Resumen

Los sistemas interactivos por naturaleza interactúan dinámicamente con el entorno durante el transcurso del tiempo. Tal comportamiento escapa del dominio estático de los formalismos tradicionales, por lo que es una necesidad imperante el contar con métodos formales de especificación para comprender, predecir y controlar el comportamiento de los sistemas interactivos. Esta necesidad nos dirige a la búsqueda de nuevos formalismos que capturen adecuadamente el comportamiento dinámico de los sistemas interactivos y lo conduzcan hacia objetivos claramente establecidos. En particular, estableciendo normas o legislando patrones de comportamiento con respecto a los recursos disponibles (alcanzables) en el sistema.

Uno de tales formalismos es la *Lógica Lineal*, la cual es una lógica dinámica consciente de los recursos disponibles en un entorno: en términos generales considera a las afirmaciones como recursos que son consumidos mediante la aplicación de reglas de inferencia de tal forma que tales afirmaciones sólo pueden ser usadas una vez, capturando de esa manera la semántica de flujos interactivos de datos o paso de mensajes.

La esencia de la Lógica Lineal radica en la profunda simetría que es otorgada por el operador de dualidad. Tal simetría se distingue en los 2 tipos de conjunciones y disyunciones, los cuales son más primitivos que los existentes en los formalismos clásicos. Estos conectivos permiten expresar diferentes tipos de interacción entre las entidades relacionadas. Aunado a ello, la Lógica Lineal distingue entre recursos de uso ilimitado y lineal.

Como consecuencia a las cualidades antes señaladas, la Lógica Lineal permite establecer modelos abstractos de cómputo que son aprovechados para propósitos de Especificación Formal.

Uno de esos modelos corresponde al área computacional que identifica un cálculo con una búsqueda de pruebas.

Gracias a las propiedades sintácticas de las pruebas en Lógica Lineal, se establece una estrategia general de búsqueda, la cual conduce a un sistema de inferencia que aprovecha tales propiedades para evitar la búsqueda de pruebas redundantes. Para ello se establece una restricción sintáctica sobre las fórmulas, permitiendo transformarlas a una forma normal, en la cual, la búsqueda de pruebas resulta más eficaz. La forma normal se conoce como LINLOG y el sistema de inferencia corresponde al *Sistema Triádico*, ambos propuestos por Jean Marc Andreoli.

Se presenta un prototipo para especificación formal de sistemas interactivos basado en LINLOG, utilizando fórmulas proposicionales exclusivamente.

Contenido

Introducción. 1

I Fundamentos 6

1 De la Lógica Clásica a la Lógica Lineal. 7
1.1 Fórmulas como Recursos . . . 7
1.2 Cálculo de Secuentes Clásico . . . 8
1.2.1 Reglas Identidad . . . 9
1.2.2 Reglas Estructurales . . . 9
1.2.3 Reglas Lógicas . . . 9
1.3 Debilitamiento y Contracción: malos contadores. . . . 11
1.4 Dos Tipos de Conjunción y Disyunción . . . 11
1.5 Negación Lineal . . . 13
1.6 Exponenciales . . . 14
1.6.1 Estados y Transiciones . . . 15
1.7 Semántica Informal . . . 16
1.8 Fragmentos . . . 17

2 Cálculo de Secuentes para la Lógica Lineal y Elementos para Búsqueda de Pruebas 18
2.1 Cálculo de Secuentes Lineal (asimétrico por la derecha) . . . 18
2.2 Elementos para la Búsqueda de Pruebas. . . . 19
2.3 Comportamiento de las Reglas. . . . 19
2.3.1 Reglas Multiplicativas: . . . 20
2.3.2 Reglas Aditivas: . . . 20
2.3.3 Reglas utilizando Exponenciales: . . . 20
2.3.4 Reglas para los Cuantificadores: . . . 21
2.4 Procedimiento de Búsqueda Básico. . . . 21
2.5 Ejemplos de Búsqueda de Pruebas . . . 22

3 Teoría de Pruebas, Semántica y Resultados de Complejidad 24
3.1 Teoría de Pruebas . . . 24
3.1.1 Propiedades Importantes de la Búsqueda de Pruebas . . . 25
3.2 Resultados de Complejidad . . . 28

3.3 Semántica 30
3.4 Aplicaciones 31
3.4.1 Programación Funcional 31
3.4.2 Programación Lógica 32
3.5 Hacia la unidad de la Lógica 33

4 Búsqueda de Pruebas en la Lógica Lineal **35**
4.1 Pruebas Enfocadas 35
4.1.1 Sistema de Inferencia *Diádico* 35
4.1.2 Sistema de Inferencia *Triádico* 36
4.2 La forma normal LINLOG 41
4.2.1 Sintaxis y Semántica Operacional 41
4.2.2 Normalización a la forma LINLOG 44

II Diseño, Implantación y Pruebas **48**

5 Diseño e Implantación **49**
5.1 Consideraciones Generales 49
5.2 Diseño 49
5.3 Implantación 51
5.3.1 Definición de los conectivos 51
5.3.2 Preprocesamiento 52
5.3.3 Normalización 55
5.3.4 Búsqueda de Pruebas 57
5.3.5 Comentarios adicionales 62

6 Pruebas y Resultados **63**
6.1 Demostración automática de Fórmulas 63
6.2 Interacción entre procesos mediante comunicación asíncrona: *ACL* 65
6.2.1 ACL en LINLOG 67
6.3 Herencia interconstruída en Objetos 69

7 Trabajo Relacionado **70**

8 Conclusión y Trabajo a Futuro **72**
8.1 Conclusión 72
8.2 Problemas Abiertos 72
8.3 Trabajo a Futuro 73

Bibliografía **76**

Agradecimientos

Deseo agradecer a varias personas que me apoyaron en la realización de este trabajo. Primero a mis padres Jovita Pérez y Mario Palacios, por brindarme la vida y la oportunidad de hacer de cada día un nuevo reto, una vía contínua de aprendizaje, trabajo y amor. A mis profesores, por infundirme la semilla de inquietud, el coraje y los deseos firmes por redituarles lo mucho que he recibido, continuando con su ejemplo para con las siguientes generaciones. A los miembros del Jurado: M.C. Jesús García F., M.C. Héctor Jiménez S., M.C. Guillermo De Ita L., Dr. Mauricio Osorio G. y Lic. Jesús Lavalle M. por sus invaluables observaciones y críticas a esta tesis, gracias a las cuales quedó mucho mejor escrita. Y al final pero no menos importante, a mi querido amigo y asesor José de Jesús Lavalle Martínez, por el inmenso caudal de conocimiento y formación, por mostrarme la vida desde una perspectiva distinta y otorgarme (aunque no lo creas!) una inquebrantable fé...

Also, I wish be grateful with a group of people I've meet by the internet, whose helpme a lot in this investigation:

- Miss Judith Burgess, from Library of Computer Science Laboratory, SRI International, USA.
- Dr. Carolyn Talcott, from Stanford University, USA.
- Dr. Vladimir Alexiev, from University of Alberta, CA.
- Dr. Jean-Marc Andreoli, from Ranx Xerox Research Center, France.
- MsC. Christian Urban, from Computational Logic Group at St Andrews University Division of Computer Science, England.

Thank you very much for the papers inconditionally bring me, for the useful discussions about my multiple questions and more important, for show me the beauty in the scientific work. I'll me best effort!

BENEMÉRITA UNIVERSIDAD AUTONOMA DE PUEBLA.
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN.

Especificación Formal de Sistemas Interactivos mediante Lógica Lineal.

TESIS

Que para obtener el título de

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

Presenta:

José Juan Palacios Pérez

Asesor :
Lic. José de Jesús Lavalle Martínez

Puebla, Pue. 1997.

Introducción

La solución de los problemas que actualmente enfrentan las ciencias computacionales exige necesariamente expresar *interacción dinámica con el entorno* y *sensibilidad al paso del tiempo*.

Ejemplo de tales problemas son los siguientes:

- Entornos Concurrente, Paralelo y Distribuido

  Para entornos del futuro cercano de procesamiento masivamente paralelo, y por ende, distribuido, los lenguajes de programación concurrente basados en comunicación asíncrona se tornan más y más importantes. Debido a la dificultad de escribir y depurar programas en tales ambientes, las computadoras deberán ayudar al programador para el mantenimiento y verificación del software, de ahí que el rol de los *Entornos Formales* para cómputo concurrente y distribuido sean tan importantes [YT89].

- Expresar adecuadamente comportamiento dinámico

  Resulta esencialmente importante poder expresar *acción* y *cambio* de manera consistente en los sistemas interactivos.

  Cuando las lógicas son consideradas (a nivel fundamental) como formas rigurosas de razonar correctamente acerca de algunos aspectos de la realidad, no hay razón por la que los objetos involucrados en el razonamiento no puedan cambiar; claro, el razonamiento relevante podría tratar precisamente con tal cambio [MMO92].

- Integración entre paradigmas:

  La Programación Concurrente Orientada a Objetos es un área muy activa de investigación. Una de las razones importantes de este interés es la *naturalidad* con la cual este estilo de programación puede modelar interacción entre objetos; esta naturalidad tiene origen en las ideas seminales del lenguaje Simula, el cual fue diseñado explícitamente para simular interacciones concurrentes entre objetos en el mundo real [YT89, MFW92].

  Las propuestas recientes para tal integración carecen de una base semántica clara y completamente aceptada. Considerablemente, conflictos serios entre el mecanismo de herencia y el desempeño concurrente, como la llamada *anomalía de la Herencia* (la cual obliga la redefinición de los mensajes heredados para mantener la integridad del sistema concurrente), retrasan una adecuada integración de los paradigmas [MFW92].

  A su vez, y debido principalmente a la convergencia entre lógica y computación (identidad entre los conceptos de cómputo y deducción), las fronteras entre sistemas lógicos, lenguajes declarativos y demostradores de teoremas es cada vez más difusa, con cada área siendo influída e influyendo en las demás. De especial importancia es establecer fundamentos para apoyar la integración entre los paradigmas Lógico (Funcional y Relacional) con la Orientación a Objetos para ofrecer entornos más robustos [MMO93, Mil89].

Por tal motivo, la tecnología computacional está tornándose cada vez más interactiva : los sistemas dejan de ser cerrados y reducibles mediante algoritmos y pasan a ser **sistemas interactivos abiertos**, cuyo comportamiento es crecientemente observado y especificado de manera incompleta en el sentido de incompletez de Gödel [Weg95].

Como consecuencia, resulta esencial proporcionar sustento formal a tales sistemas.

Sin embargo, los modelos formales tradicionales, tanto los declarativos (Cálculo Lambda, Cálculo de Predicados de Primer Orden) como los imperativos (Máquinas de Turing) resultan insuficientes para especificar la interacción de un agente con su entorno en el transcurso del tiempo; debido a que parten de la concepción de un dominio de modelado estático y eterno en el sentido platónico y no pueden representar adecuadamente el comportamiento dinámico abierto, sensible al tiempo y por tanto evolutivo de los sistemas interactivos.

Ante ello, nos preguntamos cómo adaptar o aprovechar a los modelos y paradigmas formales para enfrentarnos al reto que representan los sistemas interactivos. Necesariamente enfocamos nuestra atención en la definición formal misma de *Modelo* :

Un **Modelo** [Weg95] $M = (W, R, I)$ es una representación sintáctica R de un mundo (dominio) de Modelado W (componente semántico) y un componente pragmático (intérprete) I el cual captura las propiedades de los elementos de W mediante interpretaciones $I(r)$ de las representaciones $r \in R$.

Los modelos computacionales, por su parte, expresan propiedades mediante el comportamiento de los programas: aprendemos y razonamos acerca de W através de la ejecución de programas en M. Tres paradigmas principales son distinguibles [Weg95]:

- *Declarativo:* especifica qué es calculado independientemente de cómo es calculado.
- *Imperativo:* especifica un cálculo operacionalmente, mediante un modelo de máquina abstracta.
- *Interactivo:* especifica la interacción de un agente con su entorno durante el transcurso del tiempo.

Comparativamente, los paradigmas de modelado declarativo e imperativo se sustentan en el Mundo de Modelado Matemático (Clásico), el cual es estático (denotaciones fijas), único y cerrado (un cálculo es realizado independientemente del mundo exterior: la máquina de Turing se desconecta del entorno). Por su parte, en el mundo de modelado interactivo las propiedades semánticas trascienden a sus propiedades sintánticas, nuevo conocimiento y denotaciones son introducidas durante el curso de un cómputo, lo que les permite evolucionar dinámicamente: de ahí que sean abiertos. Además, cada máquina que expresa comportamiento interactivo tiene su propio dominio de modelado (dado que el entorno influye en ella), a diferencia de las máquinas de Turing que todas ellas tienen el mismo dominio de las funciones recursivas.

El problema consiste en cómo permitir que los modelos clásicos expresen interacción. Para tal fin, hay en general 2 estrategias:

1. **Primero el dominio del Modelo:** atender la parte Semántica, esto es,
   - Definir un *nuevo* Modelo con su dominio correspondiente,
   - *Refinar* el Modelo,

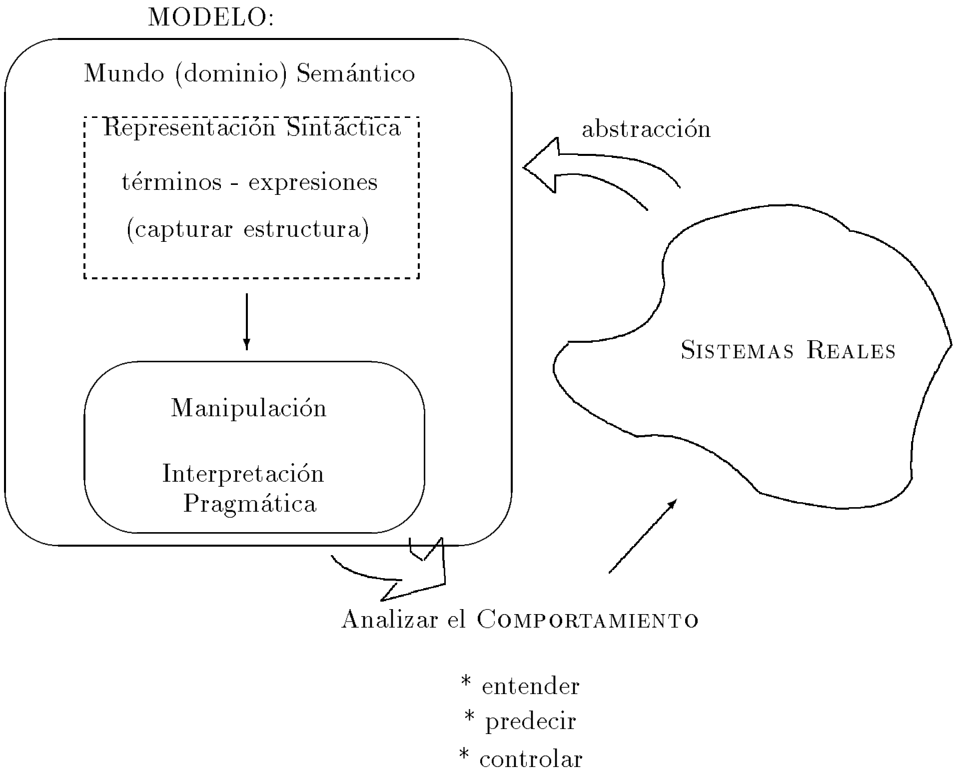

Figura 0.1: El entorno interdisciplinario de Modelado.

- *Extender* el Modelo,

2. **Primero el lenguaje del Modelo:** modificar los elementos sintácticos mediante *construcciones (ad hoc)* que expresen el comportamiento deseado.

Siguiendo la primer estrategia, las máquinas de Turing pueden ser *extendidas* para que sean interactivas adicionando *acciones de entrada y salida (sensores lógicos)*, es decir, que tanto soporten influencia externa durante el curso de un cálculo, como que ellas puedan correspondientemente afectar en él, de tal forma que [Weg95]:

- interactúan con el entorno externo, tornándose sistemas abiertos
- modelan el paso del tiempo, ya que los eventos externos afectan su comportamiento,
- capturan el comportamiento de las Máquinas de Turing con Oráculos,
- son incompletas en el sentido lógico de Gödel: nunca se observará todo su comportamiento.

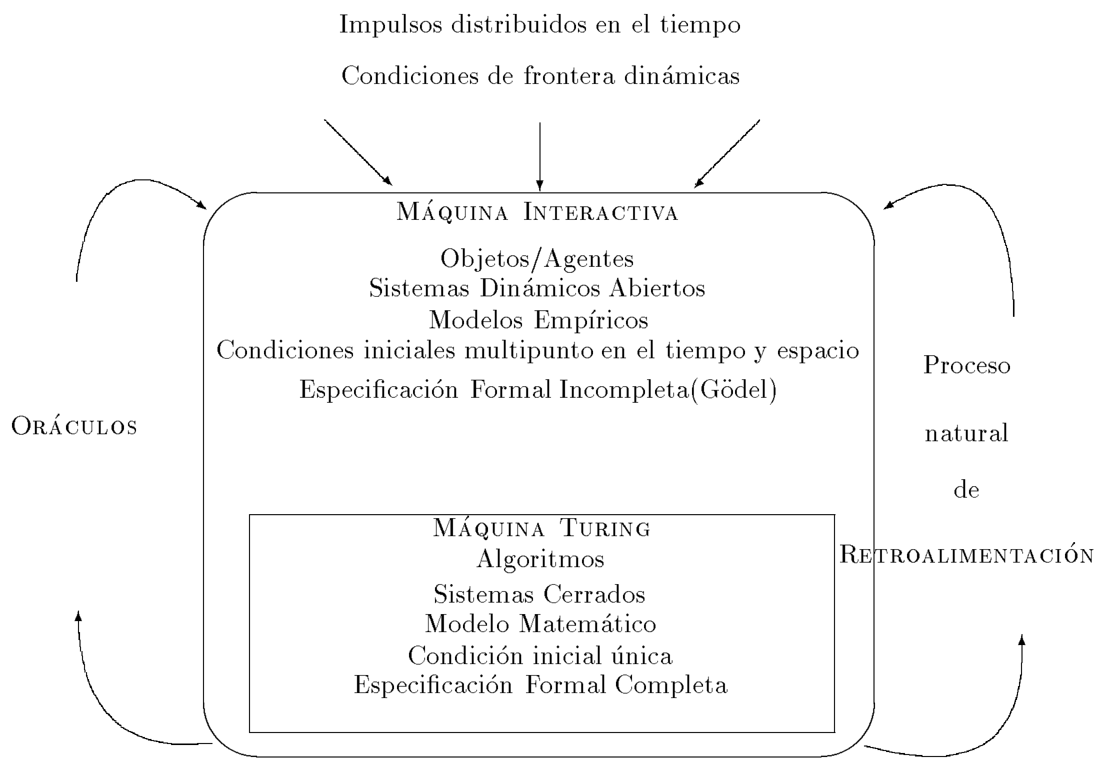

Figura 0.2: De la Máquina de Turing a la Máquina de Interacción.

Esta simple extensión las transforma de sistemas estáticos cerrados a sistemas dinámicos abiertos [Weg95].

Una vez propuesto el Modelo, es necesario conocer las características esenciales que la **Especificación Formal** de los sistemas interactivos presenta.

Necesariamente, el precio a pagar en virtud de mayor expresividad será la pérdida de *completez*: las propiedades semánticas de los mundos de modelado interactivos son más ricos que la notación sintáctica por la cual son representados, el comportamiento dinámico que expresan es crecientemente observado y nunca será completamente capturado [Weg95].

No obstante, tal comportamiento dinámico de los sistemas interactivos puede ser **parcialmente especificado** através de restricciones en las *interfaces* (conexiones entre los agentes), vistas y modos de uso, acotando y conduciendo tal comportamiento hacia propósitos de interés y utilidad [Weg95].

Particularmente importante, es establecer y legislar tales restricciones en base a los *recursos* disponibles en el sistema.

De esa manera, los sistemas y lenguajes de especificación se desplazan de ser formalismos matemáticamente precisos hacia formalismos que ofrecen especificaciones de naturaleza no exclusivamente matemática, sino esencialmente lógica [MNSP91].

Consecuentemente, los aspectos deseados para los lenguajes de especificación son, entre otros [Chi95]:

- Bastante ricos (en su expresividad)para especificar modularidad imperativa, funcional y concurrente.
- Especificación modular de las diferentes características de los lenguajes de programación.
- Uso de la *meta-teoría* del lenguaje de especificación: que tanto permita estudiar las transformaciones de los programas como verificar la correctez de las implantaciones de tales especificaciones.

Con esos objetivos, el formalismo que ha sido elegido en este trabajo es la **Lógica Lineal**: fundamentalmente, es una lógica dinámica que toma cuidado del uso de los recursos. Sus propiedades expresivas son esencialmente duales, (es decir, complementarias entre sí) permitiéndoles capturar diversas formas de comportamiento interactivo. Este formalismo se obtiene a partir de un *refinamiento* a la lógica clásica, lo cual permite que su presentación no imponga un cambio radical en los esquemas mentales para los lectores interesados.

El objetivo de esta tesis es el estudio fundamental y suficiente de la Lógica Lineal para la implantación de un prototipo demostrador de fórmulas proposicionales. El prototipo constituye una herramienta básica para la experimentación de las características generales de la Lógica Lineal y para el estudio de las bondades de la Lógica Lineal para propósitos de especificación formal de sistemas interactivos.

