0,x1)) ->
                      let (b0,r)=(x0,x1) in
                      ( if b0 then
                          let k = state_of_post(z) in
                          let (gi,est) = i in
                          let t2=(if bo then copy_state(k) else k) in
                          let l1=((sval ls1) e ((tnorma(gi,r) est),est) k) in
                          (l1 @ ((g e i t2 b0 lCG)))
```

```
                    else (g e i (sx,ox) bo lCG)
                  )
                | _ ->
                  let h="algo va mal 5\n"
                  in [signalerr(sx,(put_menserr (h,ox)))]
              )
            | _ ->
              let h="algo va mal 5bis\n"
              in [signalerr(sx,(put_menserr (h,ox)))]
          )
    )
and
auxAsigFun l ti i = function
  | [] -> []
  | (a1,z)::b0 ->
    let b = compru z in
    (if b then
      (match a1 with
        | Exprval(a) ->
          let z1 = state_of_post(z) in
          let (s0,o) = z1 in
          let k0  =
          ( if ti = tipo_stv(a) then
              (mod_alm l a z1)
            else
              let h="tipos distintos"
              in signalerr (s0,put_menserr(h,o))
          ) in
           [k0]@auxAsigFun l ti i b0
        | _ ->
          let (s1,o1)=state_of_post(z) in
          let h="semantico 13\n" in
          let o_prima = put_menserr(h,o1) in
          [signalerr(s1,o_prima)] @auxAsigFun l ti i b0
      )
    else
      [z]@auxAsigFun l ti i b0
  )

and
auxPrint eU e i (x,o) = match eU with
  | [Expres e0] ->
    (match (eval(e0) e i (x,o)) with
      | (Errexpr,z) ->
        let (s1,o1)=state_of_post(z) in
        let h="semantico 12\n" in
        let o_prima = put_menserr(h,o1) in
        [signalerr(s1,o_prima)]
      | (Exprval(v),z) ->
        (match v with
          | NumB((x0,x1)) ->
            let (s1,o1)=state_of_post(z) in
            let o_prima = esc_val((NumBc (x0,x1)),o1) in
            [return(s1,o_prima)]
          | RealB((x0,x1)) ->
            let (s1,o1)=state_of_post(z) in
            let o_prima = esc_val((RealBc (x0,x1)),o1) in
            [return(s1,o_prima)]
          | BoolB((x0,x1)) ->
            let (s1,o1)=state_of_post(z) in
            let o_prima = esc_val((BoolBc (x0,x1)),o1) in
            [return(s1,o_prima)]
```

```
          | NumT((x0,x1)) ->
            let (s1,o1)=state_of_post(z) in
            let o_prima = esc_val((NumTc (x0,x1)),o1) in
            [return(s1,o_prima)]
          | Text((x0,x1)) ->
            let (s1,o1)=state_of_post(z) in
            let o_prima = esc_val((Textc (x0,x1)),o1) in
            [return(s1,o_prima)]
          | CBor((x0,x1,b)) ->
            let (s1,o1)=state_of_post(z) in
            let o_prima = esc_val((CBorc (x0,x1,b)),o1) in
            [return(s1,o_prima)]
          | _ ->
            let (s1,o1)=state_of_post(z) in
            let h="semantico 13b\n" in
            let o_prima = put_menserr(h,o1) in
            [signalerr(s1,o_prima)]
        )
    )
  | [Cadena h] ->
    let o_prima = put_message(h,o) in
    [return(x,o_prima)]
  | [Escape st] -> let str_esc =
    (match st with
      | NuevaLine -> "\n"
      | Tabulador -> "\t"
    ) in
    let o_prima = put_message(str_esc,o) in
    [return(x,o_prima)] in (svalrec s e i0 a)
;;

(*
  P: programa -> alm -> poststate
*)

(* Especificación formal en página 128 *)

let p prog = fun s ->
  k (prog.texto)(vac_amb ) (k_est.neutro,k_est) (s,vac_sal)
;;
```

Fichero: main.ml

```
open Sys;;
open Gramat;;
open Estrat;;
open Ffvaluac;;
open Algseman;;
open Parser;;
exception Error_sintaxis of string;;

let construye_arbol canal =
  let lexbuf = Lexing.from_channel canal in
      try
      Parser.pROGRAMA Lexer.token lexbuf with
      | Parsing.Parse_error  ->
        prerr_string ("> Error sintactico. Linea "
                    ^(string_of_int(lin_act Lexer.lin )));
        prerr_endline "";
        prerr_string ("No se esperaba: "
             ^ "`" ^ (Lexing.lexeme lexbuf) ^ "'");
        prerr_endline "";
        raise (Error_sintaxis "")
      | Error_sintaxis mensaje ->
        prerr_string ("> Error sintactico. Linea "
                        ^(string_of_int(lin_act Lexer.lin )));
        prerr_endline "";
        prerr_string ("> Error : "^ mensaje ^ " .");
        prerr_endline "";
        raise (Error_sintaxis " ")
;;
let rec lista  = function
  [] -> ()
  | x::l ->
      (print_string(x)
      ); lista l
;;
let rec lista1 num = function
  [] -> print_newline()
  | a::b -> let w =
      (match a with
         | OK(z) -> z
         | Err(z) -> z
         | Abo(z) -> z
      ) in
        ( print_newline();
          print_string("<");
          print_string(string_of_int(num));
          print_string(">: ");
          let (_,o) = w in
          (lista (List.rev o))
        ); lista1 (num+1) b
;;
let interpreta nomb =
    try
       begin
         Parsing.clear_parser();
         print_string("Construido arbol");
         print_newline();
         let z = p nomb (vac_alm) in
(*
         let lF = lstate_of_Res(z) in
*)
```

```
        lista1 1 (List.rev z)
        end
  with Sys_error mensaje ->
    prerr_string "> Error del sistema: "; prerr_endline mensaje
    | Not_found1 st  -> prerr_string
                          ("'"^st^"': Identificador no declarado")
;;
let rec modulos arbol_d i =
if i < (Array.length argv ) then
    let canal = open_in argv.(i) in (print_string
                   ("Analizando "^argv.(i)^"\n");print_newline());
    let a = construye_arbol canal in
    let kk = close_in canal in
    let c = ({texto = {decls=(arbol_d.texto.decls @ a.texto.decls);
          cuerpo=Compuesta(arbol_d.texto.cuerpo, a.texto.cuerpo)}})
          in modulos c (i+1)
else arbol_d
;;
begin
  if Array.length argv >=2 & argv.(1)= "-e" then
    let arbol_d = ({texto={decls=[];cuerpo=Sentencia_nula}}) in
    interpreta (modulos (arbol_d) 3 )
  else
    (print_string("\nUsar: '"^argv.(0)
                ^"' -e (1/2/3/4/5/6) fichero(s)");print_newline());
    exit 0
end;;
```

Fichero: Makefile