Esta tesis está organizada como sigue: en la primera parte se exploran los fundamentos del formalismo propuesto para sustentar la especificación formal de sistemas interactivos: la Lógica Lineal. En el Cap.1 se presenta a la Lógica Lineal mediante un refinamiento al enfoque clásico: su reglas de inferencia y significado dentro del cálculo de secuentes. Tal presentación es completada en el Cap.2, estableciendo los elementos principales del entorno de búsqueda de pruebas por su utilidad como paradigma computacional. En el Cap.3 se revisan los entornos de la Teoría de Pruebas, Semántica, los resultados de Complejidad y las áreas generales de aplicación en los ámbitos computacionales. El capítulo concluye con la pregunta: ¿es posible una lógica única?, la cual es resultado de las consideraciones respecto a la utilización de la Lógica Lineal como Entorno lógico.

En la segunda parte se exploran con mayor detalle las estrategias de búsqueda de pruebas con el objetivo de obtener la implantación de un demostrador de teoremas.

En el Cap.4 se presenta a la clase de pruebas normal conocida como Pruebas Enfocadas y el sistema de inferencia que las realiza, el sistema Triádico; y como consecuencia, a una forma normal en las fórmulas, en donde las pruebas enfocadas tienen una interpretación computacional importante: la forma normal LINLOG.

En el Cap.5 se presenta el diseño e implantación del programa buscador fundamentado en el cap. anterior; el cual es sometido a casos de prueba en el Cap.6. El trabajo relacionado, bajo el enfoque de *Pruebas Uniformes* es presentado en el Cap.7. Finalmente, en el Cap.8 se establecen las conclusiones, las consideraciones finales y el trabajo a futuro.

Parte I

Fundamentos

Capítulo 1

De la Lógica Clásica a la Lógica Lineal.

Siendo consistentes con la estrategia señalada (Primero el Modelo), se presentará a la Lógica Lineal como formalismo originado mediante un refinamiento al enfoque clásico atendiendo problemas específicos que éste padece.

La Lógica Lineal fue desarrollada por Jean-Yves Girard alrededor de 1986, motivada originalmente por una investigación más profunda en la semántica del cálculo lambda y de la lógica Intuicionista [Gir87, Di 92].

Difiere de las lógicas Clásica e Intuicionista en varias maneras fundamentales: debido a la *naturaleza estática* de las proposiciones en esos formalismos, la validez de las afirmaciones es independiente del número en el cual éstas se presenten; así, uno puede duplicar proposiciones: $A \Rightarrow (A \wedge A)$, lo cual de manera implícita reconocemos que "la veracidad de un A es tan buena como la de dos". De igual forma, también podemos descartar proposiciones: $(P \wedge Q) \Rightarrow P$; aquí nos hemos desprendido de la proposición Q. Ambas sentencias son válidas en la lógica clásica para cualesquiera A, P y Q, por que se asume **representan hechos estáticos e invariantes acerca del mundo** [Gir87, Lin92].

1.1 Fórmulas como Recursos

Como motivación intuitiva, consideremos la lectura de las deducciones lógicas de tal manera que las afirmaciones sean entendidas como *recursos* y consecuentemente las conclusiones sean consideradas como los *requerimientos* a alcanzar mediante la *inversión* (utilización o gasto) de los recursos dados [Sce94]; en otras palabras, a cada *acción* tomada corresponderá alguna *reacción*. Asi, la primera afirmación anterior significaría que "mediante la inversión de un bien A" (acción) "puedo obtener otros dos" (reacción), mientras que la segunda significaría que "teniendo una pareja de recursos P y Q" (acción) "me quedaré con P" (reacción). Ante ésto, nos preguntamos: ¿de dónde vino la segunda A? y ¿hacia dónde se fue Q? respectivamente.

Para analizar a detalle tales aspectos, analizaremos el siguiente ejemplo: consideremos a las proposiciones D, M y C como los recursos "una moneda de diez pesos", "una cajetilla de cigarros Marlboro" y "una cajetilla de cigarros Camel". A su vez, consideremos la axiomatización de una máquina vendedora:

$$D \Rightarrow M$$

$$D \Rightarrow C$$

entonces, en la lógica Clásica (o intuicionista), podemos deducir:

$$D \Rightarrow (M \wedge C)$$

lo cual podemos leer como "con diez pesos, puedo comprar una cajetilla de Marlboros y una de Camels". Aun cuando esta deducción es válida en la lógica Clásica, no tiene sentido en la interpretación que pretendemos al considerar a las afirmaciones como recursos: no podemos adquirir 2 cajetillas de cigarros con sólo una moneda de diez pesos.

Ante lo anterior, las reglas para la Lógica Lineal implicarán que las proposiciones (lineales) se establecen respecto a *propiedades dinámicas o recursos finitos*, los cuales deben ser cuidadosamente administrados en el transcurso del tiempo, evitando las paradojas que se presentan en las lógicas clásica e intuicionista [Gir87].

Una manera conveniente para presentar la sintaxis de la Lógica Lineal es mediante una **modificación (refinamiento)** a la axiomatización de lógica Clásica (e Intuicionista) expresada mediante el cálculo de secuentes en el estilo de Gentzen tradicional [Sce93].

Para ello, se presentará brevemente a continuación al sistema Gentzen, donde se insistirá sobre las simetrías en sus reglas y a su vez se mostrará la importancia de las *reglas estructurales*. Una vez esto, se analizarán los aspectos que hacen inadecuada a las Lógicas Clásica e Intuicionista con respecto al cuidado de los recursos y consecuentemente se describirá el refinamiento correspondiente para la presentación de la Lógica Lineal en 3 pasos.

1.2 Cálculo de Secuentes Clásico

En 1934 Gentzen introdujo el *cálculo de secuentes*, el cual es una herramienta sintáctica básica para estudiar las leyes de la lógica. No es muy conveniente para construír pruebas pero resulta muy útil para estudiar sus propiedades. Gentzen introduce *secuentes*, expresiones de la forma $\Gamma \vdash \Delta$, donde Γ recibe el nombre de *antecedente* y Δ *consecuente*; a su vez, podemos referirnos a cada una como *Contexto*, en donde $\Gamma = (A_1, \ldots, A_n)$ y $\Delta = (B_1, \ldots, B_m)$ son secuencias finitas de fórmulas. El significado de un secuente es:

$$A_1 \ y \ \ldots \ y \ A_n \ \ conllevan \ a \ \ B_1 \ o \ \ldots \ o \ B_m$$

en donde el significado de *'y'*, *'conllevan a'* y *'o'* debe ser especificado por la lógica particular[1] [Gir95].

Una *regla de prueba* en el cálculo de secuentes consiste en un conjunto de secuentes *hipótesis (premisas)*, desplegados sobre la línea horizontal y un único secuente *conclusión* desplegado bajo la línea :

$$\frac{Hipótesis_1 \ Hipótesis_2 \ldots Hipótesis_n}{Conclusión}$$

Notablemente este es un sistema conveniente para razonar acerca de las pruebas y de la deducción axiomática:

- la mayoría de las lógicas pueden ser formalizadas en este sistema [LS93],
- presenta ventajas estructurales desde el punto de vista de la *mecanización*,
- ofrece el resultado importante de **eliminación de Cut (*Hauptsatz*)** (de importancia fundamental como se discutirá más adelante).

[1] Es decir, en el formalismo clásico (e intuicionista) corresponden a los operadores de conjunción ($\wedge$), consecuencia lógica ($\vdash$, a menudo se utiliza también el símbolo $\Rightarrow$) y disyunción ($\vee$), respectivamente.

Las convenciones a utilizar son las siguientes:

- a, b, p, q Fórmulas atómicas.
- A, B, C Fórmulas arbitrarias.
- $\Gamma, \Delta, \Theta, \ldots$ Multiconjuntos finitos arbitrarios (secuencias) de fórmulas.
- En general, siempre existirá alguna(s) fórmula(s) en cada Contexto que sea(n) de particular atención, a esa(s) fórmula(s) normalmente nombraremos como *principal(es)*.

Las reglas de inferencia (que constituyen el cálculo) están divididas en 3 grupos [Di 92, Gir95]:

1.2.1 Reglas Identidad

El *axioma de identidad* y la *regla cut*:

$$\frac{}{A \vdash A}\mathsf{Id} \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\mathsf{Cut}$$

1.2.2 Reglas Estructurales

Estas reglas son concernientes a la manipulación de contextos (es decir, el entorno de fórmulas en el cual se está discutiendo). Usándolas, podemos acceder a las hipótesis mediante *Intercambio:*

$$\frac{\Gamma, A, B, \Gamma' \vdash \Delta}{\Gamma, B, A, \Gamma' \vdash \Delta}\mathsf{LX} \qquad \frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, B, A}\mathsf{RX}$$

descartar hipótesis mediante *Debilitamiento:*

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\mathsf{LW} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\mathsf{RW}$$

o compartir hipótesis mediante *Contracción:*

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\mathsf{LC} \qquad \frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\mathsf{RC}$$

1.2.3 Reglas Lógicas

Las reglas de este grupo son concernientes respecto a los conectivos lógicos en la parte derecha e izquierda del símbolo $\vdash$ (consecuencia lógica):

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}L\wedge \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma' \vdash B, \Delta'}{\Gamma, \Gamma' \vdash A \wedge B, \Delta, \Delta'}R\wedge$$

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \vee B \vdash \Delta, \Delta'}L\vee \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}R\vee$$

Respectivamente, la reglas para la parte izquierda indican la *introducción* del conectivo, mientras que las reglas para la parte derecha indican la *eliminación* del conectivo.

La regla $L\wedge$ nos indica que el conectivo $\wedge$ es sólo una notación para la coma en la izquierda del símbolo $\vdash$ (contexto antecedente); similarmente, la regla $R\vee$ indica que el conectivo $\vee$ es una abreviatura para la coma en la parte derecha (contexto consecuente).

Nótese que podemos definir las reglas de introducción en una manera más económica:

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} L_1 \wedge \qquad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} L_2 \wedge \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta} R\wedge'$$

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta} L\vee' \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \vee B, \Delta} R_1 \vee \qquad \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \vee B, \Delta} R_2 \vee$$

Estos dos estilos distintos de manipular contextos en las reglas son llamados respectivamente *multiplicativo* y *aditivo*, [Di 92].

Brevemente, con propósitos de divulgación (y debido a que el resultado será de importancia posteriormente) se demostrará la equivalencia entre las reglas $R\wedge$ y $R\wedge'$ en el entorno clásico (e intuicionista).

Partiendo de la conclusión correspondiente a la regla $R\wedge'$:

$$\Gamma \vdash A \wedge B, \Delta$$

Para cualquier fórmula $A_i' \in \Gamma$, aplicamos la regla LC para adicionarla en el antecedente:

$$\cfrac{\cfrac{\vdots}{\Gamma, A_i' \vdash A \wedge B, \Delta}\text{LC}}{\Gamma \vdash A \wedge B, \Delta}\text{LC}$$

hasta que obtengamos un multiconjunto Γ' (no necesariamente igual a Γ):

$$\cfrac{\cfrac{\cfrac{\Gamma, \Gamma' \vdash A \wedge B, \Delta}{\vdots}}{\Gamma, A_i' \vdash A \wedge B, \Delta}\text{LC}}{\Gamma \vdash A \wedge B, \Delta}\text{LC}$$

y de igual forma, aplicamos RC para tener un consecuente adicional Δ':

$$\cfrac{\cfrac{\cfrac{\Gamma, \Gamma' \vdash A \wedge B, \Delta, \Delta'}{\vdots}}{\Gamma, \Gamma' \vdash A \wedge B, B_i', \Delta}\text{RC}}{\Gamma, \Gamma' \vdash A \wedge B, \Delta}\text{RC}$$

lo cual tiene la forma de la conclusión necesaria para utilizar la regla $R\wedge$:

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma' \vdash B, \Delta'}{\Gamma, \Gamma' \vdash A \wedge B, \Delta, \Delta'} R\wedge$$

quien a su vez constituye una instancia a la regla correspondiente $R\wedge'$, concluyendo la demostración parcialmente (es decir, se debe probar el recíproco). De manera similar, una prueba correspondiente para mostrar la equivalencia entre las reglas $L\wedge$ y $L1\wedge$, $L2\wedge$; asi mismo, la regla $L\vee$ es equivalente a $L\vee'$ y de forma correspondiente, la regla $R\vee$ es equivalente a las reglas $R1\vee$ y $R2\vee$.

1.3 Debilitamiento y Contracción: malos contadores.

Hay varias razones para querer liberarnos de las reglas de contracción y debilitamiento como reglas estructurales teniendo en mente nuestra motivación inicial.

Desde la perspectiva *económica*, podemos interpretarlas de la siguiente manera:

- Contracción : "si puedes comprar uno de los bienes en Δ utilizando varios bienes A, entonces lo puedes adquirir con un sólo A."
- Debilitamiento : "si puedes comprar uno de los bienes en Δ con Γ, entonces no daña gastar algunos A's sin obtener algo más a cambio."

En vista de ello, el primer paso para presentar a la Lógica Lineal como refinamiento de la lógica clásica es **eliminar las reglas estructurales de contracción y debilitamiento**, las cuales manipulan el uso de afirmaciones y conclusiones en las deducciones lógicas. Estas reglas estando juntas hacen posible descartar o duplicar las afirmaciones y serán por tanto responsables de la **pérdida de control sobre los recursos**. Esta intuición es el punto de arranque para la Lógica Lineal. La eliminación de tales reglas produce un *sistema lineal* en el cual cada suposición debe ser utilizada *solamente una vez* [Gir87, Sce94, Di 92, Lin92].

1.4 Dos Tipos de Conjunción y Disyunción

Una vez que decidimos ser cuidadosos respecto al uso de los recursos, sospechamos de los conectivos tradicionales $\wedge$ y $\vee$, descubriendo que cada uno ahora se divide en 2. De hecho, como se mostró anteriormente (en 1.2.3, pág. 10), sin las reglas de Contracción y Debilitamiento no es posible mostrar la equivalencia de las 2 formulaciones de las reglas de introducción izquierda para $\wedge(R\wedge, R\wedge')$ y $\vee(L\vee, L\vee')$; lo cual significa que las reglas distintas están describiendo *conectivos diferentes* [Gir87, Di 92].

El paso anterior conduce naturalmente a **2 formas de conjunción y disyunción**: *multiplicativa y aditiva*; a las cuales interpretamos de la siguiente manera [Gir87, Lin92]:

- $(A\otimes B)$ *Conjunción multiplicativa ("tensor")* expresa el poder disponer de ambos (recursos) A, B o lo que es lo propio, realizar dos acciones al mismo tiempo. Corresponde a la presentación tradicional no económica de $\wedge$ [Di 92]. Sus reglas correspondientes son:

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A\otimes B \vdash \Delta}\mathrm{L}\otimes \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma' \vdash B, \Delta'}{\Gamma, \Gamma' \vdash A\otimes B, \Delta, \Delta'}\mathrm{R}\otimes$$

  $L\otimes$ establece que el separador ',' es una abreviatura para el conectivo $\otimes$ en el antecedente, lo cual se entiende que todas las acciones Γ, A, B ocurren al mismo tiempo para provocar la reacción Δ. Por su parte, $R\otimes$ indica que las acciones Γ, Γ' que provocan las reacciones $A\otimes B, \Delta, \Delta'$ tienen origen debido a que se realizaron (como premisas) la acción Γ que condujo a la reacción A, Δ por un lado, y correspondientemente, la acción Γ' que provocó a la reacción B, Δ' [2]. Nótese la partición de los contextos para $R\otimes$: el antecedente se particiona en Γ, Γ' mientras que el consecuente en Δ, Δ', esto es un aspecto muy importante en la caracterización del conectivo.

[2] Al proceso de acción $\vdash$ reacción se le nombrará **interacción**.

- ($A \parr B$) *Disyunción multiplicativa ("par")*, el cual es el *dual* de $\otimes$, expresa una *relación de dependencia* entre dos tipos de acciones (y respectivamente reacciones), por la utilización de los recursos A y B. Es la forma simétrica de la *implicación lineal*, por un lado; y es el contenido constructivo (en la presentación tradicional) del $\vee$ clásico [Gir95]. Podemos parafrasearla como "Si no tengo a A, entonces me quedo con B", y consecuentemente "Si no tengo a B, entonces me quedo con A":

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \parr B \vdash \Delta, \Delta'} L\parr \quad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \parr B, \Delta} R\parr$$

  Esta disyunción puede entenderse con mayor facilidad si se considera a la *implicación lineal*, que se discute posteriormente junto con la negación lineal.

  Nótese la dualidad de $L\parr$ respecto a $R\otimes$: la diferencia reside en que la fórmula principal en $L\parr$ se encuentra en el antecedente (mientras que para $R\otimes$ se encuentra en el consecuente). La partición de los contextos se mantiene. Por otra parte, $R\parr$ indica que la coma en el consecuente es una abreviatura del conectivo. En términos de interacciones, la regla $L\parr$ indica que la interacción con acciones $\Gamma, \Gamma', A \parr B$ y reacciones Δ, Δ' es originada por las interacciones Γ, A (acción), Δ (reacción) y Γ', B (acción), Δ' (reacción).

  Notablemente, los conectivos multiplicativos **no comparten el contexto**.

- ($A\&B$) *Conjunción aditiva ("with")*: expresa el poder escoger entre A y B, pero no ambos al mismo tiempo, correspondiendo a la presentación económica del $\wedge$ clásico.

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A\&B \vdash \Delta} L_1\& \quad \frac{\Gamma, B \vdash \Delta}{\Gamma, A\&B \vdash \Delta} L_2\& \quad \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A\&B, \Delta} \mathrm{R}\&$$

  Su lectura es la siguiente: como parte de una acción ($L_1\&, L_2\&$) el resultado (reacción) es el mismo, independientemente de la fórmula (A ó B) que sea elegida; como parte de una reacción ($R\&$), tal interacción tiene origen en las 2 interacciones (premisas). En esta regla se aprecia que no se particiona el contexto, sino que se comparte.

- ($A \oplus B$) *Disyunción aditiva ("plus")* el dual de &, expresa la opción entre A y B, pero depende de su disponibilidad en ese instante. Corresponde a la presentación económica del $\vee$ clásico.

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \oplus B \vdash \Delta} L\oplus \quad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A\oplus B, \Delta} R_1\oplus \quad \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A\oplus B, \Delta} R_2\oplus$$

  De forma correspondiente, $L\oplus$ indica que dentro de una acción, las interacciones respectivas que la originan tienen la misma reacción, ya sea si estuvo A disponible ó B. La compartición de contextos se mantiene. Como parte de una reacción, ($R_1\oplus, R_2\oplus$) se interpreta que una interacción $\Gamma \vdash A\oplus B, \Delta$ puede ser originada por la interacción $\Gamma \vdash A, \Delta$ o bien por la interacción $\Gamma \vdash B, \Delta$.

  La distinción entre & y $\oplus$en términos computacionales, corresponde a la distinción entre no determinismo interno y externo.

Los nombres (multiplicativo, aditivo) vienen de una teoría semántica de la Lógica Lineal: espacios coherentes (ver 3.3, pág. 30), en donde los conectivos multiplicativos son modelados mediante el producto cartesiano de los dos operandos, mientras que los aditivos son modelados por suma directa.

Se lee al secuente lineal $\Gamma \vdash \Delta$ como la afirmación de que la conjunción multiplicativa de las fórmulas en Γ juntas conllevan a (se infiere) la disyunción multiplicativa de las fórmulas en Δ. Las reglas Identidad y Cut, así como la regla de Intercambio se mantienen.

1.5 Negación Lineal

Sin duda, este es el conectivo más importante $(.)^\perp$ [Gir89, Gir95], *nil* es la única operación negativa de la lógica, lo cual podemos observar en la definición de la **implicación lineal**:

$A \multimap B =_{def} A^\perp \parr B$, la cual puede interpretarse como "¿ puede B ser derivada a partir de A *exactamente una vez*?". Sus reglas correspondientes son:

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', (A \multimap B) \vdash \Delta, \Delta'} L\multimap \quad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash (A \multimap B), \Delta} R\multimap$$

La negación lineal se comporta como la transposición en el álgebra lineal : $A \multimap B$ es 'igual'[3] a $B^\perp \multimap A^\perp$; es decir, en este caso corresponde a un cambio en el punto de vista:

$$\textit{acción de tipo } A = \textit{reacción de tipo } A^\perp$$

Aspectos propios que expresa esta dualidad son: *entrada/salida, productor/consumidor, pregunta y respuesta, afirmación y meta*, etc.

Su propiedad principal es el ser involutiva: $(A^\perp)^\perp = A$, (lo cual asegura las leyes de De Morgan para todos los conectivos y cuantificadores, ver 2.1, pág. 18) y consecuentemente, constructiva[4].

Sus reglas de inferencia correspondientes son:

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, A^\perp \vdash \Delta} L^\perp \quad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash A^\perp, \Delta} R^\perp$$

Para cada uno de los conectivos, existe su *unidad* (neutro), (correspondiendo a que las constantes $\{V, F\}$ clásicas son divididas para multiplicativos y aditivos):

- 1 para $\otimes$: $A \otimes 1 \multimap\!\circ A$, $1^\perp \multimap\!\circ \perp$:

$$\frac{\Gamma \vdash \Delta}{\Gamma, 1 \vdash \Delta} \mathsf{L1} \quad \frac{}{\vdash 1} \mathsf{R1}$$

- $\perp$ para $\parr$: $A \parr \perp \multimap\!\circ A$, $\perp^\perp \multimap\!\circ 1$.

$$\frac{}{\perp \vdash} L\perp \quad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \perp, \Delta} R\perp$$

- $\top$ para &: $A \& \top \multimap\!\circ A$, $\top^\perp \multimap\!\circ 0$:

$$\frac{}{\Gamma \vdash \top, \Delta} R\top$$

[3] No existe igualdad en la Lógica Lineal, lo más acertado sería '$\multimap\!\circ$'.

[4] Esto es posible debido a que la negación lineal no impone un isomorfismo sobre la estructura semántica de la Lógica Lineal (mientras que la doble negación es ciertamente un isomorfismo [Gir87]). En los modelos usuales de la lógica clásica (retículos booleanos) la negación corresponde a voltear el retículo hacia abajo imponiendo un isomorfismo.

- 0 para $\oplus$: $A \oplus 0 \multimap\!\!\circ A$, $0^{\perp} \multimap\!\!\circ \top$:

$$\frac{}{\Gamma, 0 \vdash \Delta} L0$$

En una primera impresión, resulta un poco difícil imaginarse la presencia de dos tipos de verdad y falsedad. Por un lado, deben considerarse como consecuencia a la motivación respecto al cuidado de los recursos y consistentemente, dentro de las interacciones, en donde tal cuidado (administración) es realizado. Mientras que los conectivos aditivos expresan interacción mediante *elecciones* no determinísticas y comparten el contexto, los conectivos multiplicativos expresan dependencia (*necesidad*) entre los recursos relacionados y no compartir el contexto. Más importante aun, en tales interacciones el *tiempo* es importante, lo cual es ilustrado por la presencia de ambos recursos al mismo tiempo por parte de $\otimes$, la presencia de al menos uno en ese instante por parte de $⅋$ y las opciones propia de & y dependiendo de la disponibilidad temporal de $\oplus$ en el mismo contexto.

Como consecuencia, 1 puede interpretarse como una reacción que no es provocada por ninguna acción ni tiene origen en alguna interacción ($R1$): por lo que es una *interacción primitiva.* Como parte de una acción, es *consumida*, no afecta en nada a la reacción correspondiente. $\perp$ como acción también es una interacción primitiva, no provocando reacción alguna ($L\perp$). Como parte de una reacción es inofensiva ($R\perp$). Las constantes aditivas solamente pueden presentarse ya sea como reacción ($R\top$) o como acción (0) pero no ambas: denotan interacciones primitivas. $\top$ significa una elección *siempre* disponible, mientras que 0 es una opción que *nunca* estará disponible.

1.6 Exponenciales

Finalmente, el tercer paso consiste en **establecer una forma de modalidad**: *un operador de almacenamiento o reúso* **(exponencial)** que marca explícitamente a las proposiciones que pueden ser arbitrariamente reusadas:

- $!A$ ("bang" o "por supuesto" A -*productor*-) expresa proporción ilimitada del recurso A. En Lógica Lineal las fórmulas lineales (no exponenciales) son similares a *tokens* en una máquina de flujo de datos o señales en un cable eléctrico: son creadas e inmediatamente consumidas. Una fórmula exponencial corresponde computacionalmente a un dato almacenado en memoria:

  $$!p \circ\!\!-\!\!\circ 1 \,\&\, p \,\&\, (!p \otimes !p)$$

  lo cual indica que se pueden hacer tres operaciones con un dato almacenado: descartarlo (denotado por la unidad 1), utilizarlo -leerlo- (p), o duplicarlo ($!p \otimes !p$), y la opción es nuestra, [Gir89, Gir95]. Sus reglas correspondientes son: *Debilitamiento (W), Contracción (C), Derelicción (D) y Almacenamiento (S) (storage)*

  $$\frac{\Gamma \vdash \Delta}{\Gamma, !A \vdash \Delta}\mathsf{W!} \quad \frac{\Gamma, !A, !A \vdash \Delta}{\Gamma, !A \vdash \Delta}\mathsf{C!} \quad \frac{\Gamma, A \vdash \Delta}{\Gamma, !A \vdash \Delta}\mathsf{D!} \quad \frac{!\Gamma \vdash A, ?\Delta}{!\Gamma, !A \vdash ?\Delta}\mathsf{S!}$$

- $?A$ ("por qué no" A -*consumidor*-) es el conectivo dual de $!$: $?A =_{def} (!A)^{\perp}$ y expresa consumo ilimitado del recurso A: (reciben igual nombre que las reglas para *bang*)

  $$\frac{\Gamma \vdash \Delta}{\Gamma \vdash ?A, \Delta}\mathsf{W?} \quad \frac{\Gamma \vdash ?A, ?A, \Delta}{\Gamma \vdash ?A, \Delta}\mathsf{C?} \quad \frac{\Gamma \vdash A, \Delta}{\Gamma, ?A \vdash \Delta}\mathsf{D?} \quad \frac{!\Gamma, A \vdash ?\Delta}{!\Gamma, ?A \vdash ?\Delta}\mathsf{S?}$$

Debido a que el entorno básico permanece lineal, el consumo o reúso ilimitado es permitido "localmente" sólo en fórmulas específicamente marcadas con ? ó ! respectivamente. En efecto, las reglas estructurales de contracción y debilitamiento son reemplazadas por reglas lógicas explícitas acerca de las modalidades; resultando en un sistema lógico considerablemente bien estructurado [Gir89, Gir95, Gir87].

1.6.1 Estados y Transiciones

Tradicionalmente, debido a la atención (excesiva) de los estudiosos de la lógica sobre la matemática, la noción de *estado* ha sido pasada por alto: generalmente son introducidos parámetros temporales *ad-hoc* extra para que una formalización al respecto explique cómo pasar de un estado S (modelado como (S,t)) a un nuevo estado $(S', t+1)$; lo cual resulta muy desagradable [Gir89, Gir95].

El objetivo es deshacerse de tales parámetros, representando a los estados mediante fórmulas y a las transiciones por medio de implicaciones de estados, de tal forma que S' es accedible a partir de S exactamente cuando $S \multimap S'$ es demostrable a partir de una serie de transiciones consideradas como suposiciones. En la lógica clásica, el fenómeno de *actualización* no puede representarse debido principalmente a la idempotencia de los conectivos: las proporciones no importan. Por ejemplo, sea la ecuación química:

$$2H_2 + O_2 \rightarrow 2H_2O$$

Podemos parafrasearla en lenguaje natural como:

$$H_2 \text{ y } H_2 \text{ y } O_2 \text{ implican } H_2O \text{ y } H_2O$$

El sentido común sabe cómo manipular lo anterior como inferencia lógica; pero este sentido común sabe que 'y' no es idempotente (por que las proporciones son cruciales) y una vez que el estado inicial ha sido utilizado para producir el correspondiente estado final, ya no puede ser reutilizado. Las características que son necesarias aquí son las que ofrecen los conectivos $\otimes$ para representar 'y' y a $\multimap$ para representar 'implican', de esa manera, una expresión correcta será:

$$H_2 \otimes H_2 \otimes O_2 \multimap H_2O \otimes H_2O$$

por consiguiente, al establecer ecuaciones de esta forma como axiomas, la noción de *consecuencia lineal* corresponderá a la noción de estado accedible a partir de un estado inicial [Gir89, Gir95].

Originalmente en el entorno clásico, estábamos acostumbrados a la siguiente formulación de teoría :

$$\textit{Teoría} = \textit{Lógica Clásica} + \textit{Axiomas}$$

sin embargo, en la Lógica Lineal tal concepto debe ser refinado por:

$$\textit{Teoría} = \textit{Lógica Lineal} + \textit{Axiomas} + \textbf{Estado Actual}$$

Los axiomas son *para siempre* (y pueden por supuesto ser reemplazados por fórmulas $!A$), pero el estado actual es disponible para *uso único*: una vez que ha sido utilizado para probar otro estado entonces la teoría es actualizada, y tal estado pasa a ser el siguiente estado actual. Ello ofrece ventajas importantes: cierta información puede ser eliminada *logicamente*[5], en otras palabras, el *proceso de revisión* puede ser ejecutado por medio de consecuencia lógica. Se establece entonces la mencionada distinción entre fórmulas

[5] A diferencia notable en lenguajes de programación lógica como Prolog, en donde *assert* y *retract* no tienen un fundamento lógico plenamente aceptado.

!A (exponenciales) denotando *hechos estables* (por ejemplo, las reglas de un juego) y las ordinarias, denotando el *estado actual* [Gir89, Gir95].

Tales características dinámicas han sido explotadas en el área de Programación Lógica (ver Cap. 3), en donde la idea básica es que el método de resolución para la Lógica Lineal (búsqueda de pruebas en el cálculo de secuentes) *actualiza el contexto* [Gir95]. Actualización, herencia y paralelismo son las características principales de la programación lógica lineal.

1.7 Semántica Informal

Con la finalidad de familiarizar al lector con el significado de los conectivos dentro de un razonamiento, se presenta el ejemplo siguiente inspirado de Girard y Lafont [Di 92].

Supóngase que en un restaurante ofrecen el siguiente menú:

Ejemplo: Menú (Girard y Lafont).
Orden a Precio fijo: $5p
Torta
Refresco
todos los Frijoles refritos con totopos que se pueda comer
Sopa ó Ensalada (su opción)
Pastel o Helado (dependiendo disponibilidad).

Tal descripción se expresa mediante los conectivos de Lógica Lineal como sigue:

$$(p \otimes p \otimes p \otimes p \otimes p) \multimap (T \otimes R \otimes !F \otimes (S \& E) \otimes (P \oplus H))$$

Podemos señalar aspectos y propiedades computacionales muy sutiles de la Lógica Lineal con respecto al ejemplo anterior:

- Concurrencia Inherente : el poder contar con varios recursos al mismo tiempo, expresado mediante los conectivos multiplicativos ($\otimes, \invamp$), conlleva la expresión de concurrencia como una *propiedad natural* de este formalismo, más que como un aspecto operacional adicional al mismo.

  A su vez y en relación con ello, se cuenta con **no determinismo explícito** en 2 formas (expresados por los conectivos aditivos): *protagónica* (con la capacidad de decidir hacia cual recurso escoger que produzca mejores beneficios en el transcurso del cálculo) y *antagónica* (es decir, en donde la elección está supeditada a **eventos** fuera de nuestro alcance).

- Múltiples niveles de Abstracción : podemos apreciar en una manera fundamentalmente **declarativa** la especificación del comportamiento para el ejemplo anterior: expresamos las relaciones entre los recursos disponibles y los objetivos a alcanzar empleando dichos recursos. De igual forma podemos expresar el comportamiento correspondiente a cada recurso contendiente hasta el nivel que deseemos. Sin embargo, sutilmente estamos especificando al mismo tiempo de manera **operativa**, es decir, el *cómo* se están invirtiendo los recursos.

- Sensibilidad al Tiempo : como hemos mencionado inicialmente, el carácter dinámico que deseamos expresar adecuadamente implica esencialmente sensibilidad en el comportamiento con el transcurso del tiempo. Tal propiedad queda expresada en la Lógica Lineal como consecuencia a su *dependencia con respecto al entorno*, es decir, la disponibilidad actual (sutilmente temporal) de los recursos; lo cual da lugar a que la veracidad de las deducciones no sea *única*, eterna e invariante en el sentido platónico, sino se le confiera un carácter *dinámico* y potencialmente evolutivo.

- "Restricciones" implícitas : como consecuencia del cuidado de los recursos, notablemente se establecen restricciones respecto a su uso, lo cual necesariamente nos conduce a un enfoque de optimización en el uso de los recursos, pero además en una **estrategia adecuada** ante la carencia, limitación ó condicionamiento de los mismos (en el tiempo y espacio).

Tales aspectos le confieren a la Lógica Lineal una expresividad muy atractiva, lo cual permite modelar un amplio espectro de comportamiento.

1.8 Fragmentos

En la literatura, es común encontrar los siguientes nombres para los distintos fragmentos de la Lógica Lineal:

- LL el fragmento proposicional completo de la Lógica Lineal.
- MLL el fragmento multiplicativo de la Lógica Lineal (sin exponenciales).
- MALL el fragmento multiplicativo y aditivo de la Lógica Lineal (sin exponenciales).
- MLLE el fragmento multiplicativo de la Lógica Lineal con exponenciales.

Capítulo 2

Cálculo de Secuentes para la Lógica Lineal y Elementos para Búsqueda de Pruebas

2.1 Cálculo de Secuentes Lineal (asimétrico por la derecha)

La presentación del sistema de secuentes que se hizo en el capítulo anterior es muy redundante: debido a las simetrías presentes en las reglas, existen maneras de expresar algunas de ellas en términos de otras (utilizando la negación), permitiendo así una forma más concisa [Di 92]:

- Se define la *negación* de fórmulas no atómicas (arbitrarias) mediante las leyes de De Morgan:

$$
\begin{array}{ll}
(A \otimes B)^{\perp} =_{def} A^{\perp} \parr B^{\perp} & (A \& B)^{\perp} =_{def} A^{\perp} \oplus B^{\perp} \\
(A \parr B)^{\perp} =_{def} A^{\perp} \otimes B^{\perp} & (A \oplus B)^{\perp} =_{def} A^{\perp} \& B^{\perp}
\end{array}
$$

  Nótese que la negación lineal es un **concepto definido** sobre fórmulas compuestas, no como un conectivo básico.

- Convertimos al secuente simétrico: $A_1, \ldots, A_n \vdash B_1, \ldots, B_n$ en forma asimétrica (derecha únicamente) utilizando la implicación lineal (implícitamente): $\vdash A_1^{\perp}, \ldots, A_n^{\perp}, B_1, \ldots, B_n$.

De esa manera, las reglas quedan expresadas como:

- *Axioma de Identidad, reglas de Cut e Intercambio:*

$$
\frac{}{\vdash A^{\perp}, A}\mathsf{Id} \qquad \frac{\vdash \Gamma, A \qquad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta}\mathsf{Cut}
$$

$$
\frac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, B, A, \Delta}\mathrm{X}
$$

- *Debilitamiento y Contracción:*

$$\frac{\vdash \Gamma}{\vdash \Gamma, ?A}?W \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}?C$$

- *Reglas de Inferencia Lógica:*

$$\frac{\vdash \Gamma}{\vdash \Gamma, \bot}\bot \qquad \frac{\vdash A, B, \Gamma}{\vdash A \parr B, \Gamma}\parr \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}?$$

$$\frac{}{\vdash 1}1 \qquad \frac{\vdash A, \Gamma \qquad \vdash B, \Delta}{\vdash A \otimes B, \Gamma, \Delta}\otimes \qquad \frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}!$$

$$\frac{}{\vdash \Gamma, \top}\top \qquad \frac{\vdash A, \Gamma \qquad \vdash B, \Gamma}{\vdash A \& B, \Gamma}\& \qquad \frac{\vdash \Gamma, A[c/x]}{\vdash \Gamma, \forall x.A}\forall$$

$$\frac{\vdash A, \Gamma}{\vdash A \oplus B, \Gamma}\oplus_1 \qquad \frac{\vdash B, \Gamma}{\vdash A \oplus B, \Gamma}\oplus_2 \qquad \frac{\vdash \Gamma, A[t/x]}{\vdash \Gamma, \exists x.A}\exists$$

$$\frac{\vdash \Gamma, \Delta, A^{\perp}, B}{\vdash \Gamma, \Delta, (A \multimap B)}\multimap$$

Describiremos a detalle su significado dentro del procedimiento de prueba, primero debemos enunciar los elementos básicos para tal procedimiento.

2.2 Elementos para la Búsqueda de Pruebas.

Una **prueba** es representada como un árbol con raíz en el secuente conclusión en la parte inferior y con las hojas (hipótesis) en la parte superior (*bottom-up*). Con esta orientación, la notación de una regla ocurriendo arriba o abajo de otra es claro [Sce94, Di 92].

Cada **paso** en el proceso de búsqueda de pruebas es una aplicación inversa de una regla de inferencia lógica a un secuente (hoja), empatando la conclusión de la regla con el secuente para generar las premisas correspondientes (*instancia de una regla*). Como se ha indicado, la fórmula en cada paso que constituye la oportunidad para empatar alguna regla es nombrada como **fórmula principal**. Así, podemos ver a cada secuente en un instante dado constituído por alguna(s) fórmula(s) principal(es) y un resto (contexto), éste último normalmente denotado en las reglas por Δ y Γ.

Las reglas pueden separar contextos, formando bifurcaciones (ramas en el árbol de prueba) las cuales a su vez serán exploradas. El procedimiento de búsqueda terminará exitosamente cuando las únicas reglas que se puedan aplicar en un secuente hoja dado carezcan de premisas: el *axioma de identidad*, 1 y $\top$.

2.3 Comportamiento de las Reglas.

Enunciaremos a continuación la forma en cómo cada una de las reglas realizan la manipulación de los recursos (representados como fórmulas), de tal forma que nos permita vislumbrar su aplicación conveniente en el momento de presentar el procedimiento para búsqueda de pruebas. Como fue mencionado anteriormente, la oportunidad de su aplicación en un determinado paso de la prueba está sujeta a algún criterio para escoger la fórmula principal:

2.3.1 Reglas Multiplicativas:

- $(A \otimes B)$

  Observamos que la aplicación de la regla para $\otimes$ como conclusión prohibe compartir los recursos utilizados: las secuencias Γ, Δ deben asociarse adecuadamente con los recursos A y B respectivamente; esto conduce, por un lado, a una fuente de **no determinismo**, precisamente en la elección de la asociación adecuada (partición del contexto); y por otro lado a la creación de bifurcaciones (ramas) en el proceso de prueba, ampliando el espacio de búsqueda. Debido a esto, convendrá retardar su aplicación en la prueba hasta las etapas finales (prácticamente para formar hojas). Cada uno de los secuentes así formados que constituyen las hipótesis en el paso actual se convertirán en las nuevas conclusiones a ser demostradas.

- $(A \parr B)$

  La aplicación de la regla dentro de una prueba es muy simple: partiendo de un secuente de donde podemos identificar una pareja de fórmulas relacionadas con $\parr$, se procede a "separarlas", de tal manera que incrementamos en uno el número de fórmulas que componen el secuente (recordemos la coma es una abreviatura para dicha regla en las fórmulas que están en la parte derecha del secuente). En otras palabras, simple re-escritura.

2.3.2 Reglas Aditivas:

- $(A \oplus B)$

  Esta regla ofrece una fuente de incertidumbre especial debido a que tenemos oportunidad de utilizar 2 "sub"reglas a una misma instancia: en ambas, el secuente que queda como hipótesis está formado por una de las fórmulas (que estaban relacionadas por $\oplus$) y por la sucuencia Γ. Nótese que no se bifurca la búsqueda, sino que se da oportunidad, en la misma rama, de encontrar una prueba satisfactoria utilizando ya sea al recurso A y olvidarse de B o visceversa.

- $(A \& B)$

  A diferencia de $\otimes$, & requiere el compartir todos los recursos: una vez que identificamos la aplicación de ésta regla en un secuente (hoja de la prueba), procedemos a bifurcar la búsqueda sobre 2 nuevos secuentes, formados cada uno por una de las subfórmulas que estaban relacionadas con & y la secuencia completa de fórmulas (contexto).

2.3.3 Reglas utilizando Exponenciales:

Nótese la presencia de las reglas Debilitamiento **(?W)** y Contracción **(?C)**, las cuales indican explícitamente el uso no restringido de recursos dentro de un ámbito local (por la definición misma de los operadores de almacenamiento -exponenciales-).

Infinidad no significa mucho, sino *siempre* y tales reglas son por tanto las únicas responsables de la indecidibilidad (en el fragmento proposicional) [LMSS92, Gir95].

La Lógica Lineal introduce una nueva interpretación de las nociones básicas de las acciones lógicas, no elimina la noción de situación que se tiene originalmente, esto es por lo cual las reglas de Debilitamiento y Contracción para los exponenciales se mantienen.

Las manipulaciones de datos se expresan explícitamente através de las reglas **Promoción** (!) (almacenamiento), **Abandono** (?) (lectura), **Contracción** (?C) (duplicación) y **Debilitamiento** (?W) (eliminación).

2.3.4 Reglas para los Cuantificadores:

- Universal : La regla $\forall$ conduce a sustituír la variable cuantificada por una constante arbitraria c.
- Existencial : a su vez, la regla $\exists$ sustituye la variable cuantificada por un término arbitrario t para poder ser unificado al final de la prueba mediante el axioma de Identidad.

La suposición en la instanciación de las reglas anteriores puede ser eliminada mediante el empleo de las *funciones de Herbrand*. Además, el ordenamiento relativo de las reglas para cuantificadores en una prueba es particularmente importante debido a que la regla de generalización universal debe obedecer la condición de *eigen variable* [LS93].

2.4 Procedimiento de Búsqueda Básico.

En general la Lógica Lineal puede ser modelada (operacionalmente) mediante *reglas de re-escritura* sobre multiconjuntos, donde cada regla de inferencia usa elementos del multiconjunto especificado en sus afirmaciones y produce elementos del multiconjunto especificado en sus conclusiones [Abr93].

Particularmente, el Procedimiento de Búsqueda Básico puede describirse de la siguiente manera: partiendo de un secuente (nodo) inicial σ_0 dado, se construye *una* prueba Π incrementalmente; inicialmente Π es asignado a un sólo nodo (abierto[1]) etiquetado como σ_0, invocando al procedimiento BUSCAR(σ_0) [And92]:

Procedure BUSCAR(σ **: nodo abierto de** Π)

1. **Seleccionar** una instancia de una regla de inferencia del sistema de secuentes, cuyo secuente inferior (conclusión) sea igual a σ y secuentes superiores (hipótesis) $\sigma_1, \sigma_2, \ldots, \sigma_n$ (con $0 \leq n$);
2. **Expander** la prueba Π en el nodo σ con n ramas a los nuevos nodos abiertos etiquetados respectivamente con $\sigma_1, \sigma_2, \ldots, \sigma_n$;
3. Para cada $k = 1, \ldots, n$, **invocar** BUSCAR(σ_k).