```
CAMLC=ocamlc
CAMLYACC=ocamlyacc
CAMLLEX=ocamllex
#COMPFLAGS=-W
#LINKFLAGS=-g -O fast

COMPFLAGS=
LINKFLAGS=

OBJS= estrat.cmo lexer.cmo parser.cmo algseman.cmo exvaluac.cmo \
ffvaluac.cmo main.cmo

GENSOURCES=lexer.ml parser.ml parser.mli


all: genera p


genera:
#lexer.ml: lexer.mll
$(CAMLLEX) lexer.mll
#
#parser.ml parser.mli: parser.mly
$(CAMLYACC) parser.mly

p: $(OBJS)
$(CAMLC) -o p $(OBJS)

clean:
rm -f *.cmi *.cmo
rm -f $(GENSOURCES) p


.SUFFIXES : .mli .ml .cmi .cmo


.mli.cmi:
$(CAMLC) $(COMPFLAGS) -c $<

.ml.cmo:
$(CAMLC) $(COMPFLAGS) -c $<


depend: $(GENSOURCES)
mv Makefile Makefile.bak
(sed -n -e '1,/^### NO BORRAR ESTA LINEA/p' Makefile.bak; \
         ocamldep *.mli *.ml) > Makefile
rm Makefile.bak


### TODO LO QUE VA BAJO ESTE COMENTARIO ES GENERADO
### NO BORRAR ESTA LINEA
algseman.cmi: gramat.cmi
exvaluac.cmi: algseman.cmi gramat.cmi
ffvaluac.cmi: algseman.cmi exvaluac.cmi gramat.cmi
parser.cmi: gramat.cmi
algseman.cmo: estrat.cmi gramat.cmi algseman.cmi
algseman.cmx: estrat.cmx gramat.cmi algseman.cmi
estrat.cmo: estrat.cmi
```

```
estrat.cmx: estrat.cmi
exvaluac.cmo: algseman.cmi estrat.cmi gramat.cmi exvaluac.cmi
exvaluac.cmx: algseman.cmx estrat.cmx gramat.cmi exvaluac.cmi
ffvaluac.cmo: algseman.cmi estrat.cmi exvaluac.cmi gramat.cmi \
ffvaluac.cmi
ffvaluac.cmx: algseman.cmx estrat.cmx exvaluac.cmx gramat.cmi \
ffvaluac.cmi
fix.cmo: fix.cmi fix.cmi
fix.cmx: fix.cmx fix.cmi
lexer.cmo: estrat.cmi gramat.cmi parser.cmi
lexer.cmx: estrat.cmx gramat.cmi parser.cmx
main.cmo: algseman.cmi estrat.cmi ffvaluac.cmi gramat.cmi lexer.cmo \
    parser.cmi
main.cmx: algseman.cmx estrat.cmx ffvaluac.cmx gramat.cmi lexer.cmx \
    parser.cmx
parser.cmo: estrat.cmi gramat.cmi parser.cmi
parser.cmx: estrat.cmx gramat.cmi parser.cmi
```

Apéndice D

Programas

Fichero: cab_par.h1

```
\       Parecido y alrededor.
\       Para estrategias 1,2,3 usar _y
\       para estrategias 4,5,6 usar _y1
const _y = {0.,0.,0.,0.};
const _y1 = {1.,1.,1.,1.};
function parecido(ENTERO x1): BORROSO
const
    a1= 5., b1=10.;

var
    BORROSO z;
    REAL co;
begin
    z:=_y1;
    co := EXTR(x1);
    z{1}:=co-a1-b1;
    z{2}:=co-a1;
    z{3}:=co+a1;
    z{4}:=co+a1+b1;
    parecido <- z
end;

function alrededor(ENTERO x1): BORROSO
const
    b1= 4., a1=5.;

var
    BORROSO z;
    REAL co;
begin
    z:=_y1;
    co := EXTR(x1);
    z{1}:=co-a1-b1;
    z{2}:=co-a1;
    z{3}:=co+a1;
    z{4}:=co+a1+b1;
    alrededor <- z
end;
end INCLUDE
```

Fichero: cab_cmp.h1

```
\ Comparacion de borrosos

function cmp_bor(BORROSO b, r): BOOLEANO
    function cmp_rea(REAL re1,re2):BOOLEANO
    begin
        cmp_rea <- re1 < re2
    end;
    function cmp_coin(BORROSO b1,b2):BOOLEANO
    var
        REAL m1,m2,m3;
        BOOLEANO v1,v2;
    begin
        if b2{2} = b1{2} then
            cmp_coin <- true
        else
            m1 := b2{2} - b1{2};
            m2 := b1{3} - b1{2};
            m3 := m1 / m2 ;
            v2 := false; GRADO(v2) := m3;
            v1 := true ; GRADO(v1) := 1. - m3;
            CASE
                true -> cmp_coin <- v1
            [ ] true -> cmp_coin <- v2
            ESAC
        end if
    end;
    function cmp_coin1(BORROSO b1,b2):BOOLEANO
    var
        REAL m1,m2,m3;
        BOOLEANO v1,v2;
    begin
        if b2{3} != b1{2}  then
            m1 := b1{3} - b1{2};
            m2 := b2{3} - b1{2};
            m3 := m2 / m1 ;
            v1 := true; GRADO(v1) := m3;
            v2 := false; GRADO(v2) := 1. - m3;
            CASE
                true -> cmp_coin1 <- v1
            [ ] true -> cmp_coin1 <- v2
            ESAC
        else
            cmp_coin1 <- false
        end if
    end;
    function cmp_conte(BORROSO b1,b2):BOOLEANO
    var
        REAL m1,m2,m3;
        BOOLEANO v1,v2;
    begin
        if b2{2} = b1{2} then
            cmp_conte <- true
        else
            m2 := b1{3} - b1{2};
            m1 := b2{2} - b1{2};
            m3 := m2 / m1 ;
            v1 := true; GRADO(v1) := m3;
            v2 := false; GRADO(v2) := 1. - m3;
            CASE
                true -> cmp_conte <- v1
```