Al respecto, debemos reflexionar lo siguiente:

- El paso 1 **no está completamente determinado**, ya que el criterio para la selección de una regla de inferencia no está especificada. Se asume que todas las posibles opciones pueden ser exploradas; en este paso pudiera ocurrir, no obstante, que muchas de esas pruebas sean redundantes.

  La selección de una regla de inferencia (como consecuencia de la elección de la fórmula principal activa) por tanto, constituye una fuente de **no determinismo** (por el momento la bautizaremos como *'no se cual'* regla aplicar); un criterio propio al respecto se presenta más adelante.

- En el paso 3, los nodos abiertos recién creados $\sigma_{k\ k=1,\ldots,n}$ pueden ser **buscados simultáneamente** (*paralelismo global* aplicado a todas las ramas del árbol de prueba).

[1] en el sentido de Prolog de una lista -abierta- cuyo resto será instanciado posteriormente.

2.5 Ejemplos de Búsqueda de Pruebas

Ejercicio 1 *Sea el secuente a demostrar:*

$$\vdash (p^\perp \parr q^\perp) \parr (q \otimes p)$$

Hay solo 2 reglas a elegir: $\parr$ o $\otimes$; aplicando primero $\parr$:

$$\dfrac{\dfrac{\dfrac{}{\vdash p^\perp, p}\,\mathsf{Id} \qquad \dfrac{}{\vdash q^\perp, q}\,\mathsf{Id}}{\vdash p^\perp, q^\perp, q \otimes p}\,\otimes}{\vdash (p^\perp \parr q^\perp) \parr (q \otimes p)}\,\parr$$

Intencionalmente se dejó la aplicación de $\otimes$ al final de la prueba, debido a sus características de comportamiento ya mencionadas: el no determinismo y la bifurcación del árbol de prueba.

Invirtiendo ahora la aplicación de las reglas:

$$\dfrac{\dfrac{\dfrac{\dfrac{fail}{\vdash p, p^\perp, q^\perp}}{\vdash p, p^\perp \parr q^\perp}\,\parr \qquad \dfrac{fail}{\vdash q}}{\vdash (p^\perp \parr q^\perp), (q \otimes p)}\,\otimes}{\vdash (p^\perp \parr q^\perp) \parr (q \otimes p)}\,\parr$$

Notablemente, ello conduce a una prueba errónea.

Ejercicio 2 *Sea la fórmula a demostrar:*

$$(!a^\perp \otimes !b^\perp) \parr ?(a \oplus b) \& (!(a^\perp \& b^\perp) \parr (?a \parr ?b))$$

El conectivo superior es &, por lo que la prueba inmediatamente es descompuesta en dos ramas.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{rama\ izq\ 1.1}{\vdash !a^\perp, \underline{?(a \oplus b)}}\,? \qquad \dfrac{rama\ izq\ 1.2}{\vdash \underline{!b^\perp}, ?(a \oplus b)}\,!}{\vdash \underline{!a^\perp \otimes !b^\perp}, ?(a \oplus b), ?(a \oplus b)}\,\otimes}{\vdash !a^\perp \otimes !b^\perp, \underline{?(a \oplus b)}}\,?\mathrm{C}}{\vdash !a^\perp \otimes !b^\perp \parr ?(a \oplus b)}\,\parr \qquad \dfrac{rama\ derecha:}{\vdash !(a^\perp \& b^\perp) \parr (?a \parr ?b)}\,\parr}{\vdash (!a^\perp \otimes !b^\perp) \parr ?(a \oplus b) \& (!(a^\perp \& b^\perp) \parr (?a \parr ?b))}\,\&$$

En la rama izquierda una vez aplicada $\parr$ se duplica la fórmula prefijada con ? mediante *Contracción* con el fin de utilizar a cada una en la rama correspondiente como resultado de aplicar $\otimes$.

En cada subrama, es descompuesta la fórmula subrayada:

rama izquierda 1.1:

$$\dfrac{\dfrac{\dfrac{}{\vdash a^\perp, a}\,\mathsf{Id}}{\vdash a^\perp, \underline{a \oplus b}}\,\oplus}{\vdash \underline{!a^\perp}, a \oplus b}\,!$$

y rama izquierda 1.2:

$$\dfrac{\dfrac{\dfrac{}{\vdash b^{\perp}, b}\,\mathsf{Id}}{\vdash b^{\perp}, \underline{a \oplus b}}\,\oplus}{\vdash b^{\perp}, \underline{?(a \oplus b)}}\,?$$

Para las cuales se obtienen nodos terminales en los cuales se aplica el axioma de Identidad.

Para la rama derecha la prueba continúa sin problemas:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a^{\perp}, a}\,\mathsf{Id}}{\vdash a^{\perp}, \underline{?a}}\,? \qquad \dfrac{\dfrac{}{\vdash b^{\perp}, b}\,\mathsf{Id}}{\vdash b^{\perp}, ?b}\,?}{\vdash (a^{\perp} \& b^{\perp}), ?a, ?b}\,\&}{\vdash \underline{!(a^{\perp} \& b^{\perp})}, ?a, ?b}\,!$$

Nótese el uso de las reglas $?C$ y $?W$ para duplicar y reducir fórmulas siempre y cuando ello nos conduzca a una prueba exitosa.

Sin embargo, las reglas de Debilitamiento y Contracción pueden producir bucles inútiles:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash \Gamma, ?F}}{\vdash \Gamma, ?F, ?F}\,?W}{\vdash \Gamma, ?F}\,?C \quad \Longleftrightarrow \quad \dfrac{\vdots}{\vdash \Gamma, ?F}$$

Ejercicio 3 *Sea la fórmula a demostrar:*

$$(p^{\perp} \parr (\exists x.q^{\perp}(x)) \parr (\forall y.q(y) \otimes p)$$

Una prueba exitosa es la siguiente:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash (p^{\perp}, p}\,\mathsf{Id} \qquad \dfrac{\dfrac{\{v \leftarrow c\}}{\vdash q^{\perp}(v), q(c)}\,\mathsf{Id}}{\vdash (\exists x.q^{\perp}(x)), q(c)}\,\exists}{\vdash (p^{\perp}, (\exists x.q^{\perp}(x)), q(c) \otimes p}\,\otimes}{\vdash (p^{\perp}, (\exists x.q^{\perp}(x)), (\forall y.q(y) \otimes p)}\,\forall}{\vdash (p^{\perp} \parr (\exists x.q^{\perp}(x)) \parr (\forall y.q(y) \otimes p)}\,\parr$$

No obstante, una prueba similar con una ligera modificación en el orden de las reglas falla:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash (p^{\perp}, p}\,\mathsf{Id} \qquad \dfrac{\textit{falla unificación}}{\vdash q^{\perp}(v), q(h(v))}}{\vdash p^{\perp}, q^{\perp}(v), q(h(v)) \otimes p}\,\otimes}{\vdash p^{\perp}, q^{\perp}(v), (\forall y.q(y) \otimes p)}\,\forall}{\vdash (p^{\perp}, (\exists x.q^{\perp}(x)), (\forall y.q(y) \otimes p)}\,\exists}{\vdash (p^{\perp} \parr (\exists x.q^{\perp}(x)) \parr (\forall y.q(y) \otimes p)}\,\parr$$

Nótese que la variable de Herbrand v *domina* a la función de Herbrand $h(v)$ debido a que es introducida primero. La verificación de ocurrencia falla indicando una posible transgresión a la condición de eigen variable.

Capítulo 3

Teoría de Pruebas, Semántica y Resultados de Complejidad

3.1 Teoría de Pruebas

Entre algunos enfoques para hacer a la lógica clásica más constructiva cabe citar [Ale94]:

- Considerar un fragmento: (el ejemplo más importante lo constituyen las *cláusulas Horn*) pero ello reduce el poder expresivo de la lógica y la torna inaceptable para ciertas aplicaciones.
- Intuicionismo: las pruebas son consideradas como funciones efectivas partiendo de afirmaciones y conduciendo a una conclusión.
- Lógicas Subestructurales: eliminar algunas reglas, como Contracción o Debilitamiento.

Por su parte, la Lógica Lineal tiene una Teoría de Pruebas muy rica y bien desarrollada; la cual es aplicada directamente para obtener interpretaciones computacionales de la lógica misma.

No obstante, como se ha ilustrado, existen problemas con el proceso de Búsqueda de Pruebas:

- hay muchas maneras de demostrar una proposición,
- muchas de esas pruebas difieren en el orden de aplicación de las reglas: aspectos de permutabilidad y simetría, que conducen a la "burocracia de la taxonomía" de Girard: muchas formas de expresar lo mismo [Gir87].

Las siguientes propiedades sintácticas de la Lógica Lineal ofrecen ventajas en el proceso de Búsqueda de Pruebas, y servirán para establecer una vía segura hacia la estrategia general que es enunciada más adelante (en 3.1.1, pág. 28).

3.1.1 Propiedades Importantes de la Búsqueda de Pruebas

Eliminación de Cut

Es una propiedad fundamental del cálculo lógico que la regla Cut pueda ser omitida sin pérdida de completez: cualquier derivación involucrando la regla Cut puede ser convertida a una correspondiente sin Cut[1]. Tal regla resulta computacionalmente infactible debido a que sus premisas contienen una fórmula que no aparece en su conclusión, originando así que el procedimiento de búsqueda haga suposiciones arbitrarias para escoger alguna fórmula [Sce94, MNSP91, And92, Gir95].

Para ilustrar esta propiedad, se utilizará un ejemplo:

Sea el secuente a demostrar $\vdash b^{\perp}, a^{\perp}, c$, y los axiomas propios:

$$A_1 :\vdash b^{\perp}, a^{\perp}, b \quad A_2 :\vdash b^{\perp}, (c \oplus c)$$

El uso de la regla Cut dentro de una prueba queda restringido de tal forma que una instancia de Cut debe introducir como premisas únicamente a axiomas propios de la teoría (eliminando así las suposiciones):

$$\dfrac{\dfrac{}{\vdash b^{\perp}, a^{\perp}, b}A_1 \qquad \dfrac{\dfrac{}{\vdash b^{\perp}, (c \oplus c)}A_2 \qquad \dfrac{\dfrac{}{\vdash c^{\perp}, c}Id \quad \dfrac{}{\vdash c^{\perp}, c}Id}{\vdash (c^{\perp} \& c^{\perp}), c}\&}{\vdash b^{\perp}, c}Cut}{\vdash b^{\perp}, a^{\perp}, c}Cut$$

En el primer paso se identifican $\Gamma = \{b^{\perp}, a^{\perp}\}$, $\Delta = \{c\}$ y $A = b$, mientras que en el segundo $\Gamma = \{b^{\perp}\}$, $\Delta = \{c\}$ y $A = (c \oplus c)$.

A su vez, para la prueba correspondiente *sin Cut*, los axiomas propios son incluídos como parte del secuente mediante la siguiente traducción:

Se define la traducción $[T]$ de una teoría T con k axiomas, a la fórmula de Lógica Lineal pura (es decir, sin axiomas) [LMSS92]:

$$[\{t_1, t_2, \ldots, t_k\}] =_{def} ?[t_1], ?[t_2], \ldots, ?[t_k]$$

en donde cada $[t_i]$ se define para cada axioma t_i de la siguiente manera:

$$[\vdash F, p_a^{\perp}, p_b^{\perp}, \ldots, p_z^{\perp}] =_{def} (F^{\perp} \otimes p_a \otimes \ldots \otimes p_z)$$

en donde F es una fórmula y $p_a, p_b, \ldots, p_z$ son átomos.

Así cada axioma se convierte en una fórmula reutilizable, donde la paridad (dualidad) de las subfórmulas del axioma han sido invertidas en el secuente correspondiente.

Formalmente, para cualquier conjunto finito de axiomas T, el secuente $\vdash \Gamma$ es demostrable en la teoría T si y sólo si $\vdash [T], \Gamma$ es demostrable.

La traducción para el ejemplo es:

$$[A_1] = (b \otimes a \otimes b^{\perp}) \quad [A_2] = (b \otimes (c^{\perp} \& c^{\perp}))$$

y la prueba sin Cut correspondiente es:

[1] "Un cálculo de secuentes sin eliminación de Cut es como un automóvil sin máquina" [Gir95]

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{Id}
\UnaryInfC{$\vdash b^{\perp}, b$}
\AxiomC{}
\RightLabel{Id}
\UnaryInfC{$\vdash a^{\perp}, a$}
\AxiomC{}
\RightLabel{Id}
\UnaryInfC{$\vdash b^{\perp}, b$}
\AxiomC{}
\RightLabel{Id}
\UnaryInfC{$\vdash c^{\perp}, c$}
\AxiomC{}
\RightLabel{Id}
\UnaryInfC{$\vdash c^{\perp}, c$}
\RightLabel{$\&$}
\BinaryInfC{$\vdash c, \underline{(c^{\perp}\&c^{\perp})}$}
\RightLabel{$\otimes$}
\BinaryInfC{$\vdash c, b^{\perp}, \underline{(b\otimes(c^{\perp}\&c^{\perp}))}$}
\RightLabel{$?\ [A_2]$}
\UnaryInfC{$\vdash \underline{?(b\otimes(c^{\perp}\&c^{\perp})), a^{\perp}, c, (a\otimes b^{\perp})}$}
\RightLabel{$\otimes$}
\BinaryInfC{$\vdash ?(b\otimes(c^{\perp}\&c^{\perp})), a^{\perp}, c, \underline{(a\otimes b^{\perp})}$}
\RightLabel{$\otimes$}
\BinaryInfC{$\vdash ?(b\otimes(c^{\perp}\&c^{\perp})), b^{\perp}, a^{\perp}, c, \underline{(b\otimes a\otimes b^{\perp})}$}
\RightLabel{$?\ [A_1]$}
\UnaryInfC{$\vdash \underline{?(b\otimes a\otimes b^{\perp})}, ?(b\otimes(c^{\perp}\&c^{\perp})), b^{\perp}, a^{\perp}, c$}
\end{prooftree}
$$

Subfórmula

Una consecuencia importante de la propiedad de eliminación de Cut tanto en el fragmento proposicional como en el de primer orden, es la **Propiedad de Subfórmula**; esto es, si una fórmula A es probable (demostrable) entonces existe una prueba de A en la cual *únicamente las subfórmulas de A ocurren.* La Propiedad de Subfórmula constituye una base teórica de la búsqueda de pruebas [Sce94] debido justamente a que limita el espacio de búsqueda.

Permutabilidades

Un rol igualmente importante en el paradigma de búsqueda de pruebas es realizado por ciertas **Propiedades de Permutabilidad**, las cuales producen presentaciones optimizadas de las pruebas libres de Cut. Las propiedades de Permutabilidad aseguran que si una fórmula A es probable (demostrable) entonces existe una prueba de A de una forma permutada, es decir, permutando el *orden* en cómo se aplican las reglas de inferencia en una prueba, a fin de lograr un tipo de *forma normal.* Las propiedades de Permutabilidad son utilizadas para tener control sobre la forma de las pruebas de Lógica Lineal, permitiendo así una lectura computacional más directa [HM94].

Una regla R_1 se dice que es *impermutable* bajo R_2 (R_1/R_2) si hay una Γ tal que la fórmula principal de R_1 y R_2 ocurren como fórmulas distintas en Γ, pero $\vdash \Gamma$ puede ser demostrado únicamente con R_2 abajo de R_1. De otra forma, se dice que R_1 es permutable bajo R_2 (equivalentemente, R_2 es permutable sobre R_1) [LS93].

Como consecuencia, una manera para explotar las permutabilidades es restringir el retroceso en la búsqueda de pruebas a aquellas reglas que no pueden permutar bajo otras. Esto lo podemos visualizar mejor en la siguiente tabla:

$\frac{R_1}{R_2}$	$\otimes$	$⅋$	$\&$	$\oplus$	$?D$	$!$	$\perp$	$\forall$	$\exists$
$\otimes$							0		
$⅋$	1						0		
$\&$	2			3	4	0		5	
$?D$							0		
$!$						6	0		
$\perp$							0		
$\forall$							0		
$\exists$							0		

Algunas permutabilidades (tipo 1, 3 y 7) han sido ilustradas en los ejemplos del capítulo anterior. Nótese el siguiente ejemplo, en el que el orden de aplicación de las reglas es irrelevante:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A, A^{\perp}}Id \qquad \dfrac{}{\vdash B, B^{\perp}}Id}{\vdash (A\otimes B), A^{\perp}, B^{\perp}}\otimes}{\vdash (A\otimes B), A^{\perp} \parr B^{\perp}}\parr \qquad \dfrac{}{\vdash C, C^{\perp}}Id}{\vdash (A\otimes B)\otimes C, A^{\perp} \parr B^{\perp}, C^{\perp}}\otimes \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A, A^{\perp}}Id \qquad \dfrac{}{\vdash B, B^{\perp}}Id}{\vdash (A\otimes B), A^{\perp}, B^{\perp}}\otimes \qquad \dfrac{}{\vdash C, C^{\perp}}Id}{\vdash (A\otimes B)\otimes C, A^{\perp}, B^{\perp}, C^{\perp}}\otimes}{\vdash (A\otimes B)\otimes C, A^{\perp} \parr B^{\perp}, C^{\perp}}\parr$$

Búsqueda conducida por la Sintaxis

Los conectivos transmiten mejor el uso proyectado de las fórmulas: la prueba está codificada directamente en las fórmulas a ser demostradas, lo que J.M.Andreoli y R.Pareschi nombran "búsqueda de pruebas dirigida por la sintaxis".

Para ilustrar mejor esta idea, recordemos el ejemplo 2 (pág. 22). La fórmula a ser demostrada es descompuesta mediante un *árbol sintáctico*, en el cual cada conectivo aparece como un nodo interior y las hojas estan constituidas por fórmulas atómicas (ver la fig. 3.1). La disposición de las ramas obedece a alguna prioridad sobre los conectivos: en el ejemplo, $\parr$ es el conectivo más externo, que une a las subfórmulas con $\otimes$ y ? como operadores principales, respectivamente.

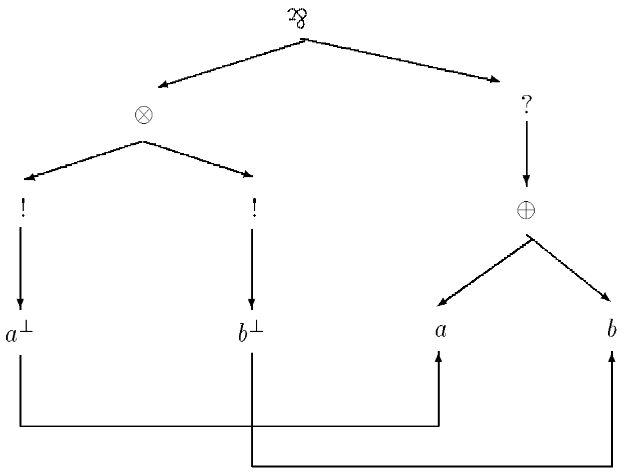

Figura 3.1: Árbol sintáctico correspondiente a la rama izquierda de la prueba del ejercicio 2.

El recorrido de este árbol guiará los pasos correspondientes a su prueba: por un lado, un exámen de cada nivel del árbol indicará las reglas que convienen a ser utilizadas en base a la naturaleza del conectivo principal, debido a que en cada nivel se encuentran las subfórmulas en el secuente actual; y por otro, (gracias a que las hojas constituyen fórmulas atómicas) conducirá la búsqueda por duales correspondientes para aplicar el axioma identidad.

En el ejemplo, al recorrer el árbol se parte de $\parr$, el cual conduce a descomponer las subfórmulas que está relacionando, indicadas como hijo derecho e izquierdo; tales subfórmulas tienen como conectivos más externos a $\otimes$ y ? respectivamente: este nivel involucra elegir cual fórmula descomponer, como se ha mencionado, conviene retardar la aplicación de $\otimes$, a su vez, la subfórmula derecha al estar modalizada podemos considerarla como "pasiva" (en el sentido que conviene usarla sólo cuando sea necesario, es decir, para cuando las subfórmulas que la constituyen contengan alguna fórmula dual que necesitamos para obtener una prueba exitosa aplicando el axioma de identidad). Por tanto, podemos descomponer $\otimes$: ello conduce a la prueba de las dos subfórmulas $!a^{\perp}$ y $!b^{\perp}$, para las cuales la regla correspondiente es inmediata, obteniendo los átomos $a^{\perp}$ y $b^{\perp}$. Para que cada rama sea exitosa (y por ende, toda la subfórmula padre), debemos obtener del contexto los duales de cada uno de estos átomos, es entonces cuando echamos mano de la fórmula modalizada ?: esta búsqueda es indicada por las flechas, el sentido indica de dónde se parte.

El uso de árboles sintácticos resulta muy útil para conducir la búsqueda y evitar retrocesos, debido principalmente a que se aprovecha la *propiedad de Subfórmula* e implícitamente se establece un orden (fundamentado en la naturaleza de los conectivos) para realizar tal búsqueda.

Notablemente, un aspecto en contra del desempeño es que el orden (*secuencialidad*) en la presentación de una prueba es inevitable: si deseamos aplicar un conjunto S de n reglas a diferentes partes de un secuente, no podemos aplicarlas en un solo paso, aun si las reglas no intervienen entre sí. Se debe escoger un orden lineal sobre S y aplicar las reglas en n pasos de acuerdo a este orden, lo cual conduce potencialmente a $n!$ pruebas intencionalmente distintas pero que pueden considerarse como una sola prueba [Di 92].

Podemos resumir, en términos generales, que el paradigma de búsqueda de pruebas (cuyo objetivo permite obtener un sistema factible), está fundamentado en la siguiente estrategia general [And92]:

1. restringir la atención hacia una cierta clase de pruebas las cuales *sean más fáciles de construír* que todas las demás pruebas en general y entonces,

2. *identificar un fragmento sensible* de la lógica para el cual este tipo restringido de pruebas *es completo.*

Una construcción que evita la burocracia de la taxonomía está constituída por las *Redes de Pruebas (Proof nets)* inventadas por J.Y.Girard [Gir87]; consisten en grafos con nodos marcados por fórmulas (no secuentes) y aristas dirigidas que unen las premisas de la ocurrencia de una regla con su conclusión. Una derivación puede tener más de una conclusión (unidas por $\invamp$), los axiomas son construídos uniendo las proposiciones duales (que no necesitan estar en el mismo nodo) mediante una arista no dirigida. Gracias a ello, el problema de secuencialidad no está presente en las Redes de Pruebas.

Tales consideraciones han inspirado a J.Y.Girard la formulación de la *Geometría de la Interacción* [Gir89], que expresa el deseo de obtener una buena formulación de la concurrencia (análoga a lo que el cálculo λ ha logrado para el cómputo secuencial) intentando capturar el comportamiento de programas finitos mediante estructuras dinámicas.

3.2 Resultados de Complejidad

Considerablemente, la expresividad de la búsqueda de pruebas en Lógica Lineal también es indicada por los trabajos sobre complejidad e indecidibilidad de (demostrabilidad en) los fragmentos naturales de la Lógica Lineal debidos principalmente a P.Lincoln y M.Kanovich [Sce94, Ale94, LMSS92] entre otros. Dichos trabajos se enfocan en las representaciones directas, bloqueo-paso (*lockstep*) de varios entornos computacionales genéricos mediante búsquedas de pruebas libres de cut en los fragmentos correspondientes de la Lógica Lineal. Tales representaciones revelan una relación estructural entre los fragmentos naturales de la Lógica Lineal por un lado y las clases de complejidad estándar por el otro. Desafortunadamente, los resultados de complejidad son bastante restrictivos.

- La Lógica Lineal Proposicional es *indecidible*, debido principalmente a la presencia de los exponenciales. Este resultado es obtenido codificando el comportamiento de máquinas contadoras simples [LMSS92].
- El fragmento Multiplicativo con exponenciales (MLLE) tiene complejidad EXPSPACE, lo cual resulta de la codificación de Redes de Petri para este fragmento.
- La parte multiplicativa-aditiva de primer orden (sin exponenciales) tiene complejidad NEXPTIME [LMSS92].
- El fragmento multiplicativo y aditivo (sin exponenciales y cuantificadores) es PSPACE-completo [LMSS92].
- Los fragmentos multiplicativos y de Horn son NP-completos [LMSS92].
- Aun partes severamente limitadas de los fragmentos anteriormente mencionados son NP-completos, esto significa uso de sólo constantes (no letras proposicionales) o libre de constantes, con dos o tres letras predicales.

Tras observar tales resultados, inevitablemente se establece la pregunta: ¿Cual puede ser el uso práctico de la Lógica Lineal?. Las siguientes consideraciones son refrescantes al respecto [Ale94]:

- La característica de ser "bastante compleja" tiene por sinónimo el ser "muy expresiva"; de tal suerte que la necesidad de una lógica expresiva (para especificación de software, CASE o para modelado de alto nivel) implica aceptar su alta complejidad.
- Mucho de la indecidibilidad de la Lógica Lineal tiene origen en la introducción de los exponenciales. No obstante el concepto de las fórmulas como recursos ya está presente en el fragmento sin exponenciales y puede ser suficiente para algunas aplicaciones: especificación de interconexión para procesos concurrentes basados en redes de pruebas, por ejemplo.
- Los resultados de complejidad anteriores no cuentan para diferencias pragmáticamente importantes, por ejemplo, la Lógica Lineal Intuicionista también es indecidible como la Lógica Lineal completa, pero ello no importa dado que éste fragmento se utiliza como un lenguaje de programación funcional y no se buscan sino que se reducen pruebas (ver 3.4.1). Un trabajo considerable por J.Y.Girard, A.Scedrov y P.Scott (citado en [Ale94]) proporciona una formulación de la clase de funciones computables en tiempo polinomial utilizando la *Lógica Lineal Acotada (Bounded)*, en donde los exponeciales son indexados mediante números naturales $!_n$ y $?_n$ limitando así su utilización.

Conectivos en el Fragmento			Complejidad de la Lógica Lineal
$\otimes$ ⅋	& $\oplus$	!?	Indecidible
$\otimes$ ⅋	& $\oplus$		PSPACE-Completo
$\otimes$ ⅋		!?	desconocida
$\otimes$ ⅋			NP-Completo

3.3 Semántica

En su artículo inicial [Gir87] J.Y.Girard introdujo dos entornos semánticos. El primero es la *semántica de espacio fase (phase space semantics)*, que asocia valores a las fórmulas en el espíritu de la teoría de modelo de la lógica clásica, modelando únicamente a la probabilidad y no a las pruebas [Gir95]. Esta semántica tiene un origen algebraico en donde los dominios de interpretación son estructuras algebraicas (posteriormente nombradas estructuras de Girard). Tales estructuras son monoides conmutativos M con un subconjunto distinguido llamado el conjunto de "anti fases" $\perp$. Para un conjunto $X \subseteq M$, su dual está definido como $X^{\perp} = y|\forall x \in X.xy \in \perp$. Entonces $(.)^{\perp\perp}$ es monotónico, inflacionario ($X \subseteq X^{\perp\perp}$) e idempotente, es decir, una operación de cerradura. Los conjuntos cerrados (con los cuales $X = X^{\perp\perp}$) son llamados *hechos* y tales son las entidades que interpretan a las fórmulas de la Lógica Lineal. Puede demostrarse que los hechos considerados bajo la inclusión de conjuntos forman un retículo completo y la operación monoide conmuta con las operaciones del retículo.

La relación del retículo $\geq$ es utilizada para interpretar probabilidad, los teoremas están caracterizados por $1 \geq \phi$. La interpretación de los conectivos es la siguiente:

1. $\otimes$: $X \otimes Y := \{mn; m \in X \wedge n \in Y\}^{\perp\perp}$
2. $⅋$: $X ⅋ Y := (X^{\perp} \otimes Y^{\perp})^{\perp}$
3. 1: $1 := \{1\}^{\perp\perp}$, donde 1 es el elemento neutro de M.
4. $\oplus$: $X \oplus Y := (X \cap Y)^{\perp\perp}$
5. &: $X \& Y := X \cap Y$
6. cero: $0^{\perp\perp}$
7. Top: $\top := M$
8. bang: $!X := (X \cap I)^{\perp\perp}$, donde I es el conjunto de idempotentes de M los cuales pertenecen a 1.
9. ¿por que no?: $?X := (X^{\perp} \cap I)^{\perp}$

El segundo entorno semántico es la *semántica de espacio coherente (coherence space semantics)*, (la cual corresponde al área denotacional) cuyo propósito es analizar el contenido implícito de las pruebas [Gir95].

Se define primero a las "telarañas" como grafos no dirigidos (conjuntos equipados con una relación simétrica y reflexiva '$\circ$' llamada "coherencia"). Los espacios de coherencia son *uniones (cliques)* de la telaraña, (subconjuntos donde cada pareja de elementos es coherente), considerados como parcialmente ordenados completos bajo la inclusión de conjuntos. Cada espacio de coherencia interpreta una fórmula. Dados dos espacios A y B, la combinación multiplicativa $A \times B$, $A ⅋ B$ o $A \multimap B$ es interpretada tomando el producto cartesiano de las dos telarañas correspondientes, esto es, $|A \otimes B| = |A ⅋ B| = |A \multimap B| = |A|\mathrm{x}|B|$; y estipulando ciertas condiciones (intuitivas) sobre coherencia (por ejemplo, para $\times$, $(a_1, b_1) \circ (a_2, b_2)$ está definido como $a_1 \circ a_2$ y $b_1 \circ b_2$). Las combinaciones aditivas son interpretadas tomando la unión disjunta de las telarañas correspondientes ($|A \& B| = |A \oplus B| = |A| + |B| = \{(0, a)|a \in |A|\} \cap \{(1, b)|b \in |B|\}$), de nuevo con ciertas condiciones de coherencia. La modalidad exponencial es interpretada mediante el conjunto de todos los subconjuntos coherentes finitos de el operando $|A!| = ⅋(|A|)$. La negación no juega un rol importante aquí como en la semántica espacio de fase.

D.Miller y J.Hodas [HM94] proponen una semántica para su lenguaje de programación lógica basado en Lógica Lineal, (la cual constituye una excepción a la carencia de interpretaciones al estilo Tarsky de forma "$M \models \phi \wedge \psi$ sii $M \models \phi y M \models \psi$" para la Lógica Lineal completa); concebida para el propósito pragmático de proporcionar un modelo canónico (determinado) para el lenguaje y no es completamente general, debido a que carece de negación (aunque los teoremas de *soundness* y *completeness* son demostrados para el fragmento correspondiente). Tal semántica consiste de un conjunto de interpretaciones *Kripke* K_r (todas definidas en el mismo entorno de mundos $< \mathcal{W}, \geq >$) donde $r \in R$ y $< R, +, 0 >$ es un monoide conmutativo de "recursos", correspondiendo a el tratamiento de multiconjuntos en la Lógica Lineal.

Una clase de semántica importante la constituyen las *semánticas categóricas*. Eliminan la concretez innecesaria presente en las otras semánticas y establecen una conexión a un rico conjunto de resultados existentes en la teoría de categorías. Las nociones categóricas proporcionan luz para el diseño de lógicas e interpretaciones, como también para lenguajes de programación y máquinas abstractas. Las *categorías Dialécticas* de V.DePaiva pertenecen también a esta clase [Ale94].

De reciente desarrollo y con aplicaciones importantes hacia los modelos de concurrencia basados en Lógica Lineal es la *semántica de juegos*, en la cual una prueba de Lógica Lineal es interpretada como una estrategia de éxito independiente a la historia en un juego con dos oponentes [Gir95].

3.4 Aplicaciones

La aplicabilidad de la Lógica Lineal dentro de los ámbitos computacionales obedece principalmente a su profunda *simetría* (expresada por el operador definido de *dualidad* $(.^{\perp})$ y las leyes de De Morgan); (ver 1.5). Tal simetría la hace adecuada para proporcionar **modelos abstractos** de cálculo libres de detalles de implantación los cuales son, por naturaleza, orientados y no simétricos [And92].

Las ramificaciones de la Lógica Lineal en la ciencias computacionales pueden ser divididas generalmente en dos amplias áreas (involucrando ambas aspectos de concurrencia) : [And92]

- Programación Funcional
- Programación Lógica

Brevemente se describen a continuación[2].

3.4.1 Programación Funcional

Probablemente ésta es el área de aplicación más desarrollada de la Lógica Lineal. En éste paradigma la simetría de la Lógica Lineal conduce a la formulación del procedimiento de eliminación de Cut, en donde un cómputo es visto como **manipulación de recursos**: la *entrada* corresponde al consumo de recursos y la *salida* a la producción de los mismos. Ambas operaciones son estrictamente duales e inherentemente concurrentes [And92].

La idea fundamental aquí es el *isomorfismo Curry-Howard*, o paradigma de *fórmulas como tipos*. Este isomorfismo afirma la existencia de una correspondencia entre las pruebas de un sistema lógico intuicionista y los cálculos en un lenguaje de programación funcional; lo cual tiene su base en el enfoque intuicionista: una

[2] Para una descripción detallada de las aplicaciones, consultar el reporte de V.Alexiev [Ale94].

prueba debe ser una función efectiva en la cual dadas ciertas afirmaciones produce conclusiones. Consistentemente, en este entorno un cálculo es identificado como reducción de términos correspondiente a la reducción de pruebas (es decir, el proceso de eliminación de Cut para normalización de pruebas) [Sce93, Sce94].

Ello conduce como consecuencia a un *refinamiento del cálculo* λ en el cual el programador tiene control más fino sobre el orden de evaluación (tardío -*lazy*- versus temprano) y sobre la asignación de memoria (estructuras de datos con hilos de referencia múltiples las cuales demandan recolección de basura versus estructuras de datos con un sólo hilo -lineal- la cual no necesita recolección).

3.4.2 Programación Lógica

A su vez, un enfoque computacional basado en administración de recursos se aplica en ésta área, especialmente a los sistemas de *programación lógica concurrente.* En estos sistemas un cálculo consiste en un conjunto de procesos ejecutándose concurrentemente (particulamente en paralelo), posiblemente intercambiando información. Cada proceso está completamente caracterizado (en cualquier instante dado) por su *estado* el cual es una **colección de recursos disponibles**. Las *transiciones de estado* (que constituyen los pasos elementales en la evolución de los procesos) consisten en la *manipulación de los recursos:* producción, consumo, duplicación, distribución, etc. [And92].

Para esta área mencionada, el modelo abstracto proporcionado por la Lógica Lineal se fundamenta en el principio computacional básico [Sce94, Lin92, LS93, And92]:

Cálculo = Búsqueda de Pruebas (Demostraciones)

Esto es debido a que las pruebas expresadas mediante el cálculo de secuentes ofrecen una herramienta satisfactoria para **formalizar** la *historia de ejecución (de un conjunto de procesos) durante un intervalo de tiempo dado.* Con esto en mente, podemos establecer la siguiente correspondencia:

Evolución de procesos $\rightarrow$ árbol de Prueba.
Estado Inicial $\rightarrow$ raíz del árbol *(conclusión).*
Estados Intermedios $\rightarrow$ nodos internos *(hipótesis).*
Estados Finales $\rightarrow$ hojas *(fórmulas inferibles).*

De tal forma, un Sistema de Secuentes (el cual describe las inferencias correctas en las pruebas) puede verse como una **Especificación Formal** *de las transiciones de estados permitidas para un conjunto de procesos* [And92].

La Lógica Lineal ofrece una estructura más rica (en comparación al fragmento clásico restringido a las cláusulas Horn) para la representación de estados de procesos:

- Estado representado como *Secuente,* el cual es un multiconjunto de fórmulas (no ordenado) más adecuado a ser accedido concurrentemente.

- Las fórmulas son *recursos de uso restringido* que no pueden ser duplicadas arbitrariamente. Tal aspecto es explícito en la Lógica Lineal [Lin92, Sce94, Sce93], y no deriva de una interpretación del fragmento de la lógica (cláusulas Horn y metas, como en Prolog).

Es importante, por todo lo anterior, idear un procedimiento para búsqueda de pruebas que sea tan eficiente en la manipulación de recursos como la resolución SLD para las cláusulas Horn. Tal procedimiento encuentra dos corrientes importantes: las *Pruebas Uniformes* por Dale Miller y las *Pruebas Enfocadas* por Jean Marc Andreoli.

Pruebas Uniformes

Esta perspectiva computacional está basada en el trabajo de Dale Miller y colaboradores [MNSP91]. El fundamento teórico consiste en considerar a los *programas como colecciones de fórmulas (teorías)* usadas para especificar el significado de constantes no lógicas y el cálculo es identificado precisamente con la búsquedas de pruebas libres de Cut en el cálculo de secuentes al estilo Gentzen. En este entorno, el secuente:

$$\Sigma : \Delta \to G$$

denota el estado de un intérprete de programación lógica ideal, en donde el *estado actual* de las constantes no lógicas (*signature*) es Σ, el *programa lógico actual* (contexto, teoría) acerca de tales constantes es el conjunto de fórmulas Δ y la fórmula a ser establecida (demostrada), llamada *consulta* o *meta* es G. (Se asume que todas las constantes no lógicas en G y en las fórmulas en Δ están contenidas en Σ).

El entorno constituído por una lógica y su sistema de pruebas correspondiente será considerado como un lenguaje de programación lógica si un tipo simple de búsqueda *conducido por metas (goal-directed)* es completo. Una *prueba uniforme* es, por consiguiente, una prueba dirigida por metas libre de cut en la que cada ocurrencia de un secuente cuya parte derecha es no atómica es la conclusión inmediata de una regla de introducción derecha; en otras palabras, el procedimiento de búsqueda reduce cada meta compuesta a sus partes y consultará en el programa (contexto, parte izquierda del secuente) únicamente cuando la meta sea atómica. Dentro de tal proceso, la estructura de la parte derecha del secuente (la meta) puede ser reflejada directamente en la prueba que se está construyendo. Tal entorno (la lógica y su sistema de pruebas) es llamado un *lenguaje de programación lógica abstracto* si y sólo si un secuente tiene una prueba uniforme.

Pruebas Enfocadas

Jean Marc Andreoli y Remo Pareschi desarrollaron un tratamiento declarativo de la comunicación entre objetos y el cómputo concurrente orientado a objetos [AP90]. Como parte de este trabajo implantaron un lenguaje de especificación concurrente **LO**, el cual fue uno de las primeras propuestas para programación lógica basada en Lógica Lineal. Esencialmente extiende las cláusulas de Horn permitiendo que las cabezas de las cláusulas estén formados no sólo por átomos, sino por multiconjuntos de átomos unidos por el conectivo ($\invamp$). De esa forma, la ejecución de LO puede verse principalmente como re-escritura sobre multiconjuntos.

Linlog, por su parte, extiende LO incorporando varios operadores adicionales. El fragmento es completo para toda la Lógica Lineal (es decir, cualquier fórmula bien formada en Lógica Lineal pura puede pasarse a su forma Linlog correspondiente). Su fundamento operacional está determinado por la noción de pruebas conducidas por metas llamadas *Pruebas Enfocadas*, (cercanamente relacionadas a las Pruebas Uniformes) cuyo propósito es eliminar la búsqueda de pruebas redundantes, enfocándose en un solo conjunto de pruebas a partir de las cuales todas las demás pueden ser derivadas.

3.5 Hacia la unidad de la Lógica

Originalmente, la situación concerniente a la lógica fue bastante simple: la lógica clásica podía ser utilizada (modificando el conjunto de axiomas propios) en diversas áreas: su objetivo era el *razonamiento puro* [Gir95]. Brouwer destruyó tal ideal de unidad: la lógica clásica no estaba adaptada para características constructivas, perdiendo por ello su universalidad [Ale94]. Al final del siglo nos encaramos ante un número considerable de lógicas, surgidas fundamentalmente gracias a una convergencia cada vez más profunda entre las áreas de la lógica y la computación: la fuerte conexión (y en algunos casos, como se ha descrito, hasta el punto de identidad completa) entre deducción y cómputo; por lo que nuevas dudas surgen: ¿Sigue siendo el objetivo de la lógica el razonamiento puro? ¿Podría existir una manera de reunificar a los sistemas lógicos en *un sólo* cálculo de secuentes? ¿Podría manejarse la distinción (legítima) entre lógica clásica/intuicionista no mediante un cambio de sistema sino que a través de un cambio de fórmulas? [Gir95].

Para responder a tales cuestiones, es propuesto un *Entorno Lógico*[MMO93] dentro del cual muchas lógicas puedan ser expresadas, mediante *mapeos* (correspondencias) entre ellas. Por ejemplo, la Lógica Lineal permite expresar sin problemas a las lógicas clásica e intuicionista [Gir95]. Además, recientes trabajos establecen correspondencia entre la Lógica Lineal y la Lógica Modal [Ale94]. Tales resultados podrían señalar la utilidad de la Lógica Lineal como *metateoría* lógica que permita entender y explorar los límites del espacio de un gran número de lógicas y relacionarlas en forma general y precisa. En consecuencia, su utilidad como *Entorno Semántico* que permita la especificación y fomento de desarrollo de prototipos para una amplia variedad de lenguajes de programación y sistemas merece ser investigada.

Capítulo 4

Búsqueda de Pruebas en la Lógica Lineal

4.1 Pruebas Enfocadas

En 3.1.1 se estableció el objetivo de centrar los esfuerzos por obtener una clase de pruebas normales que fuesen más fáciles de obtener y que fuesen completas para un fragmento de la lógica.

Esa clase de pruebas se conoce como **Pruebas Enfocadas** (*Focusing Proofs*), cuya construcción se debe a Jean Marc Andreoli [And92]. Para obtener esa clase de pruebas, Andreoli, a partir de una investigación a profundidad a las propiedades de la teoría de pruebas de la Lógica Lineal, establece primero una transformación al sistema de secuentes original (que apartir de este punto se nombrará *sistema monádico* denotado como Σ_1), con el propósito de evitar la búsqueda de pruebas equivalentes (redundantes) originadas por el uso de las reglas de modalidad (exponenciales). A partir de éste sistema, nombrado *diádico* Σ_2 (*diadic*), establece otra transformación para evitar, en este caso, los múltiples casos de permutabilidad que originan las reglas de inferencia lógica. El sistema así obtenido, nombrado *triádico* Σ_3 (*triadic*) permite la búsqueda de pruebas enfocadas.

Las transformaciones empleadas para obtener cada sistema no sólo conservan las propiedades estructurales originales, sino que las pruebas así obtenidas resultan completas para toda la Lógica Lineal. Como consecuencia, conducen a un lenguaje sintácticamente restringido llamado LINLOG para el cual las pruebas enfocadas son naturales (de manera análoga a la normalización de fórmulas en lógica clásica a su forma clausular).

La obtención y características de las pruebas enfocadas es descrita brevemente a continuación.