```
            [ ] true -> cmp_conte <- v2
            ESAC
        end if
    end;
    function cmp_conte1(BORROSO b1,b2):BOOLEANO
    var
        REAL m1,m2,m3;
        BOOLEANO v1,v2;
    begin
        m1 := b2{3} - b2{2};
        m2 := b1{3} - b1{2};
        m3 := m2 / m1 ;
        v1 := true; GRADO(v1) := m3;
        v2 := false; GRADO(v2) := 1. - m3;
        CASE
            true -> cmp_conte <- v1
        [ ] true -> cmp_conte <- v2
        ESAC
    end;
    var
        BOOLEANO bo1,bo2,bo3,bo4,bo;
        REAL d,dn,dd;
begin
\    print("b ",b, " r ",r,nl); \
    bo1:= cmp_rea(b{3},r{2});  \Comp nucleos
    bo2:= cmp_rea(r{3},b{2});  \
    bo3:= cmp_rea(b{3},r{3});
    bo4:= cmp_rea(b{2},r{2});
    cmp_bor <- false (0.0); \ Por el problema de la eleccion

    if (bo1 or bo2) then \ Caso |----b----| |--r--| o contrario
        cmp_bor <- false
    else
        if bo3 then
            if not(bo4) then        \ Caso    |---b---|
\               print("bo3 y Not bo4"); \
                cmp_bor <- true       \       |-----r-----|
            else
\               print("bo3 y bo4");
                bo := cmp_coin(b,r);    \ Caso  |---b---|
                cmp_bor <- bo           \         |---r---|
            end if
        else
            if not(bo4) then
\               print("not bo3 y Not bo4");
                bo := cmp_coin1(b,r);   \ Caso   |---b---|
                cmp_bor <- bo           \      |---r---|
            else
\               print("not bo3 y bo4"); \ Caso |----b----|
                bo := cmp_conte1(b,r);  \        |--r--|
                cmp_bor <- bo
            end if
        end if
    end if
end;
end INCLUDE
```

Fichero: bru7.k2

```
\ pru7.p
const y = {0.5,6.,7.,0.8},
       c_joven = {17.,20.,50.,60.},
       c_viejo = {40.,45.,70.,75.} ;
var
       BOOLEANO b_p_d,b_p_g,b_e_j,b_e_v ;

type Edad   = TIPO_LI
                  BORROSO edad
               VA_LI
               joven: ()
                   var
                       BOOLEANO va;
               begin
                   va:= cmp_bor(edad, c_joven);
                   joven <- va
               end |
               viejo : ()
                   var
                       BOOLEANO va;
               begin
                   va:= cmp_bor(edad,c_viejo);
                   viejo <- va
               end
            end,
     Peso   = TIPO_LI
                BORROSO peso
            VA_LI
            delgado: ()
                var
                    BOOLEANO va;
            begin
                va:= cmp_bor(peso,c_delgado) ;
                delgado <- va
            end |
            gordo: ()
                var
                    BOOLEANO va;
            begin
                va:= cmp_bor(peso,c_gordo);
                gordo <- va
            end
         end;
var
     Edad e1 ;
     BOOLEANO bo2,bo3;
     ENTERO n;
begin
    e1.edad:= {30.,34.,44.,50.};
    n := 3;
    while (n < 10) do
       bo2 := e1.joven();
       bo3 := e1.viejo();
       print("EDAD = ",e1.edad," joven ",bo2," viejo ",bo3,nl);
       e1.edad := e1.edad + 4.;
       n := n + 2
    end do
end.
```

Produciendo la siguiente salida:

```
<1>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo FALSO(0.7)
EDAD = {38, 42, 52, 58}(1) joven FALSO(0.2) viejo FALSO(0.3)
EDAD = {42, 46, 56, 62}(1) joven FALSO(0.6) viejo CIERTO(1)

<2>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo FALSO(0.7)
EDAD = {38, 42, 52, 58}(1) joven FALSO(0.2) viejo FALSO(0.3)
EDAD = {42, 46, 56, 62}(1) joven CIERTO(0.4) viejo CIERTO(1)

<3>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo FALSO(0.7)
EDAD = {38, 42, 52, 58}(1) joven FALSO(0.2) viejo CIERTO(0.7)
EDAD = {42, 46, 56, 62}(1) joven FALSO(0.6) viejo CIERTO(1)

<4>: EDAD = {30, 34, 44, 50}(1) joven (1)CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo FALSO(0.7)
EDAD = {38, 42, 52, 58}(1) joven FALSO(0.2) viejo CIERTO(0.7)
EDAD = {42, 46, 56, 62}(1) joven CIERTO(0.4) viejo CIERTO(1)

<5>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo FALSO(0.7)
EDAD = {38, 42, 52, 58}(1) joven CIERTO(0.8) viejo FALSO(0.3)
EDAD = {42, 46, 56, 62}(1) joven FALSO(0.6) viejo CIERTO(1)

<6>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo FALSO(0.7)
EDAD = {38, 42, 52, 58}(1) joven CIERTO(0.8) viejo FALSO(0.3)
EDAD = {42, 46, 56, 62}(1) joven CIERTO(0.4) viejo CIERTO(1)

<7>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo FALSO(0.7)
EDAD = {38, 42, 52, 58}(1) joven CIERTO(0.8) viejo CIERTO(0.7)
EDAD = {42, 46, 56, 62}(1) joven FALSO(0.6) viejo CIERTO(1)

<8>: EDAD = {30, 34, 44, 50}(1) joven (1)CIERTO(1) viejo (1)FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo FALSO(0.7)
EDAD = {38, 42, 52, 58}(1) joven CIERTO(0.8) viejo CIERTO(0.7)
EDAD = {42, 46, 56, 62}(1) joven CIERTO(0.4) viejo CIERTO(1)

<9>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo CIERTO(0.3)
EDAD = {38, 42, 52, 58}(1) joven FALSO(0.2) viejo FALSO(0.3)
EDAD = {42, 46, 56, 62}(1) joven FALSO(0.6) viejo CIERTO(1)