4.1.1 Sistema de Inferencia *Diádico*

La primer transformación trata con las pruebas redundantes (equivalentes) causadas por el uso de recursos no restringidos mediante las reglas de inferencia correspondientes a la modalidad (exponenciales): contracción y debilitamiento. Para ello, las aplicaciones de tales reglas deberán ser *retardadas*, utilizándolas sólo cuando sean necesarias; preferentemente, cuando se alcance una hoja en el árbol de búsqueda (es decir, las fórmulas en el secuente actual sean literales). Para ello, el nuevo sistema, **Diádico** difiere del original en que se adiciona un elemento en los secuentes que denota un tanque Θ que contiene los recursos de uso no limitado:

fórmulas prefijadas con ?. De esa manera, el secuente diádico está constituído por una parte de recursos para uso no limitado (Θ) y una parte lineal (Γ):

$\vdash \Theta : \Gamma$ corresponde al secuente monadico original $\vdash ?\Theta, \Gamma$ (en donde cada fórmula en Θ está prefijada implícitamente por ?).

Notablemente, en este sistema desaparecen las reglas de Contracción y Debilitamiento (como reglas explícitas).

El siguiente teorema[1] garantiza la *completez* del nuevo sistema:

Teorema 1 *Sean Θ y Γ multiconjuntos de fórmulas. El secuente $\Theta : \Gamma$ es derivable en el sistema diádico Σ_2 si y sólo si el secuente correspondiente $?\Theta,\ \Gamma$ es derivable en Σ_1; formalmente,*

$$\begin{array}{c} \vdash \Theta \ : \ \Gamma \ \textit{si y sólo si} \vdash ?\Theta, \Gamma \\ \Sigma_2 \ \mapsto \ \Sigma_1 \end{array}$$

4.1.2 Sistema de Inferencia *Triádico*

Para evitar pruebas redundates causadas por las permutaciones en el orden de aplicación de las reglas lógicas, Andreoli establece los siguientes criterios:

Cuando una regla de inferencia va a ser aplicada a un nodo (secuente) dado, deben ser tomadas dos decisiones en este orden:

1. Escoger una fórmula (no atómica) llamada **Principal** (o activa),
2. Escoger una instancia de la regla de inferencia asociada al conectivo *más externo* de la fórmula principal seleccionada.

Tales decisiones se toman mediante una distinción entre los conectivos en base a sus propiedades:

Asíncronos : estos conectivos no requieren hacer elecciones[2] por parte del procedimiento de búsqueda, puesto que hay una y sólo una instancia de la regla de inferencia correspondiente, no importa el orden en cómo son aplicadas, permitiendo por tanto que su aplicación sea inmediata y determinística (e inclusive pueda realizarse en paralelo).

Pertenecen a esta clase:

- Multiplicativos: $\perp$, $⅋$, ?.
- Aditivos: $\top$, &, $\forall$.

Síncronos : causan que el procedimiento de prueba realice una selección correspondiente para escoger adecuadamente la instancia adecuada de la regla en cuestión, (involucrando con ello la partición del contexto disponible). Debido a que existen en general muchas maneras posibles para establecer una de tales opciones, estas reglas son fuente de no-determinismo tipo "no sé", ampliando con ello notablemente el espacio de búsqueda.

Tales conectivos son (justamente los duales de los asíncronos):

[1] Para su demostración, consultar [And92]

[2] Es decir, decidir cómo optar instanciar las reglas de inferencia correspondientes con las fórmulas disponibles en el secuente al momento de realizar la prueba.

- Multiplicativos: 1, $\otimes$, $!$.
- Aditivos: 0, $\oplus$, $\exists$.

De hecho, como puede apreciarse, la clasificación de los conectivos obedece a la dualidad: los conectivos son duales entre sí para cada grupo.

Dentro de ésta notación, una fórmula se llama asíncrona (respectivamente, síncrona) si el conectivo en el nivel superior (más externo) es asíncrono (respectivamente síncrono)[3].

Adicionalmente, la propiedad de *Reversibilidad* de las reglas de inferencia se define de la siguiente manera: la conclusión de la regla de inferencia asociada a un conectivo asíncrono es derivable si y sólo si todas sus premisas son derivables (lo cual no se cumple para los conectivos síncronos) [And92].

Gracias a la clasificación establecida, las restricciones en el criterio para seleccionar la fórmula principal son las siguientes:

1. Si el secuente contiene algunas fórmulas asíncronas, cualquiera de ellas puede ser seleccionada inmediata y aleatoriamente como fórmula principal.
2. Cuando todas las fórmulas asíncronas han sido descompuestas, entonces una fórmula (síncrona) principal debe ser seleccionada no determinísticamente. Tan pronto como es seleccionada, la búsqueda puede *enfocarse* en ella; es decir, subsecuente y sistemáticamente seleccionar como fórmula principal a cada subfórmula que la compone. Tal fórmula principal será así descompuesta hasta que se alcance alguna fórmula no síncrona (atómica ó asíncrona).

De esta forma, en una prueba normal cada fórmula es vista como una sucesión de etapas constituídas por conectivos síncronos y asíncronos (recordar los árboles sintácticos, pág. 27); cada etapa síncrona es descompuesta en una **sección crítica** (en donde el procedimiento de búsqueda no debe ser interrumpido) llamada *sección crítica de enfocamiento* de la prueba. Como convención para uniformidad, cuando una sección crítica de enfocamiento alcanza a un átomo negativo como fórmula principal, la regla de identidad debe ser aplicada.

Los secuentes triádicos son secuentes diadicos más un tercer campo (parte activa):

- En una etapa asíncrona tienen la forma $\Theta : \Gamma \Uparrow L$, donde L es una lista ordenada de fórmulas que actúa como una pila en la cual cada fórmula es desempilada para ser descompuesta, si es asíncrona se descompone inmediatamente y sus subfórmulas regresan a la pila; si es una fórmula modalizada (prefijada con el exponencial ?) es acumulada en el tanque Θ. De otra forma (síncrona, modalizada por ! ó literal) es acumulada en el contexto Γ para ser descompuesta una vez terminadas todas las asíncronas.
- Y consecuentemente en una etapa síncrona: $\Theta : \Gamma \Downarrow F$; donde F denota la fórmula (principal *enfocada*) que es elegida no determinísticamente apartir del contexto, ya sea del *acumulador* Γ o del tanque Θ una vez estando vacía la pila ($\Uparrow$).

El sistema de secuentes *triádico* está constituído de la siguiente manera:

Θ denota un conjunto de fórmulas, Γ y Δ son multiconjuntos de fórmulas no asíncronas.

[3] Esta terminología tiene su origen en las nociones de concurrencia computacional, en donde la clase de conectivos síncronos introduce un punto de sincronización (y por tanto, secuencialidad) entre dos procesos [And92].

- Reglas de Inferencia Lógica:
  - *Asíncronas*

$$\frac{\vdash \Theta : \Gamma, \Uparrow L}{\vdash \Theta : \Gamma, \Uparrow L, \bot}[\bot] \quad \frac{\vdash \Theta : \Gamma, \Uparrow F \qquad \vdash \Theta : \Gamma, \Uparrow G}{\vdash \Theta : \Gamma, \Uparrow L, F \& G}[\&] \quad \frac{\vdash \Theta : \Gamma \Uparrow L, F, G}{\vdash \Theta : \Gamma, \Uparrow L, F \parr G}[\parr]$$

$$\frac{}{\vdash \Theta : \Gamma, \Uparrow L, \top}[\top] \quad \frac{\vdash \Theta, F : \Gamma \Uparrow L}{\vdash \Theta : \Gamma \Uparrow L, ?F}[?] \quad \frac{\vdash \Theta : \Gamma \Uparrow L, F[c/x]}{\vdash \Theta : \Gamma \Uparrow L, \forall x.F}[\forall]$$

  - *Síncronas*

$$\frac{}{\vdash \Theta : \Downarrow 1}[1] \quad \frac{\vdash \Theta : \Gamma \Downarrow F \qquad \vdash \Theta : \Delta \Downarrow G}{\vdash \Theta : \Gamma \Downarrow F \otimes G}[\otimes] \quad \frac{\vdash \Theta : \Gamma \Uparrow F}{\vdash \Theta : \Gamma \Downarrow !F}[!]$$

$$\frac{\vdash \Theta : \Gamma \Downarrow F[t/x]}{\vdash \Theta : \Gamma \Downarrow \exists x.F}[\exists] \quad \frac{\vdash \Theta : \Gamma \Downarrow F}{\vdash \Theta : \Gamma \Downarrow F \oplus G}[\oplus_1] \quad \frac{\vdash \Theta : \Gamma \Downarrow G}{\vdash \Theta : \Gamma \Downarrow F \oplus G}[\oplus_2]$$

- Reacciones
  - *Reacción $R \Uparrow$:* F no es asíncrona: (acumularla en Γ)

$$\frac{\vdash \Theta : \Gamma, F \Uparrow L}{\vdash \Theta : \Gamma \Uparrow \ L, F}[R \Uparrow]$$

  - *Reacción $R \Downarrow$:* F no es síncrona ni átomo positivo: (descomponer inmediatamente, termina sección crítica de enfocamiento)

$$\frac{\vdash \Theta : \Gamma \Uparrow F}{\vdash \Theta : \Gamma \Downarrow F}[R \Downarrow]$$

- Identidades:

$$\frac{}{\vdash \Theta : X \Downarrow \ X^{\perp}}[Id_1] \quad \frac{}{\vdash \Theta, X : \Downarrow \ X^{\perp}}[Id_2]$$

- Decisiones: F no es átomo positivo: (comienza Sección Crítica de Enfocamiento)

$$\frac{\vdash \Theta : \Gamma \Downarrow \ F}{\vdash \Theta : \Gamma, F \Uparrow}[D_1] \quad \frac{\vdash \Theta, F : \Gamma \Downarrow \ F}{\vdash \Theta, F : \Gamma \Uparrow}[D_2]$$

Las reglas lógicas mantienen la etapa de prueba asíncrona (resp. síncrona) actual, salvo la regla [!] que inmediatamente termina la sección de enfocamiento. La regla [&] dice que ambas subfórmulas tienen que ser probadas (repitiendo el contexto) eventualmente en paralelo; la regla [$\parr$] es una re-escritura sintáctica simple; la regla [?] almacena a la fórmula exponencial en la parte no limitada (Θ) del contexto tan pronto como aparezca (es decir, la almacena pero puede ser recuperada de ahí en cualquier momento); y la regla [$\forall$] dice que la meta deberá ser probada parametrizada con una constante arbitraria c.

La regla para [$\otimes$] involucra una opción para dividir el contexto en dos partes disjuntas (pero nótese que la parte exponencial Θ no cambia); la regla [$\oplus$] también introduce una alternativa para escoger entre cuál de los dos disjuntos deberá intentar probarse; la regla [!] hace irreversible la decisión de abandonar al exponencial (no se tiene la opción de almacenar $!A$ en Θ debido a que sólo ? es el exponencial correcto para marcar a los secuentes de la parte derecha); finalmente, la regla $[\exists]$ significa encontrar un término concreto t que permitirá unificar (eventualmente) en el axioma de Identidad (Id_1, Id_2) .

Las reglas de reacción para cada etapa corresponden a acumular fórmulas no asíncronas ($[R \Uparrow]$) en Γ hasta que todas las asíncronas hayan sido descompuestas; y a terminar la sección crítica de enfocamiento ($[R \Downarrow]$) ante la presencia de una fórmula activa no síncrona.

Las reglas de identidad separan la búsqueda por átomos duales en el acumulador Γ ($[Id_1]$), y en el tanque Θ ($[Id_2]$), (con la condición en esta última que el acumulador esté vacío). Tales reglas son activadas ante la presencia de un átomo negativo como fórmula principal activa en una etapa síncrona exclusivamente.

Por su parte y de especial importancia, las reglas de Decisión realizan la transición entre etapas, partiendo de una etapa asíncrona hacia la síncrona: se escoge alguna fórmula síncrona (ó átomo negativo) del acumulador Γ ($[D_1]$) o del tanque Θ ($[D_2]$). Tal opción es no determinística, involucrando además una fuente de incertidumbre especial: mientras que $[D_1]$ *consume* a cada fórmula de Γ en una prueba exitosa, la aplicación de $[D_2]$ mantiene la fórmula a ser intentada en el tanque (esto es obvio, ya que su uso es ilimitado); evidentemente, esta regla es utilizada cuando todas las fórmulas en el acumulador no ofrecieron una prueba exitosa, por lo que se recurre a las fórmulas del tanque. El peligro latente es que puede incurrirse en ciclos: debe llevarse la cuenta de cada fórmula del tanque que haya sido seleccionada en una rama de prueba, para no ser utilizada dos o más veces con el mismo contexto[4]. Estas reglas se basan en la "confianza" que la fórmula seleccionada como activa conducirá eventualmente (*lazy*) a una prueba exitosa.

Notablemente, conducir la selección de la fórmula principal en estas reglas auxiliadas por los árboles sintácticos (ver 3.1.1, pág. 27) correspondientes permitirá reducir retrocesos y lograr una búsqueda más efectiva. Esta selección estará determinada por la disponibilidad de subfórmulas en el contexto que permitan a las subfórmulas de la fórmula principal seleccionada correspondencia entre duales y/o disponibilidad de aplicar los axiomas para las constantes 1 y $\top$.

Para apreciar la funcionalidad de este sistema de inferencia, se utilizará el ejemplo 2 (de la pág. 22):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{Rama\ izquierda}{\vdash a\oplus b :!a^{\perp}\otimes !b^{\perp} \Uparrow}[D_1]}{\vdash :!a^{\perp}\otimes !b^{\perp} \Uparrow ?(a\oplus b)}[?]}{\vdash :\Uparrow (!a^{\perp}\otimes !b^{\perp}), ?(a\oplus b)}[R\Uparrow]}{\vdash :\Uparrow (!a^{\perp}\otimes !b^{\perp}) \parr ?(a\oplus b)}[\parr] \qquad \dfrac{\dfrac{\dfrac{\dfrac{Rama\ derecha:}{\vdash a,b : !(a^{\perp}\& b^{\perp}) \Uparrow}[D_1]}{\vdash :\underline{!(a^{\perp}\& b^{\perp})} \Uparrow ?a, ?b}[?]}{\vdash :\Uparrow \underline{!(a^{\perp}\& b^{\perp})}, \underline{?a, ?b}}[R\Uparrow]}{\vdash :\Uparrow !(a^{\perp}\& b^{\perp}) \parr (?a \parr ?b)}[\parr]}{\vdash :\Uparrow ((!a^{\perp}\otimes !b^{\perp}) \parr ?(a\oplus b))\&(!(a^{\perp}\& b^{\perp}) \parr (?a \parr ?b))}[\&]$$

Rama izquierda:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{-fail-}{\vdash a\oplus b :\Downarrow a^{\perp}}}{\vdash a\oplus b : a^{\perp} \Uparrow}[D_1] \leftarrow \ i_1}{\vdash a\oplus b :\Uparrow a^{\perp}}[R\Uparrow]}{\vdash a\oplus b :\Downarrow !a^{\perp}}[!] \qquad \dfrac{\dfrac{\dfrac{\dfrac{-fail-}{\vdash a\oplus b :\Downarrow b^{\perp}}}{\vdash a\oplus b : b^{\perp} \Uparrow}[D_1] \leftarrow \ i_2}{\vdash a\oplus b :\Uparrow b^{\perp}}[R\Uparrow]}{\vdash a\oplus b :\Downarrow !b^{\perp}}[!]}{\vdash a\oplus b :\Downarrow !a^{\perp}\otimes !b^{\perp}}[\otimes]$$

Rama derecha:

[4] Técnicamente se está permitiendo la implantación de programas que contengan bucles infinitos.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a,b :\Downarrow a^{\perp}}[Id_2]}{\vdash a,b : a^{\perp} \Uparrow}[D_1]}{\vdash a,b :\Uparrow a^{\perp}}[R\Uparrow] \qquad \dfrac{\dfrac{\dfrac{}{\vdash a,b :\Downarrow b^{\perp}}[Id_2]}{\vdash a,b : b^{\perp} \Uparrow}[D_1]}{\vdash a,b :\Uparrow b^{\perp}}[R\Uparrow]}{\vdash a,b :\Uparrow a^{\perp}\&b^{\perp}}[\&]}{\vdash a,b :\Downarrow !(a^{\perp}\&b^{\perp})}[!]$$

Aparentemente, la fórmula no es demostrable con las elecciones $[D_1]$ en la sub rama izquierda (i_1) y $[D_1]$ (i_2) en la sub rama derecha de la rama izquierda principal. Eligiendo a las fórmulas del tanque a ser descompuestas (mediante $[D_2]$, lo cual ilustra el no determinismo arriba mencionado) se tiene lo siguiente

para i_1:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a\oplus b : a \Downarrow a^{\perp}}Id_1}{\vdash a\oplus b : a^{\perp}, a \Uparrow}[D_1]}{\vdash a\oplus b : a^{\perp} \Uparrow a}[R\Uparrow]}{\vdash a\oplus b : a^{\perp} \Downarrow a}[R\Downarrow]}{\vdash a\oplus b : a^{\perp} \Downarrow a\oplus b}[\oplus_1]}{\vdash \underline{a\oplus b} : a^{\perp} \Uparrow}[D_2]$$

para i_2:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[D_1] \to fail, [D_2] \to endless\ loop!}{\vdash a\oplus b : b^{\perp}, a \Uparrow}?}{\vdash a\oplus b : b^{\perp} \Uparrow a}[R\Uparrow]}{\vdash a\oplus b : b^{\perp} \Downarrow a}[R\Downarrow]}{\vdash a\oplus b : b^{\perp} \Downarrow a\oplus b}[\oplus_1](*)}{\vdash \underline{a\oplus b} : b^{\perp} \Uparrow}[D_2]$$

Nótese que la elección de $\oplus_1$ (*) obedece a una búsqueda realizada en el orden de lectura de las subfórmulas de izquierda a derecha (búsqueda primero en profundidad). En este caso, ello conduce a bucles inútiles como consecuencia que en la aplicación de la regla de decisión $[D_2]$ se intenta con cada fórmula (de izquierda a derecha) del tanque *desde el principio* para cada nueva etapa, y se continúa con la siguiente fórmula ante el fallo de la última. Para evitarlo, la etapa de decisión $[D_2]$ debe entenderse *globalmente* para toda la rama, en vez de localmente (como el caso de $[D_1]$, ya que el contexto lineal así enriquecido es posteriormente consumido). De esa manera, pueden "marcarse" las fórmulas en el tanque una vez intentadas y se debe continuar con la siguiente aún cuando la actual no haya resultado en una prueba fallida.

Recordando el árbol sintáctico para esta rama, la elección adecuada (por la propiedad de subfórmula) es $[\oplus_2]$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a\oplus b : b \Downarrow b^{\perp}}[Id_1]}{\vdash a\oplus b : b^{\perp}, b \Uparrow}[D_1]}{\vdash a\oplus b : b^{\perp} \Uparrow b}[R\Uparrow]}{\vdash a\oplus b : b^{\perp} \Downarrow b}[R\Downarrow]}{\vdash a\oplus b : b^{\perp} \Downarrow a\oplus b}[\oplus_2]$$

De igual forma para el sistema diádico, el siguiente teorema garantiza la completez de la transformación al nuevo sistema triádico[5]:

Teorema 2 *Sean* Θ *y* Γ *multiconjuntos de fórmulas,* Γ *no contiene fórmulas asíncronas, y sea* L *una lista ordenada de fórmulas. El secuente* $\Theta : \Gamma \Uparrow L$ *es derivable en el sistema triádico* Σ_3 *si y sólo si el secuente correspondiente* $\Theta : \Gamma, L$ *es derivable en* Σ_2*; formalmente,*

$$\vdash \Theta : \Gamma \Uparrow L \quad \textit{si y sólo si} \quad \vdash \Theta : \Gamma, L$$
$$\Sigma_3 \;\mapsto\; \Sigma_2$$

4.2 La forma normal Linlog

Las pruebas enfocadas tienen una interpretación simple y computacionalmente importante en un fragmento de la Lógica Lineal, llamado Linlog [And92]. Las restricciones sintácticas que definen este fragmento no inducen alguna restricción en la expresividad del lenguaje mismo: cualquier fórmula de la Lógica Lineal puede ser mapeada en su forma Linlog mediante un procedimiento de normalización, preservando la estructura y demostrabilidad de las pruebas correspondientes.

4.2.1 Sintaxis y Semántica Operacional

Linlog está constituído por dos clases de fórmulas: *metas y métodos.* Hay dos tipos de metas:

- **Metas Elementales:** formadas por combinaciones de **átomos positivos** conectados por conectivos asíncronos, ? se aplica a átomos únicamente

$$g ::= X \mid ?X \mid \bot \mid g \& g \mid \top \mid g \parr g \mid \forall x.g$$

- **Metas:** constituídas por metas elementales conectadas por conectivos síncronos, en donde ! se aplica sólo a metas elementales

$$G ::= g \mid !g \mid 1 \mid G \otimes G \mid 0 \mid G \oplus G \mid \exists x.G$$

Las metas son fórmulas positivas (no contienen átomos negativos) en las cuales un conectivo síncrono (resp. asíncrono) nunca debe ocurrir en el ámbito de un exponencial de la misma clase (síncrona o asíncrona).

Los **Métodos** son fórmulas básicas de la forma:

$$\forall \vec{x}(X_1 \parr X_2, \ldots, \parr X_r \circ\!\!- G)$$

donde $X_1, X_2, \ldots, X_r$ son átomos positivos ($r \geq 1$), que forman la *cabeza* del método y G es una meta llamada *cuerpo.* Definimos a una *instancia* de un método M como la pareja $\Phi_0 \circ\!\!- G_0$ en donde Φ_0 es un multiconjunto de átomos positivos *básicos* y G_0 es una meta básica, de tal forma que existe una *sustitución básica* σ que permite verificar:

$$\sigma \circ \{X_1, \ldots, X_r\} = \Phi_0 \text{ y } \sigma \circ G = G_0$$

[5] Consultar la demostración en [And92].

Una **consulta** LINLOG es una pareja $< \mathcal{P}; g >$ donde $\mathcal{P}$ es un conjunto de métodos (el programa) y g es una meta elemental.

Una *ejecución* LINLOG consiste de la búsqueda de pruebas del secuente $!\mathcal{P} \vdash g$, es decir: $\vdash \ ?\,\mathcal{P}^{\perp}, g$ (en la forma asímetrica por la derecha). Mediante los teoremas 1 y 2 tal búsqueda puede llevarse a cabo aplicando el procedimiento de búsqueda BUSCAR (2.4, pág. 21) al secuente triadico $\vdash \mathcal{P}^{\perp} : \Uparrow g$, en donde:

- El dual del programa $\mathcal{P}$ actúa como un conjunto de recursos de uso no limitado que proporciona la fuente de "alimento" computacional.
- La meta g representa el **estado inicial** del procesamiento en términos de recursos restringidos (lineales).

La sintaxis de metas y métodos presentada está diseñada en forma tal que las pruebas en el sistema triadico de las consultas LINLOG tienen una estructura especial, estando compuestas repetidamente por:

1. Una etapa de aplicación de *reglas de inferencia lógica* mezcladas con las reglas de *Reacción* $[R \Uparrow]$, $[R \Downarrow]$, conteniendo solamente secuentes de la forma:

   $$\vdash \mathcal{P}^{\perp}, \Psi : \Phi \Downarrow G \text{ ó } \vdash \mathcal{P}^{\perp}, \Psi : \Phi \Uparrow \mathcal{G}$$

   donde: Ψ, Φ denotan multiconjuntos de átomos positivos, G es una meta y $\mathcal{G}$ es una lista ordenada de metas elementales.

2. Una etapa de reglas de inferencia comenzando con un secuente de la forma

   $$\vdash \mathcal{P}^{\perp}, \Psi : \Phi \Uparrow$$

   Esta es la única etapa en donde los métodos del programa entran en acción mediante el paso de inferencia *progresión*:

   $$\frac{\vdash \mathcal{P}^{\perp}, \Psi, \Psi_0 : \Phi \Downarrow G_0}{\vdash \mathcal{P}^{\perp}, \Psi, \Psi_0 : \Phi, \Phi_0 \Uparrow}[\circ\!-]$$

   en donde: $\Psi_0, \Phi_0 \circ\!- G_0$ es una instancia de un método en $\mathcal{P}$.

De esa manera, es posible especializar el sistema de inferencia triadico Σ_3 en un sistema de inferencia $\Sigma_3[\mathcal{P}]$, en donde el paso de progresión reemplaza a las reglas de decisión $([D_1], [D_2])$ e identidad $([Id_1], [Id_2])$ del sistema general.

Las ocurrencias de $\mathcal{P}^{\perp}$ en las pruebas dentro del sistema de inferencia $\Sigma_3[\mathcal{P}]$ son implícitas:

- $\mathcal{P}$ es el programa LINLOG actual.
- $\Psi, \Psi_0, \Phi, \Phi_1, \Phi_2$ son multiconjuntos de átomos positivos.
- g, g_1, g_2 denotan metas elementales.
- $\mathcal{G}$ denota a una lista ordenada de metas elementales.
- G, G_1, G_2 son metas.
- X denota un átomo positivo.

- **Reglas de Inferencia Lógica:**

$$\frac{\vdash \Psi : \Phi, \Uparrow \mathcal{G}}{\vdash \Psi : \Phi, \Uparrow \mathcal{G}, \bot}[\bot] \quad \frac{\vdash \Psi : \Phi, \Uparrow \mathcal{G}, g_1 \qquad \vdash \Psi : \Phi, \Uparrow \mathcal{G}, g_2}{\vdash \Psi : \Phi, \Uparrow \mathcal{G}, g_1 \& g_2}[\&] \quad \frac{\vdash \Psi : \Phi \Uparrow \mathcal{G}, g_1, g_2}{\vdash \Psi : \Phi, \Uparrow \mathcal{G}, g_1 ⅋ g_2}[⅋]$$

$$\frac{}{\vdash \Psi : \Phi, \Uparrow \mathcal{G}, \top}[\top] \quad \frac{\vdash \Psi, X : \Phi \Uparrow \mathcal{G}}{\vdash \Psi : \Phi \Uparrow \mathcal{G}, ?X}[?] \quad \frac{\vdash \Psi : \Phi \Uparrow \mathcal{G}, g[c/x]}{\vdash \Psi : \Phi \Uparrow \mathcal{G}, \forall x.g}[\forall]$$

$$\frac{}{\vdash \Psi : \Downarrow 1}[1] \quad \frac{\vdash \Psi : \Phi_1 \Downarrow F \qquad \vdash \Psi : \Phi_2 \Downarrow G}{\vdash \Psi : \Phi_1, \Phi_2 \Downarrow F \otimes G}[\otimes] \quad \frac{\vdash \Psi : \Phi \Uparrow g}{\vdash \Psi : \Phi \Downarrow !g}[!]$$

$$\frac{\vdash \Psi : \Phi \Downarrow G[t/x]}{\vdash \Psi : \Phi \Downarrow \exists x.G}[\exists] \quad \frac{\vdash \Psi : \Phi \Downarrow G_1}{\vdash \Psi : \Phi \Downarrow G_1 \oplus G_2}[\oplus_1] \quad \frac{\vdash \Psi : \Phi \Downarrow G_2}{\vdash \Psi : \Phi \Downarrow G_1 \oplus G_2}[\oplus_2]$$

- **Reacciones:**

$$\frac{\vdash \Psi : \Phi, X \Uparrow \mathcal{G}}{\vdash \Psi : \Phi \Uparrow \mathcal{G}, X}[R \Uparrow] \quad \frac{\vdash \Psi : \Phi \Uparrow g}{\vdash \Psi : \Phi \Downarrow g}[R \Downarrow]$$

Consistentemente, el siguiente teorema garantiza la correspondencia de demostrabilidad con el sistema de inferencia original [6]:

Teorema 3 *Sea $< \mathcal{P}; g >$ una consulta* Linlog*, el secuente $!\mathcal{P} \vdash g$ es derivable en Lógica Lineal si y sólo si $\vdash : \Uparrow g$ es derivable en $\Sigma_3[\mathcal{P}]$; formalmente,*

$$\begin{array}{c} !\mathcal{P} \vdash g \quad \textit{si y sólo si} \vdash_{\mathcal{P}} : \Uparrow \; g \\ \Sigma_1 \;\mapsto\; \Sigma_3[\mathcal{P}] \end{array}$$

Este sistema de inferencia manipula secuentes que en general pueden contener dos tipos de contexto, pudiendo por ello caracterizarle en dos cláusulas:

1. *Contexto No Uniforme* : el contexto está constituído por multiconjuntos de metas básicas con a lo más una meta no elemental, por tanto, el sistema de inferencia debe seleccionar aleatoriamente a una meta y descomponerla usando la regla correspondiente a su conectivo más externo (superior).

   El paso de selección involucra no determinismo del tipo "no importa".

   El paso de descomposición puede involucrar no determinismo clase "no sé" si la meta seleccionada no es elemental (y por tanto síncrona, por lo que al establecer cierto orden en la disposición de las fórmulas pueden evitarse retrocesos).

2. *Contexto Uniforme* : el contexto está constituído por átomos positivos (algunos posiblemente almacenados en el tanque Ψ), por lo cual, debe encontrarse una instancia de un método dentro del programa $\mathcal{P}$ cuya cabeza empate exactamente a un submulticonjunto de los átomos (modalizados o no) en el contexto; entonces, reemplazar en el contexto aquellos y sólo aquellos átomos del multiconjunto no modalizado (es decir, en el acumulador) por el cuerpo del método seleccionado.

   La selección tanto del método como la del multiconjunto del contexto para activarlo involucran no determinismo del tipo "no sé".

[6] Consultar la demostración en [And92]

4.2.2 Normalización a la forma LinLog

El procedimiento de normalización establece una transformación de cualquier fórmula F de Lógica Lineal en una consulta LinLog de tal forma que las pruebas enfocadas de F son *isomórfas* a las pruebas en el sistema de inferencia LinLog $\Sigma_3[\mathcal{P}]$ de la consulta LinLog correspondiente, salvo algunas convenciones de nombrado irrelevantes [And92].

Dada una fórmula F_0, los pasos para llevarla a su forma normal LinLog son:

1. Ver a F_0 como una meta elemental, identificando a la subfórmula más profunda que le evite cumplir la sintaxis de meta elemental.

2. Sea g la fórmula obtenida al reemplazar en F_0 cada una de las subfórmulas identificadas en el paso anterior por un átomo positivo nuevo u, diferente para cada subfórmula. Por construcción, g es una meta elemental, la cual define al componente g de la consulta LinLog.

3. Para cada átomo u introducido en F_0 para reemplazar una subfórmula F, intentar ver a la fórmula $u \circ\!- F$ como un conjunto de métodos LinLog (conectados por &). Esto podría conducir a nuevos reemplazos de subfórmulas por átomos nuevos, los cuales son tratados recursivamente en la misma forma y produciendo a su vez más métodos.

4. El conjunto de todos los métodos producidos en el paso anterior constituye el programa (componente $\mathcal{P}$) de la forma normal.

Antes de presentar al algoritmo, deben hacerse algunas definiciones:

- Se define a un *quasi-método* como la pareja $[\Phi; \Upsilon]$, donde Φ es un multiconjunto de átomos positivos y Υ es un multiconjunto de metas; un quasi-método se dice *estricto* si Φ es no vacío.

- Se define al operador * sobre los quasi-métodos $\mathcal{S}_1$ y $\mathcal{S}_2$:

$$\mathcal{S}_1 * \mathcal{S}_2 =_{def} \bigcup_{\substack{[\Phi_1;\Upsilon_1]\in\mathcal{S}_1 \\ [\Phi_2;\Upsilon_2]\in\mathcal{S}_2}} \{[\Phi_1, \Phi_2; \Upsilon_1, \Upsilon_2]\}$$

  $\mathcal{S}_1$ * $\mathcal{S}_2$ denota el conjunto de quasi-métodos combinando los elementos de cada uno, uniendo las cabezas y cuerpos por parejas, respectivamente.

- Se define al operador $||.||$ sobre un conjunto de quasi-métodos $\mathcal{S}$ como el conjunto de métodos correspondientes $||\mathcal{S}||$:

$$||\mathcal{S}|| =_{def} \bigcup_{[X_1,X_2,\ldots,X_r;G_1,G_2,\ldots,G_s]\in\mathcal{S}} \{[X_1 \parr X_2 \parr \ldots \parr X_r \circ\!- G_1 \otimes G_2 \otimes \ldots \otimes G_s]\}$$

  cuando $s = \emptyset$, el cuerpo del método es reducido a la constante 1.

El algoritmo de normalización está constituído por las siguientes funciones [And92]:

La estructura Global $\mathcal{P}$ denota al programa LINLOG.

function $\mathsf{Normalize}$(F : formula);
  regresa : consulta LINLOG
  $\mathcal{P} := \emptyset$;
  $g := \mathsf{ngoal}$(F);
  return $(\mathcal{P};\ g)$.

function $\mathsf{ngoal}(F$: formula);
  regresa : meta elemental

En caso de:	Regresar:
$F_1 \parr F_2$	ngoal$(F_1) \parr$ ngoal(F_2)
$F_1 \& F_2$	ngoal(F_1) & ngoal(F_2)
$\bot$	$\bot$
$\top$	$\top$
$?F'$	$?\ \mathsf{natom}(F')$
de otra forma	$\mathsf{natom}(F)$

Esta función examina a la fórmula de entrada F hasta que alcance las ocurrencias de subfórmulas, en donde, para que F sea una meta elemental, tales subfórmulas deben ser átomos positivos, aplicando recursivamente la función **natom** a cada una de ellas.

function $\mathsf{natom}(F$: formula);
  regresa : átomo positivo.
  Si F es un átomo positivo entonces:
    regresar F;
  de lo contrario:
    Sea X un *nuevo átomo positivo* (constante Skolem);
    Actualizar $\mathcal{P}$:
    $\mathcal{P} := \mathcal{P} \cup \|\ \mathsf{nmeth}(X^{\bot} \otimes F)\|$;
  Regresar: X.

Esta función también es elemental: si F no es átomo positivo, se introduce una nueva constante Skolem X la cual es regresada y establece una correspondencia entre la definición de X y F mediante el conjunto métodos $(X \circ\!\!- F)$ los cuales son incorporados a $\mathcal{P}$. Para evitar *dualización múltiple* dentro de F, es más fácil intentar mapear la fórmula dual $X^{\bot} \otimes F$ en el dual del conjunto de métodos obtenidos mediante la invocación de nmeth.

function $\mathsf{nmeth}(F$: formula);
  regresa: conjunto de quasi-métodos
  Si F es un átomo negativo $X^{\bot}$ entonces:
    regresar $\{[X;\emptyset]\}$;

De lo contrario:

En caso de:	Regresar:
$F_1 \otimes F_2$	$\mathsf{nmeth}(F_1)$ * $\mathsf{nmeth}(F_2)$
$F_1 \oplus F_2$	$\mathsf{nmeth}(F_1) \cup \mathsf{nmeth}(F_2)$
1	$\{[\emptyset; \emptyset]\}$
0	$\emptyset$
$!F'$	$\{[\emptyset\ ;\ !\ \mathsf{ngoal}(F')]\}$
de otra forma	$\{[\emptyset\ ;\ \mathsf{ngoal}(F)]\}$

Esta función se basa en el hecho de que cualquier sección crítica de enfocamiento (de una prueba enfocada para F) puede mapearse a un *paso de progresión* LINLOG ($[\multimap]$ extendido para tratar con quasi-métodos no estrictos -que no tienen cabeza-) usando uno de los métodos de $\mathsf{nmeth}(F)$. Por ejemplo, si F es de la forma $F_1 \oplus F_2$, una sección crítica de enfocamiento para F necesariamente depende ya sea de la sección de F_1 ó de F_2; para ambos casos puede establecerse la correspondencia con un paso de progresión utilizando un quasi-método en $\mathsf{nmeth}(F_i)$ $(i = 1, 2)$; por lo tanto, la sección crítica de enfocamiento para F puede corresponderse con un paso de progresión utilizando un quasi-método en *nmeth(F_1)* $\cup$ *nmeth(F_2)*.

Correspondientemente, para el caso de $\otimes$, la sección crítica de enfocamiento para $F = F_1 \otimes F_2$ estará constituída por la combinación de las secciones críticas correspondientes para cada subfórmula F_i.

A continuación se detalla la normalización LINLOG para una fórmula arbitraria:

Sea $F = (a^\perp \otimes b) \parr a \parr b^\perp$. Esta fórmula no obedece a la sintaxis de meta LINLOG: en la subfórmula $F_1 = (a^\perp \otimes b)$ ocurre un conectivo síncrono en el ámbito de uno asíncrono y un átomo negativo; de igual forma, en la subfórmula $F_2 = b^\perp$ ocurre un átomo negativo. Si F_1 y F_2 fueran átomos positivos, entonces $F = F_1 \parr a \parr F_2$ estaría en forma normal.

Para la construcción de los métodos correspondientes, se reemplaza a cada subfórmula F_1 y F_2 por los átomos positivos nuevos u_1 y u_2 respectivamente, donde se trata de ver a las fórmulas $u_1 \multimap F_1$ y $u_2 \multimap F_2$ como los métodos M_1 y M_2 respectivamente:

$$\begin{array}{cc}
M_1 \stackrel{\text{def}}{=} & M_2 \stackrel{\text{def}}{=} \\
u_1 \multimap F_1 & u_2 \multimap F_2 \\
u_1 \parr F_1^\perp & u_2 \parr F_2^\perp \\
u_1 \parr a \parr b^\perp & u_2 \parr b \\
u_1 \parr a \multimap b & u_2 \parr b \multimap 1
\end{array}$$

Los átomos u_1, u_2 son nuevos (no ocurren en alguna otra parte de F cuando son introducidos), siendo identificadores de ocurrencias sintácticas de las subfórmulas respectivas. La operación de renombrado es estrictamente similar al paso de Skolem pero a nivel de subfórmulas.

La consulta LINLOG para este ejemplo es $< \{u_1 \parr a \multimap b, u_2 \parr b \multimap 1\}; u_1 \parr a \parr u_2 >$.

Los primeros pasos de la prueba LINLOG de F son los mismos que de la prueba enfocada para F:

$$\dfrac{\dfrac{\vdots}{\vdash: F_1, a, F_2 \Uparrow}[R \Uparrow]}{\vdash: \Uparrow F_1 \parr a \parr F_2}[\parr]$$

En la prueba enfocada de F, el siguiente paso consiste en la selección de F_1 (leyendo de izquierda a derecha en el acumulador) como fórmula principal mediante la regla $[D_1]$, seguida por una breve sección de enfocamiento crítico; mientras que en la prueba LINLOG, tal sección es mapeada a un paso de progresión $[\multimapinv]$ activando al método M_1 :

$$Enfocada\ F: \qquad\qquad Linlog\ <\ \mathcal{P}\ ;\ g\ >$$

$$\dfrac{\dfrac{\dfrac{}{\vdash: a \Downarrow a^{\perp}}[Id_1] \qquad \dfrac{\vdots}{\vdash: F_2 \Downarrow b}}{\vdash: a, F_2 \Downarrow F_1}[\otimes]}{\vdash: F_1, a, F_2 \Uparrow}[D_1] \qquad \dfrac{\dfrac{\vdots}{\vdash: F_2 \Downarrow b}}{\vdash: \underline{u_1, a}, F_2 \Uparrow}[\multimapinv]$$

A continuación, tanto la prueba enfocada como la prueba LINLOG proceden de igual manera:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash: F_2, b \Uparrow}}{\vdash: F_2 \Uparrow b}[R \Uparrow]}{\vdash: F_2 \Downarrow b}[R \Downarrow]$$

Nuevamente, en la prueba enfocada de F se selecciona a F_2, comenzando una nueva sección de enfocamiento crítico; mientras que en la prueba LINLOG se activa al método M_2 mediante el paso de progresión:

$$Enfocada\ F: \qquad Linlog\ <\mathcal{P}; g>$$

$$\dfrac{\dfrac{}{\vdash: b \Uparrow F_2}[Id_1]}{\vdash: F_2, b \Uparrow}[D_1] \qquad \dfrac{\dfrac{}{\vdash: \Uparrow 1}[1]}{\vdash: u_2, b \Uparrow}[\multimapinv]$$

Consistentemente, el siguiente teorema garantiza la completez de la normalización LINLOG[7]:

Teorema 4 *Sean F una fórmula bien formada en Lógica Lineal y $< \mathcal{P}; g >$ su forma normal* LINLOG *correspondiente. F es derivable si y sólo si g es derivable en el sistema de inferencia* LINLOG *utilizando $\mathcal{P}$:*

$$\vdash F \ \textit{si y sólo si} \ \vdash_{\mathcal{P}}: \Uparrow g$$

[7] Consultar la demostración en [And92].

Parte II

Diseño, Implantación y Pruebas

Capítulo 5

Diseño e Implantación

Gracias al entorno formal en el cual este trabajo se ha discutido, la definición del problema y su especificación ha sido cubierta a detalle en los capítulos precedentes, por lo que se puede pasar a detallar los aspectos propios de diseño e implantación del prototipo demostrador de teoremas en Lógica Lineal.

5.1 Consideraciones Generales

En términos generales, el prototipo aceptará como entrada a una fórmula bien formada proposicional F en Lógica Lineal, obtendrá su forma normal LINLOG (4.2) correspondiente $< \mathcal{P}; g >$ y procederá a demostrarla en el sistema de inferencia triádico LINLOG (4.2.1) $\Sigma_3[\mathcal{P}]$.

El prototipo tiene propósito de servir como herramienta para experimentar la demostración mecánica de fórmulas en Lógica Lineal. El hecho que tales fórmulas pueden representar la especificación del comportamiento de un sistema computacional interactivo, la demostración que el prototipo ofrece constituye la especificación formal de un cálculo de dicho sistema. Por tanto, criterios de desempeño no son contemplados en su implantación.

Se escogió el lenguaje PROLOG para la implantación del prototipo debido a las ventajas que ofrece:

- interfaz en forma de consultas,
- estructuras de datos dinámicas, (fundamentalmente en forma de listas),
- algoritmo de *match*, resolución y unificación los cuales permiten concentrarse en el contenido más que en la forma,
- por consiguiente, permite que el código ofrecido sea claro, fácil de entender y de mantener.

5.2 Diseño

Las fases que constituyen el prototipo son las siguientes:

- **Preprocesamiento:** fundamentalmente prepara a la fórmula de entrada F para la siguiente fase, produciendo una versión procesada F_p en la cual las implicaciones lineales ($\multimap$) han sido re escritas y la negación lineal se aplica a subfórmulas átomicas exclusivamente. Esta fase no altera en esencia a la fórmula de entrada, sino que proporciona su forma pre procesada a fin de la fase siguiente se desempeñe mejor.