<10>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo CIERTO(0.3)
EDAD = {38, 42, 52, 58}(1) joven FALSO(0.2) viejo FALSO(0.3)
EDAD = {42, 46, 56, 62}(1) joven CIERTO(0.4) viejo CIERTO(1)

<11>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo CIERTO(0.3)
EDAD = {38, 42, 52, 58}(1) joven FALSO(0.2) viejo CIERTO(0.7)
EDAD = {42, 46, 56, 62}(1) joven FALSO(0.6) viejo CIERTO(1)

<12>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo CIERTO(0.3)
EDAD = {38, 42, 52, 58}(1) joven FALSO(0.2) viejo CIERTO(0.7)
EDAD = {42, 46, 56, 62}(1) joven CIERTO(0.4) viejo CIERTO(1)

<13>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
```

```
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo CIERTO(0.3)
EDAD = {38, 42, 52, 58}(1) joven CIERTO(0.8) viejo FALSO(0.3)
EDAD = {42, 46, 56, 62}(1) joven FALSO(0.6) viejo CIERTO(1)

<14>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo CIERTO(0.3)
EDAD = {38, 42, 52, 58}(1) joven CIERTO(0.8) viejo FALSO(0.3)
EDAD = {42, 46, 56, 62}(1) joven CIERTO(0.4) viejo CIERTO(1)

<15>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo CIERTO(0.3)
EDAD = {38, 42, 52, 58}(1) joven CIERTO(0.8) viejo CIERTO(0.7)
EDAD = {42, 46, 56, 62}(1) joven FALSO(0.6) viejo CIERTO(1)

<16>: EDAD = {30, 34, 44, 50}(1) joven CIERTO(1) viejo FALSO(1)
EDAD = {34, 38, 48, 54}(1) joven CIERTO(1) viejo CIERTO(0.3)
EDAD = {38, 42, 52, 58}(1) joven CIERTO(0.8) viejo CIERTO(0.7)
EDAD = {42, 46, 56, 62}(1) joven CIERTO(0.4) viejo CIERTO(1)
```

Fichero: cab_par.d1

```
\       Parecido y alrededor.
\       Para estrategias 1,2,3 usar _y
\       para estrategias 4,5,6 usar _y1
const _y = {0.,0.,0.,0.};
const _y1 = {1.,1.,1.,1.};
function parecido(ENTERO x1): BORROSO
const
    a1= 3000., b1=4000.;
var
    BORROSO z;
    REAL co;
begin
    z:=_y1;
    co := EXTR(x1);
    z{1}:=co-a1;
    z{2}:=co;
    z{3}:=co;
    z{4}:=co+b1;
    parecido <- z
end;

function alrededor(ENTERO x1): BORROSO
const
    b1= 4., a1=5.;
var
    BORROSO z;
    REAL co;
begin
    z:=_y1;
    co := EXTR(x1);
    z{1}:=co-a1;
    z{2}:=co;
    z{3}:=co;
    z{4}:=co+b1;
    alrededor <- z
end;
end INCLUDE
```

Fichero: cab_cmp.d1

```
\ Comparacion de borrosos r es el testigo (fijo) y b (variable)
\ Nucleos de testigos no solapan

function cmp_bor(BORROSO b, r): BOOLEANO
    function cmp_rea(REAL re1,re2):BOOLEANO
    begin
        cmp_rea <- re1 < re2
    end;
    function inter(REAL b1,b2,r1,r2):REAL
    var
       REAL x1,x2,x3,x4;
    begin
       x1 := b2 - b1 ;
       x2 := r2 - r1 ;
       x3 := b1 - r1  ;
       x4 := x3 / (x2 - x1) ;
       inter <- x4
    end;
    function tri6a(BORROSO b,r):BOOLEANO
    var
        REAL d1,d2,d3,d4;
        BOOLEANO v1;
    begin
        d1 := (b{4}-b{1});
        d2 := inter(b{4},b{3},r{1},r{2});
        d3 := inter(b{1},b{2},r{1},r{2});
        d3 := (d2 + d3) / 2. ;
        d4 := 1. - (d3 * d3) ;
        v1 := true ; GRADO(v1) := d4 ;
        tri6a <- v1
    end;
    function tri6b(BORROSO b,r):BOOLEANO
    var
        REAL d1,d2,d3,d4;
        BOOLEANO v1;
    begin
        d1 := (b{4}-b{1});
        d2 := inter(b{1},b{2},r{4},r{3});
        d3 := inter(b{4},b{3},r{4},r{3});
        d3 := (d2 + d3) / 2. ;
        d4 := 1. - (d3 * d3) ;
        v1 := true ; GRADO(v1) := d4 ;
        tri6b <- v1
    end;
    function tri3a(BORROSO b,r):BOOLEANO
    var
        REAL d1,d2,d3,d4;
        BOOLEANO v1;
    begin
        d1 := (b{4}-b{1});
        d2 := inter(b{4},b{3},r{1},r{2});
        d3 := d2 * (b{4} - r{1}) ;
        d4 := d3 / d1 ;
        v1 := true ; GRADO(v1) := d4 ;
        tri3a <- v1
    end;
    function tri4a(BORROSO b,r):BOOLEANO
    var
        REAL d1,d2,d3,d4;
        BOOLEANO v1;
```

```
    begin
        d1 := (b{4}-b{1});
        d2 := inter(b{1},b{2},r{4},r{3});
        d3 := d2 * (r{4} - b{1}) ;
        d4 := d3 / d1 ;
        v1 := true ; GRADO(v1) := d4 ;
        tri4a <- v1
    end;
    function tri4b(BORROSO b,r):BOOLEANO
    var
        REAL d1,d2,d3,d4, r1,r2,r4,r3;
        BOOLEANO v1;
    begin
        r1:=b{4};r2:=b{3};r3:=r{3};r4:=r{4};
        d1 := (b{4}-b{1});
        d2 := inter(b{4},b{3},r{4},r{3});
        d3 := d2 * (r{4} - b{4}) ;
        d4 := (d1 - d3) / d1 ;
        v1 := true ; GRADO(v1) := d4 ;
        tri4b <- v1
    end;
    function tri3b(BORROSO b,r):BOOLEANO
    var
        REAL d1,d2,d3,d4;
        BOOLEANO v1;
    begin
        d1 := (b{4}-b{1});
        d2 := inter(b{1},b{2},r{1},r{2});
        d3 := d2 * (r{1} - b{1}) ;
        d4 := (d1 - d3) / d1 ;
        v1 := true ; GRADO(v1) := d4 ;
        tri3b <- v1
    end;
var
    BOOLEANO nu22,nu33,so14,so41,so11,so44,test;
begin
\    print("b ",b, " r ",r,nl); \
    nu22 := cmp_rea(r{2},b{2});  \Comp nucleos
    nu33 := cmp_rea(b{3},r{3});  \
    so41 := cmp_rea(r{4},b{1});  \Comp soportes
    so14 := cmp_rea(b{4},r{1});  \
    so11 := cmp_rea(b{1},r{1});  \
    so44 := cmp_rea(b{4},r{4});  \

    if (so14 or so41) then \ Caso __|--b--|__ __|--r--|__ o contrario
       cmp_bor <- false
    else                  \ Soportes no disjuntos
       if so11 then
          if so44 then               \ Caso |---b---| (3)
             if (nu22 and nu33) then \           |--r--|
                test := tri3b(b,r) ;
                cmp_bor <- test
             else
                test := tri3a(b,r) ;
                cmp_bor <- test
             end if
          else                       \ Caso |----r----| (5)
             if (nu22 and nu33) then \        |--b--|
                test := tri3a(b,r) ; \ NO
                cmp_bor <- test
             else
                test := tri3b(b,r) ; \ NO
```

```
               cmp_bor <- test
            end if
         end if
      else
         if so44 then               \ Caso |---b---|(6)
            if (nu22 and nu33) then \       |--r--|
               test := true ;       \        (6c)
               cmp_bor <- test
            else
               if not(nu22) then
                  test := tri6a(b,r) ; \     (6a)
                  cmp_bor <- test
               else
                  test := tri6b(b,r) ; \     (6b)
                  cmp_bor <- test
               end if
            end if
         else                       \ Caso    |-b-| (4)
            if (nu22 and nu33) then \     |--r--|
               test := tri4b(b,r) ;
               cmp_bor <- test
            else
               test := tri4a(b,r) ;
               cmp_bor <- test
            end if
         end if
      end if
   end if
end ;
end INCLUDE
```

Fichero: d1.k

```
\       Programa prueba. Nucleos de muestra no solapados
\       Testa triangulares.
\       Necesita el fichero cab_cmp.d1
type Empleados = TIPO_LI
                 BORROSO emp
                 VA_LI
        bajo   : ()
                 const
                     t={0.0,0.0,40.0,60.0};
                 var
                     BOOLEANO va;
             begin
                 va:= cmp_bor(emp, t);
                 bajo <- va
             end |
        mediano: ()
                 const
                     t={40.0,60.0,100.0,120.0};
                 var
                     BOOLEANO va;
             begin
                 va:= cmp_bor(emp,t);
                 mediano <- va
             end |
        alto    : ()
                 const
                     t={100.0,120.0,200.0,200.0};
                 var
                     BOOLEANO va;
             begin
                 va:= cmp_bor(emp,t);
                 alto <- va
             end
             end,
     Factura = TIPO_LI
                 BORROSO fac
             VA_LI
             bajo: ()
                 const
                     t={0.0,0.0,10000.0,20000.0};
                 var
                     BOOLEANO va;
             begin
                 va:= cmp_bor(fac,t) ;
                 bajo <- va
             end |
             mediano: ()
                 const
                     t={10000.0,20000.0,80000.0,100000.0};
                 var
                     BOOLEANO va;
             begin
                 va:= cmp_bor(fac,t);
                 mediano <- va
             end |
             alto: ()
                 const
                     t={80000.0,100000.0,200000.0,200000.0};
                 var
                     BOOLEANO va;
```

```
            begin
                va:= cmp_bor(fac,t);
                alto <- va
            end
            end,
     Epresa = TIPO_LI
                Empleados e1;
                Factura f1
            VA_LI
            dar_credito: ()
                var
                    BOOLEANO b_f_d,b_f_m,b_f_g,b_e_j, b_e_m,b_e_v,test ;
            begin
                b_e_j := e1.bajo();
                b_e_m := e1.mediano();
                b_e_v := e1.alto();
                b_f_d := f1.bajo();
                b_f_m := f1.mediano();
                b_f_g := f1.alto();
\                print("em bajo=",b_e_j,"em mediano=",b_e_m,"em alto=",b_e_v,nl);
\                print("fa bajo=",b_f_d,"fa mediano=",b_f_m,"fa alto=",b_f_g,nl);
                CASE
                    b_e_j and b_f_d -> test:=false; dar_credito <- test
                [ ]  b_e_m and b_f_d -> test:=false; dar_credito <- test
                [ ]  b_e_v and b_f_d -> test:=false; dar_credito <- test
                [ ]  b_e_j and b_f_m -> test:=false; dar_credito <- test
                [ ]  b_e_m and b_f_m -> test:=false; dar_credito <- test
                [ ]  b_e_v and b_f_m -> test:=true; dar_credito <- test
                [ ]  b_e_j and b_f_g -> test:=false; dar_credito <- test
                [ ]  b_e_m and b_f_g -> test:=true; dar_credito <- test
                [ ]  b_e_v and b_f_g -> test:=true; dar_credito <- test
                ESAC
            end |
            no_credito : ()
                var
                    BOOLEANO b_f_d,b_f_m,b_f_g,b_e_j, b_e_m,b_e_v,test ;
            begin
                b_e_j := e1.bajo();
                b_e_m := e1.mediano();
                b_e_v := e1.alto();
                b_f_d := f1.bajo();
                b_f_m := f1.mediano();
                b_f_g := f1.alto();
                CASE
                    b_e_j and b_f_d -> test:=true; no_credito <- test
                [ ]  b_e_m and b_f_d -> test:=true; no_credito <- test
                [ ]  b_e_v and b_f_d -> test:=true; no_credito <- test
                [ ]  b_e_j and b_f_m -> test:=true; no_credito <- test
                [ ]  b_e_m and b_f_m -> test:=false; no_credito <- test
                [ ]  b_e_v and b_f_m -> test:=false; no_credito <- test
                [ ]  b_e_j and b_f_g -> test:=false; no_credito <- test
                [ ]  b_e_m and b_f_g -> test:=false; no_credito <- test
                [ ]  b_e_v and b_f_g -> test:=false; no_credito <- test
                ESAC
            end
            end;

var
     Epresa pe1 ;
     BOOLEANO bo1,bo3;
     ENTERO i,i1,i2,j,j1,j2 ;
```

```
begin
      j:= 0;
      i1:= 50; i2 := 60;
      j1 := 15000 ; j2 := 75000 ;
      while (j <= 1) do
         pe1.f1.fac := parecido(j1+(j*j2));
         i := 0;
         while (i <= 1) do
            pe1.e1.emp := alrededor(i1+(i*i2)) ;
            bo1:=pe1.dar_credito();
            print("Empleados = ",pe1.e1.emp," Factu. = ",pe1.f1.fac,nl);
            print("Dar credito ",bo1,nl);
            i := i + 1
            end do;
         j := j + 1
      end do
end.
```

La salida

```
<1>: Empleados = {45, 50, 50, 54}(1) Factu. = {12000, 15000, 15000, 19000}(1)
Dar credito FALSO(0.619)
<2>: Empleados = {45, 50, 50, 54}(1) Factu. = {12000, 15000, 15000, 19000}(1)
Dar credito FALSO(0.598)
<3>: Empleados = {45, 50, 50, 54}(1) Factu. = {12000, 15000, 15000, 19000}(1)
Dar credito FALSO(0.669)
<4>: Empleados = {45, 50, 50, 54}(1) Factu. = {12000, 15000, 15000, 19000}(1)
Dar credito FALSO(0.645)
<5>: Empleados = {105, 110, 110, 114}(1) Factu. = {12000, 15000, 15000, 19000}(1)
Dar credito CIERTO(0.619)
<6>: Empleados = {105, 110, 110, 114}(1) Factu. = {12000, 15000, 15000, 19000}(1)
Dar credito FALSO(0.598)
<7>: Empleados = {105, 110, 110, 114}(1) Factu. = {12000, 15000, 15000, 19000}(1)
Dar credito FALSO(0.669)
<8>: Empleados = {105, 110, 110, 114}(1) Factu. = {12000, 15000, 15000, 19000}(1)
Dar credito FALSO(0.645)
<9>: Empleados = {165, 170, 170, 174}(1) Factu. = {12000, 15000, 15000, 19000}(1)
Dar credito CIERTO(0.784)
<10>: Empleados = {165, 170, 170, 174}(1) Factu. = {12000, 15000, 15000, 19000}(1)
<11>: Empleados = {45, 50, 50, 54}(1) Factu. = {87000, 90000, 90000, 94000}(1)
Dar credito CIERTO(0.594)
<12>: Empleados = {45, 50, 50, 54}(1) Factu. = {87000, 90000, 90000, 94000}(1)
Dar credito FALSO(0.573)
<13>: Empleados = {45, 50, 50, 54}(1) Factu. = {87000, 90000, 90000, 94000}(1)
Dar credito FALSO(0.615)
<14>: Empleados = {45, 50, 50, 54}(1) Factu. = {87000, 90000, 90000, 94000}(1)
<15>: Empleados = {105, 110, 110, 114}(1) Factu. = {87000, 90000, 90000, 94000}(1)
Dar credito CIERTO(0.594)
<16>: Empleados = {105, 110, 110, 114}(1) Factu. = {87000, 90000, 90000, 94000}(1)
Dar credito CIERTO(0.573)
<17>: Empleados = {105, 110, 110, 114}(1) Factu. = {87000, 90000, 90000, 94000}(1)
Dar credito CIERTO(0.615)
<18>: Empleados = {105, 110, 110, 114}(1) Factu. = {87000, 90000, 90000, 94000}(1)
<19>: Empleados = {165, 170, 170, 174}(1) Factu. = {87000, 90000, 90000, 94000}(1)
Dar credito CIERTO(0.752)
<20>: Empleados = {165, 170, 170, 174}(1) Factu. = {87000, 90000, 90000, 94000}(1)
<21>: Empleados = {45, 50, 50, 54}(1) Factu. = {162000, 165000, 165000, 169000}(1)
Dar credito CIERTO(0.789)
<22>: Empleados = {45, 50, 50, 54}(1) Factu. = {162000, 165000, 165000, 169000}(1)
<23>: Empleados = {105, 110, 110, 114}(1) Factu. = {162000, 165000, 165000, 169000}(1)
Dar credito CIERTO(0.789)
<24>: Empleados = {105, 110, 110, 114}(1) Factu. = {162000, 165000, 165000, 169000}(1)
```

```
<25>: Empleados = {165, 170, 170, 174}(1) Factu. = {162000, 165000, 165000, 169000}(1)
Dar credito CIERTO(1)
```

Bibliografía

[1] J.M. Adamo. L.p.l. a fuzzy programming language: 1. syntactic aspects. *Fuzzy Set and Systems*, 3:151–179, 1980.

[2] J.M. Adamo. L.p.l. a fuzzy programming language: 2. semantic aspects. *Fuzzy Set and Systems*, 3:261–289, 1980.

[3] Claudi Alsina. A primer of t-norms. *Proc. F.I.S.A.L.-83 Universitat de Palma de Mallorca*, pages 27–36, 1983.

[4] K.R. Apt and G.D. Plotkin. A cook's tour of countable nondeterminism. *LNCS*, 115:479–494, 1981.

[5] J.F. Baldwin and S.Q. Zhou. A fuzzy relational inference language. *Fuzzy Sets and Systems*, 14:155–174, 1984.

[6] Henk Barendregt. Pairing witout conventional restraints. *Zeitschr. f. math. Logik und Grundlagen d. Math.*, 20:289–306, 1974.

[7] H.P. Barendregt. *The Lambda calculus its syntax and semantics.* North-Holland, Amsterdam, 1985.

[8] H.J. Boehm and R. Cartwright. Exact real arithmetic: Formulating real number as functions. *In Research Topics in Funcional Programming. (Ed. D. Turner). Addison-Wesley. pag.43-64*, 1990.

[9] E. Trillas C. Alsina and L. Valverde. On some logical connetives for fuzzy set theory. *Journal of Mathematical Analysis and Applications*, 93:15,26, 1983.

[10] P.-T. Chang and E.S. Lee. Fuzzy arithmetics and comparison of fuzzy numbers. *En Fuzzy Optimization. (Ed. M.Delgado,J.Kacprzyk,J-L. Verdegay, M.A.Vila). Springer Verlag*, 1994.

[11] Shi-Kuo Chang. On the execution of fuzzy programs using finite-states machines. *IEEE Transactions on Computers*, 3:241–253, 1972.

[12] E.W. Chapin. Set-valued set theory: part one. *Notre Dame Journal of Formal Logic*, XV-4:619–634, 1974.

[13] Buenaventura Clares. Una introducción a la W-calculabilidad. Tesis Doctoral, Universidad de Granada, 1983.

[14] David F. Clark and Abraham Kandel. Halo - a fuzzy programming language. *Fuzzy Sets and Sistems*, 44:199–208, 1991.

[15] J. Cuena. Las técnicas fuzzy y el futuro del software. *In Fundamentos e Introducción a la Ingeniería "Fuzzy" (Ed. Enric Trillas). Omron Electronics S.A. pag.131-156*, 1994.

[16] Haskell B. Curry and Robert Feys. *Lógica combinatoria*, volume 1. Editorial Tecnos, S.A., Madrid, 1967.

[17] Haskell B. Curry, J. Roger Hindley, and Jonathan P. Seldin. *Combinatory Logic*, volume 2. North-Holland, Amsterdam, 1972.

[18] Saumya K. Debray and Prateek Mishra. Denotational and operational semantics for prolog. *The Journal of Logic Programming*, 5:61–91, 1988.

[19] M. Delgado. Razonamiento aproximado. *En Algunos aspectos del tratamiento de la Información en Inteligencia Artificial. (Ed. Universidad de Granada).*, pages 113,133, 1991.

[20] Miguel Delgado Calvo-Flores y M.A. Vila Miranda. Software difuso. *En Fundamentos e Introducción a la Ingeniería "Fuzzy" (Ed. Enric Trillas). Omron Electronics S.A. pag.115-130*, 1994.

[21] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, New Jersey, 1976.

[22] V. Donzeau-Gouge, G. Khan, and B. Lang. On the formal definition of ada. *LNCS*, 94:475–489, 1980.

[23] D. Dubois and H. Prade. *Fuzzy Set and System: Theory and applications.* Academic Press, New York, 1980.

[24] Abbas Edalat and Philipp Sünderhauf. A domain-theoretic approach to computability on the real line. *Theoretical Computer Science*, 210:73–98, 1999.

[25] M.H. Escardó. PCF extended whith real numbers. *Theoretical Computer Science*, 162:79–115, 1996.

[26] Brian R. Gaines and Ladislav J. Kohout. Logic of automata. *Int. J. Gen. Syst.*, 2:191–208, 1976.

[27] Giangiacomo Gerla. Turing l-machines and recursive computability for l-maps. *Studia Logica*, XLVIII,no. 2:179–192, 1989.

[28] Joseph Goguen and Grant Malcolm. *Algebraic Semantics of Imperative Programs.* The MIT Press, Cambridge, Massachusetts, 1996.

[29] I. Grattan-Guinness. Fuzzy membership mapped onto intervals and many-valued quantities. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 22:149–160, 1976.

[30] C.A. Gunter and D.S. Scott. Semantics domains. *In Formals Models and Semantics. (Ed Jan van Leewen). Elsewier. pag.633-674*, 1990.

[31] Carl A. Gunter. *Semantics of Programming Languages. Structures and Techniques.* The MIT Press, Cambridge, Massachusetts, 1992.

[32] M.M. Gupta and J. Qi. Theory of t-norms and fuzzy inference methods. *Fuzzy Set and Systems*, 40:431–450, 1991.

[33] Matthew Hennessy. *Algebraic Theory of Processes.* The MIT Press, Cambridge, Massachusetts, 1988.

[34] IEEE. IEEE Standard 754 for Binary Floating-Point Arithmrtic. *SIGPLAN*, 22(2):9–25, 1985.

[35] J.-L.Castro. La lógica en la inteligencia artificial. *En Algunos aspectos del tratamiento de la Información en Inteligencia Artificial. (Ed. Universidad de Granada).*, pages 93,111, 1991.

[36] Arnold Kaufmann and Madan M. Gupta. *Introduction to fuzzy arithmetica. Theory and Applications.* Van Nostrand Reinhold Company, New York, 1985.

[37] Peter Kornerup and David W. Matula. Finite precision lexicographic continued fraction number system. *In Procedings of the 7^{th} IEEE Symposium on Computer Arithmetic,pag 207-214Urbana,1995. IEEE Computer Society Press*, 1995.

[38] Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.

[39] Jean-Louis Krivine. *Lambda-calcul, types et modelès.* Masson, Paris, 1990.

[40] René Lalement. *Computation as Logic.* Prentice Hall, Cambridge, 1993.

[41] E.T. Lee and L.A. Zadeh. Note on fuzzy languages. *Inf. Sci.*, 1:421–434, 1969.

[42] Xavier Leroy. *The Objective Caml system, release 1.07. Documentation and user's manual*, volume disponible en `http://pauillac.inria.fr./caml/`. INRIA, 1997.

[43] Jean-Jacques Lévy. Réductions correctes et optimales dans le lambda-calculo. Tesis Doctoral, Universite Paris VII, 1978.

[44] A. De Luca and S. Termini. Algebraic properties of fuzzy sets. *Journal of Mathematical Analysis and Applications*, 40:373–386, 1972.

[45] Zohar Manna. The correctness of nondeterministic programs. *Artificial Intelligence*, 1:1–26, 1970.

[46] T.P. Martin, J.F. Baldwin, and B.W. Pilsworth. The implementation of fprolog - a fuzzy prolog interpreter. *Journal of Mathematical Analysis and Applications*, 85:409–451, 1982.

[47] Greg Michaelson. *An Introduction to Functional Programing through Lambda Calculus.* The MIT Press, Cambridge, Massachusetts, 1989.

[48] M. Mizumoto, J. Toyoda, and K. Tanaka. Fuzzy languages. *Syst. Comput. Control*, 1:36–43, 1969.

[49] M. Mizumoto, J. Toyoda, and K. Tanaka. General formulation of formal grammars. *Inf. Sci.*, 4:87–100, 1972.

[50] Rafael Morales. Calculabilidad difusa mediante gramáticas. Diseño de un lenguaje imperativo de programación difusa. Tesis Doctoral, Universidad de Málaga, 1991.

[51] P.D. Moses. Denotational semantics. *In Formals Models and Semantics. (Ed Jan van Leewen). Elsewier. pag.575-629*, 1990.

[52] Ketan Mulmuley. *Full Abstraction and Semantic Equivalence.* The MIT Press, Cambridge, Massachusetts, 1987.

[53] Devendra S. Negi. Fuzzy analysis and optimization. Tesis Doctoral, Kansas State University, 1989.

[54] Hanne Riis Nielson and Flemming Nielson. *Semantics whith Applications. A Formal Introduction.* John Wiley, Chichester, 1992.

[55] Olufemi O. Oguntade. Semantics and pragmatics of fuzzy sets and systems. *Fuzzy Sets and Systems*, 6:119–143, 1981.

[56] W. Ostasiewicz. A new approach to fuzzy programming. *Fuzzy Sets and Sistems*, 7:139–152, 1982.

[57] Mario Pérez Jiménez. *Teoría de clases y conjuntos.* EDUNSA, Barcelona, 1988.

[58] Simon L. Peyton Jones. Arbitrary precision arithmetic using continued fractions. *INDRA Note 1530, University College London*, 1984.

[59] Wolfgang Polak. *Compiler Specificaction and Verification.* Springer-Verlag, Berlin, 1981.

[60] P.J. Potts and A. Edalat. Exact real computer arithmetic. *Departament of Computing Technical Report DOC97/9, Imperial College*, disponible en `http://www-tfm.ic.ac.uk/~pjp`, 1997.

[61] G. Révész. A list-oriented extension od the lambda-calculus satisfying the Church-Rosser theorem. *Theoretical Computer Science*, 93:75–89, 1992.

[62] Daniel Sanchez Alvarez. El lambda cálculo_b. *En Actas del VII Congreso Español sobre Tecnología y Lógica Fuzzy. pag.311-316*, 1997.

[63] Daniel Sanchez Alvarez y Antonio F. Gómez Skarmeta. Semánticas de lenguajes con datos borrosos. *En Actas del VIII Congreso Español sobre Tecnología y Lógica Fuzzy. pag.271-278*, 1998.

[64] E.S. Santos. Fuzzy algorithms. *Information and Control*, 17:326–339, 1970.

[65] E.S. Santos. Context-free fuzzy languages. *Inf. Control*, 26:1–11, 1974.

[66] E.S. Santos. Fuzzy and probabilistic programs. *Information and Control*, 10:331–335, 1976.

[67] E.S. Santos. Fuzzy automata and languages. *Inf. Sci.*, 10:193–197, 1976.

[68] D.A. Schmidt. *Denotational Semantics. A Methodology for Language Development.* Wm. C. Brown Publishers, Dubuque, Iowa, 1988.

[69] Kurt J. Schmucker. *Fuzzy set, natural language computations, and risk analysis.* Computer Science Press, Maryland, 1984.

[70] D.S. Scott. Domains for denotational semantics. *Lecture Notes in Computer Science*, 140, 1982.

[71] Z. Shen, L. Ding, and M. Mukaidono. Fuzzy resolution principle. *Proceeding of the XVIII International symposium on multiple value logic*, 1:210–215, 1988.

[72] M.B. Smyth. Power domains. *Journal of computer and system sciences*, 16:23–36, 1978.

[73] J.E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, Cambridge.Mass., 1977.

[74] R.D. Tennent. A denotational definition of the programing language pascal. *Programmig Research Group Memorandum, Oxford University*, 1978.

[75] R.D. Tennent. *Principles of Programming Languages.* Prentice-Hall, Englewood Cliffs,N.J., 1981.

[76] M.G. Thomason. Fuzzy syntax-directed translation. *J. Cybern*, 4:87–94, 1974.

[77] E. Trillas. Lógica borrosa. *En Fundamentos e Introducción a la Ingeniería "Fuzzy" (Ed. Enric Trillas). Omron Electronics S.A. pag.65-92*, 1994.

[78] E. Trillas and L. Valverde. On implication and indistinguishability in the setting of fuzzy logic. *En Management decision support using fuzzy sets and possibility theory. (Ed. J.Kacprzyk,R.R. Yager). Verlag TUV*, pages 198,212, 1985.

[79] Steven Vickers. *Topology Via Logic.* Cambridge University Press, Cambridge, 1989.

[80] J. Vuillemin. Exact real computer arithmetic with continued fractions. *IEEE Transaction on computer*, 39:1087–1105, 1990.

[81] Siegfried Weber. A general concept of fuzzy connectives, negations and implications based on t-norms and t-conorms. *Fuzzy Set and Systems*, 11:115–133, 1983.

[82] E. Willaeys and N. Malvache. Utilisation d'un referenciel de sous-ensembles flous. aplications à un algoritme flou. *Congreso. International Conference on System Science. Wroclaw. Poland*, 1976.

[83] G. Winskel and K. Larsen. Using information systems to solve recursive domain equations effectively. *Lecture Notes in Computer Science*, 173, 1984.

[84] Glyn Winskel. *The Formal Semantics of Programming Languages.* The MIT Press, Cambridge, Massachusetts, 1993.

[85] L.A. Zadeh. Fuzzy sets. *Inform. Control*, 8:338–353, 1965.

[86] L.A. Zadeh. Fuzzy algorithms. *Information and Control*, 15:94–102, 1968.

[87] L.A. Zadeh. The concept of a linguistic variable and its applications to approximate reasoning-I. *Information Sciences*, 8:199–249, 1975.

[88] L.A. Zadeh. The concept of a linguistic variable and its applications to approximate reasoning-II. *Information Sciences*, 8:301–357, 1975.

[89] L.A. Zadeh. The concept of a linguistic variable and its applications to approximate reasoning-III. *Information Sciences*, 9:43–80, 1975.

[90] L.A. Zadeh. Fuzzy sets as a basic for a theory of posibility. *Fuzzy Set and Systems*, 1:3–28, 1978.

[91] L.A. Zadeh. Pruf - a meaning representation language for natural languages. *In Fuzzy Reasonig and its Applications. (Ed E. Mamdani and B. Gaines). Academic Press*, 1981.

[92] Ma Zherui and Wu Wangming. Logical operators on complete lattices. *Information Science*, 55:77–97, 1991.

Índice de figuras

Índice de cuadros

Índice alfabético