- **Normalización:** esta fase obtiene la consulta Linlog de la fórmula F_p mediante las correspondientes funciones ngoal, natom y nmeth. Además de tales funciones propias de la normalización, esta fase incluye también rutinas *auxiliares* que implantan los operadores para procesamiento de quasi-métodos $(. * ., \|.\|)$.
- **Búsqueda de Pruebas:** en esta fase el sistema de inferencia triádico $\Sigma_3[\mathcal{P}]$ Linlog tiene lugar. Recibe como entrada a la consulta Linlog calculada en la fase anterior y ofrece el árbol de búsqueda de prueba (si existe) correspondiente. Tal búsqueda esta cimentada en dos rutinas principales: provea y proves las cuales corresponden a las etapas de búsqueda asíncrona y síncrona respectivamente.

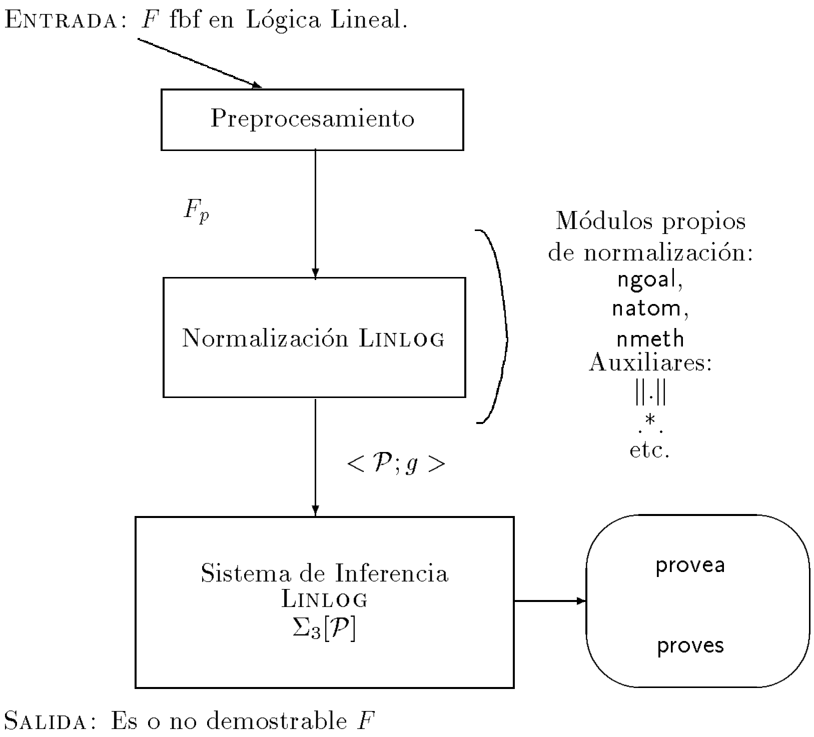

Figura 5.1: Fases del Prototipo.

Las estructuras de datos a utilizar son:

- Una fórmula bien formada proposicional F está constituída por subfórmulas bien formadas relacionadas por los conectivos lineales:

$$F ::= F_1 \multimap F_2 | F_1 \circ\!\!- F_2 | F_1 \circ\!\!-\!\!\circ F_2 | !F_1 | ?F_1 | F_1^{\perp} | F_1 \parr F_2 | F_1 \& F_2 | F_1 \otimes F_2 | F_1 \oplus F_2 | \top | \bot | 1 | 0 | X.$$

(X representa a una fórmula atómica). Se supone una prioridad en los operadores.

- Componentes de la consulta LINLOG:

  Programa P (lista de métodos) y meta elemental g.

  Método: Cabeza (multiconjunto de átomos) y Cuerpo (multiconjunto de Metas).

  Quasi-método: pareja formada por un multiconjunto de átomos y un multiconjunto de metas.

- Un Contexto dentro del procedimiento de bśqueda de pruebas está constituído por un multiconjunto[1] de átomos.

5.3 Implantación

Se presenta a continuación cada módulo que conforma al prototipo.

5.3.1 Definición de los conectivos

Lo siguiente constituye la definición de los conectivos de la Lógica Lineal en forma de operadores de Prolog, así como las constantes correspondientes a los valores de verdad y falsedad para cada grupo: (módulo `operator.pl`)

```
/**************/
/* OPERADORES */
/**************/

:-op(30,fx,~).          % Negacion Lineal - dualidad
:-op(110,xfy,#).        % Disyuncion Multiplicativa  (Par)
:-op(110,xfy,&).        % Conjuncion Aditiva        (With)
:-op(100,xfy,*).        % Conjuncion Multiplicativa  (Times)
:-op(100,xfy,+).        % Disjuncion Aditiva        (Plus)
:-op(150,xfy,->).       % Implicacion Lineal
:-op(150,xfy,<-).       % if Lineal
:-op(150,xfy,<->).      % iff Lineal

/* Constantes:

  mTrue : (one) Verdad multiplicativa
  mFalse: (bottom) Falsedad multiplicative
  aTrue : (Top) additive true
  aFalse: (zero) additive false
*/
```

[1]En el caso del tanque de recursos no modalizado (Θ en Σ_3 y Ψ en $\Sigma_3[\mathcal{P}]$), conviene considerarlo como **conjunto** debido a que representa la parte clásica y la idempotencia de las fórmulas que contiene no es importante. Con esta consideración, la implantación del paso de Progresión debe ser capaz de empatar adecuadamente la parte Ψ_0 de la cabeza de un método (si es el caso) con los átomos del tanque.

5.3.2 Preprocesamiento

La fase de preprocesamiento es implantada en las siguientes cláusulas : (módulo `preproce.pl`)

```
/*************/
/* DUALIDAD  */
/*************/

dual((~P),       P1)           :- !, negin(P,P1).
dual(why(P),   bang(P1)) :- !, dual(P,P1).
dual(bang(P),   why(P1)) :- !, dual(P,P1).
dual((P * Q),    (P1 # Q1))    :- !, dual(P,P1), dual(Q,Q1).
dual((P & Q),    (P1 + Q1))    :- !, dual(P,P1), dual(Q,Q1).
dual((P # Q),    (P1 * Q1))    :- !, dual(P,P1), dual(Q,Q1).
dual((P + Q),    (P1 & Q1))    :- !, dual(P,P1), dual(Q,Q1).
dual(P,          (~P)).  % atomic

/***********************************
 * REESCRIBIENDO LAS IMPLICACIONES  *
 ***********************************/
/*
  La implicaci'on lineal A -> B se rescribe como ~A # B [Girard87], [Girard95]
  y A <-> B se reescribe como A -> B & B -> A
*/

implout((P -> Q),   ~(P1) # (Q1)) :- !, implout(P,P1), implout(Q,Q1).
implout((P <- Q),   (P1) # ~(Q1)) :- !, implout(P,P1), implout(Q,Q1).
implout((P <-> Q),  ~(P1) # (Q1) & ~(Q1) # (P1)) :- !,
           implout(P,P1), implout(Q,Q1).
implout(why(P),   why(P1))    :- !, implout(P,P1).
implout(bang(P),  bang(P1))    :- !, implout(P,P1).
implout((P & Q),    (P1 & Q1))    :- !, implout(P,P1), implout(Q,Q1).
implout((P * Q),    (P1 * Q1))    :- !, implout(P,P1), implout(Q,Q1).
implout((P # Q),    (P1 # Q1))    :- !, implout(P,P1), implout(Q,Q1).
implout((P + Q),    (P1 + Q1))    :- !, implout(P,P1), implout(Q,Q1).
implout((~P),       (~P1))        :- !, implout(P,P1).
implout(P, P).

/***************************
 * MOVER LA NEGACION ADENTRO *
 ***************************/

negin((~P),       P1) :- !, dual(P,P1).
negin(why(P),   why(P1)) :- !, negin(P,P1).
negin(bang(P), bang(P1)) :- !, negin(P,P1).
negin((P & Q),    (P1 & Q1))    :- !, negin(P,P1), negin(Q,Q1).
negin((P * Q),    (P1 * Q1))    :- !, negin(P,P1), negin(Q,Q1).
```

```
negin((P # Q),     (P1 # Q1))     :- !, negin(P,P1), negin(Q,Q1).
negin((P + Q),     (P1 + Q1))     :- !, negin(P,P1), negin(Q,Q1).
negin(P,P).
```

El primer grupo de cláusulas (`dual`) obtienen la fórmula dual de la fórmula que reciben como primer argumento. `implout` realiza la re escritura de los operadores concernientes a la implicación lineal. `negin` incorpora el operador de dualidad exclusivamente ante fórmulas atómicas.

Las siguientes cláusulas realizan funciones auxilliares a ser utilizadas en varias fases: (módulo `auxiliar.pl`)

```
/*************************
 * PREDICADOS AUXILIARES *
 *************************/

% Forma negativa de definir un a'tomo positivo:
noPosA(_ * _).  noPosA(_ + _).  noPosA(_ # _). noPosA(_ & _). noPosA(why(_)).
noPosA(bang(_)). noPosA(~_).
posAtom(X):- not(noPosA(X)).

 nula([]).

 append([],X,X).
 append([X|Y], Z, [X|Z1]) :- append(Y, Z, Z1).

 /* gensym/2 crea un a'tomo nuevo a partir del nombre Root proporcionado
    terminando con un nu'mero u'nico. Es utilizado por el procedimiento
    de normalizaci\'on (cl\'ausula natom/2). */

gensym(Root,Atom) :- get_num(Root,Num),
                     name(Root,Name1),
                     integer_name(Num,Name2),
                     append(Name1,Name2,Name),
                     name(Atom,Name).

get_num(Root,Num) :-  retract(current_num(Root,Num1)), !, /* ya estaba */
                      Num is Num1+1,
                      asserta(current_num(Root,Num)).
get_num(Root,1)   :-  asserta(current_num(Root,1)).       /* 1a vez*/

 /* integer_name/2 convierte a un entero a su lista de caracteres
    correspondiente*/

integer_name(Int,List)           :- integer_name(Int,[],List).
integer_name(I,Sofar,[C|Sofar]) :- I<10, !, C is I+48.
integer_name(I,Sofar,List)       :- Tophalf is I//10,     %In C&M I/10
```

```
                                  Bothalf is I mod 10,
                                  C is Bothalf+48,
                                  integer_name(Tophalf,[C|Sofar],List).


/*******************************
 * Operaciones entre conjuntos: *
   (m'odulo auxiliar.pl)
 *******************************/

 pert(X, [X|Y]).
 pert(X, [Y|Z]):- pert(X,Z).

 union([], X, X).
 union([X|Y], X1, Z) :- pert(X,X1), union(Y,X1,Z).
 union([X|Y], X1, Z) :- union(Y,X1,Z1), append([X], Z1, Z).

 inters([],X,[]).
 inters([X|Y], X1, Z):- pert(X,X1), inters(Y,X1,Z1), append([X], Z1, Z).
 inters([X|Y], X1, Z):- inters(Y,X1,Z1).

 subconj([],_).
 subconj([X|Y], X2) :- pert(X,X2), subconj(Y,X2).

 difer([],X,[]).
 difer([X|Y], X1, Z):- pert(X,X1), difer(Y,X1,Z).
 difer([X|Y], X1, Z):- difer(Y,X1,Z), append([X], Z1, Z).

 /**********************************
 * Operaciones entre multiconjuntos: *
 **********************************/

 % diferencia:
 diffms([], X, []).
 diffms(X, [], X).
 diffms(X1, [X|Y], Z) :- sacarms(X, X1, Y1), diffms(Y1, Y, Z).

 % sacarms: elimina la primer ocurrencia del 1er par'ametro en el segundo
 sacarms(X, [], []).
 sacarms(X, [X|Y], Y) :-!.
 sacarms(X, [Y|T], [Y|R]) :- sacarms(X, T, R).

 % intersecci'on:
 interms([], X, []).
 interms([X|R], Y, [X|T]) :- pert(X,Y), sacarms(X,Y,V), interms(R, V, T), !.
 interms([X|R], Y, T) :- interms(R,Y,T).
```

5.3.3 Normalización

Las cláusulas correspondientes a las funciones propias de normalización son las siguientes y su lectura es directa (ver 4.2.2, página 44): (archivo principal `linlog.pl`)

```
/*********************************************************************
 *       TRANSFORMACION A LA SINTAXIS   LINLOG:                       *
 *********************************************************************
  El algoritmo de normalizaci'on a la forma LINLOG  [J.M.Andreoli 92]:

  Se obtiene a partir de una fbf en L'ogica Lineal una consulta Linlog
  <P;G> donde P es una cl'ausula de nombre prog, que contiene un
  conjunto de m'etodos y G es una meta elemental.

  Supuestos:
  1) La fbf de entrada se supone "preprocesada", de tal manera que las
  implicaciones (<-, ->) han sido reescritas (A -> B por ~A#B) (implout);
  a su  vez, la dualidad se restringe a f'ormulas 'atomicas exclusivamente
  (negin).

  2) El programa P denotado por la cl'ausula prog(P) ser'a utilizado por
  el sistema de inferencia Linlog.

 *********************************************************************/

normalize(F, G ) :-
   asserta( prog([])),  % Programa actual P = {}
   ngoal(F, G).         % obtener el componente meta elemental G

/*************************************
 * NGOAL : regresa una meta elemental *
 *************************************/

ngoal(F1 # F2, NF1 # NF2) :- !,
   ngoal(F1, NF1), ngoal(F2, NF2).

ngoal(F1 & F2, NF1 & NF2) :- !,
   ngoal(F1, NF1), ngoal(F2, NF2).

ngoal(aTrue, aTrue) :- !. % Top
ngoal(mFalse, mFalse) :- !.%bottom
ngoal(why(F), why(Nf)) :- !, natom(F, Nf).
ngoal(F, Af) :- !, natom(F, Af).
```

```
/************************************
 * NATOM : regresa un atomo positivo *
 (tiene efectos laterales: actualiza el conjunto de m'etodos P)
 ************************************/

natom(X,X) :- posAtom(X), !.
natom(F,U) :-
   gensym(u,U),       % constante Skolem a nivel F'ormulas
   nmeth( ~U * F, Nmethf),
   barras(Nmethf, BNmethf),
   retract(prog(P)),
   union(P, BNmethf, NewP),
   asserta(prog(NewP)).

/****************************************
 * NMETH : regresa un conjunto de quasi-metodos
 ****************************************/
% -la f'ormula s'olo tiene 'atomos negativos: se aplic'o negin

 nmeth(~X, [quasi([X],[])] ) :- !.
 nmeth(F1 * F2, AsF) :- !,
   nmeth(F1, NF1), nmeth(F2, NF2),
   aster(NF1,NF2, AsF).
 nmeth(F1 + F2, AsF) :- !, nmeth(F1, NF1), nmeth(F2, NF2),
   union(NF1, NF2, AsF).
 nmeth(mTrue, [quasi([],[])] ) :- !. % One
 nmeth(aFalse, []) :- !. % Zero
 nmeth(bang(F), [quasi([], [bang(NF)])] ) :- ngoal(F,NF), !.
 nmeth(F, [quasi([], [Ngf])]) :- ngoal(F,Ngf), !.
```

Nótese el uso de cut (!) para evitar retroceso (es decir, la forma normal obtenida es única).

Nótese también que nmeth regresa el dual del conjunto de quasi métodos, de tal manera que lo que se obtiene al final es el dual del programa P y de esa manera se utiliza directamente en el sistema de inferencia. A su vez, es importante hacer notar que los operadores síncronos explícitos en las metas son ! y $\otimes$: $\oplus$ está implícito en la coma, separando a cada método del programa. De esa manera, se reduce el no determinismo a la etapa de selección de métodos en el procedimiento de búsqueda.

Las cláusulas auxiliares son:

```
 /*****************************************************
 * Operaciones auxiliares al proceso de normalizaci'on: *
 *****************************************************/

 % Fusiona dos quasi-m'etodos en un conjunto que contiene un quasi-m'etodo:
```

```
 fusiona(quasi(Phi1, Y1), quasi(Phi2, Y2), [quasi(Phi, Y)]) :-
    append(Phi1, Phi2, Phi),
    append(Y1, Y2, Y).

 % Fusiona al quasi m'etodo A con todos los quasi metodos del conjunto
 % [H|T]:
 fusiqc(A, [H|T], Z) :-
   fusiona(A,H,X),
   fusiqc(A,T,Y),
   union(X,Y,Z).
 fusiqc(A,[],[]).

/* Operador _*_ sobre conjuntos de quasi me'todos:
   para cada quasi m'etodo del primer conjunto, fusionarlo con cada
   quasi m'etodo del segundo conjunto
*/
 aster([H|T], X, Qm) :-
    fusiqc(H, X, Qm1),
    aster(T, X, Qm2),
    union(Qm1, Qm2, Qm).
 aster([], X, []).

 /* Operador ||_|| sobre un conjunto de quasi me'todos:
   Transforma a un conjunto de quasi-metodos en un conjunto de metodos:
       method(Cabeza) <- Cuerpo             % U
 */
 barras([quasi(H, T)|R], [(method(H) <- Cuerpo)|S]) :-
        embody(T, Cuerpo),
        barras(R, S).
 barras([], []) :- !.

 % Construye la meta Cuerpo:
 embody([], aTrue).            % vac'io: one
 embody([X], X).               % una meta 'unica
 embody([X|T], X * Y) :- embody(T, Y).  % Times expl'icito
```

5.3.4 Búsqueda de Pruebas

Para la implantación del sistema de inferencia, son importantes las siguientes consideraciones:

- **Contextos de Entrada y Salida:** el procedimiento de búsqueda puede verse como la *manipulación de recursos*: producción y consumo. Para ello, es útil diferenciar en cada secuente triádico los contextos de entrada y salida:
  1. *Contexto de Entrada* : fórmulas a ser consumidas, formado por

- *Theta*: (tanque de recursos de uso no limitado) átomos prefijados con el exponencial ? (why).
- *Gamma*: acumulador de átomos positivos.

Este contexto permite la activación de métodos en el paso de progresión.

El tercer componente depende si el proceso de prueba está en una etapa:

- *asíncrona*: (pila) [F—L], F es la meta elemental actual (tope de la pila) cuyas subfórmulas serán descompuestas y empiladas;
- *síncrona*: F es la meta actual (activa) en la cual la búsqueda se enfoca.

2. *Contexto de Salida (Out)*: fórmulas que quedan una vez consumidas (por los axiomas $\top$, 1 al alcanzar una prueba exitosa parcial para la rama actual) las fórmulas del Contexto de Entrada (más específicamente, del acumulador Gamma). Si la prueba global es exitosa, Out debe ser una lista vacía. (Precisamente en las cláusulas correspondientes a los axiomas, se aplica la técnica de cerrar el circuito In - Out).

El prólogo del procedimiento de búsqueda es el siguiente:

```
/*********************************************************************
 *                     SISTEMA TRIADICO LINLOG                       *
                               (linlog.pl)
 *********************************************************************/

   Supuestos:
   1) La entrada para (provea/4) es una meta elemental Linlog (denotada
      por una lista de f'ormulas donde ',' es una abreviatura para # 'par')

   2) La f'ormula de entrada para (proves/4) es una meta Linlog
   (f'ormula s'incrona enfocada)

   3) Los m'etodos del programa P se suponen en el 'area de texto,
   (cl'asula prog(P)/1), en notaci'on de lista:
     P = [(method( Cabeza ) <- Cuerpo)|T]

   4) La selecci'on de un m'etodo (por parte de la regla de progresio'n)
    tiene el criterio siguiente:
     i] activar primero todos los m'etodos cuyas cabezas sean submulticonjuntos
del acumulador Gamma exclusivamente;
     ii] activar enseguida a me'todos cuyas cabezas sean submulticonjuntos de
Gamma U Theta, incluso exclusivamente en Theta.

   5) No se permite activar al mismo m'etodo 2 veces consecutivas, para
   ello se guarda el 'indice (cardinalidad) del 'ultimo m'etodo
   seleccionado en la cl'ausula (lastmeth/1)
*/
```

La etapa asíncrona es implantada en el siguiente conjunto de cláusulas (ver 4.2.1, página 42):

```
/*******************
* ETAPA ASINCRONA  *
********************/
/* El uso de cut poda las ramas del 'arbol de b'usqueda, pero es del tipo
   rojo: quitarlo implica cambiar la sem'antica operacional del c'odigo */

provea(Theta, Gamma, [(mTrue)|L], []).  % Axioma Top

provea(Theta, Gamma, [(mFalse)|L], Out) :- !,
   provea(Theta, Gamma, L, Out).

provea(Theta, Gamma, [why(F)|L], Out) :- !, % alimenta al tanque
   provea([F|Theta], Gamma, L, Out).

provea(Theta, Gamma, [(F # G)|L], Out) :- !, % Par
   provea(Theta, Gamma, [F,G|L], Out).

provea(Theta, Gamma, [(F & G)|L], Out) :-  % With (Exchange expl'icita)
   provea(Theta, Gamma, [F|L], O1),
   provea(Theta, Gamma, [G|L], O2),
   append(O1, O2, Out)
provea(Theta, Gamma, [(F & G)|L], Out) :-
   provea(Theta, Gamma, [G|L], O1),
   provea(Theta, Gamma, [F|L], O2),
   append(O1, O2, Out).

provea(Theta, Gamma, [F|L], Out) :-    % F no as'{\i}ncrona :
   !,
   provea(Theta, [F|Gamma], L, Out).    % incorporarla en Gamma

provea(_, [], [], []). % contracci'on impl'icita: prueba exitosa

/*************************************************
 * Paso de PROGRESION : activaci'on de m'etodos   *
 *************************************************/
provea(Theta, Gamma, [], Out) :-    % Pila vac'{\i}a : Progresi'on
   append(Theta, Gamma, Contexto),    % contexto completo (Theta U Gamma)
   prog(P),        % conjunto de me'todos (programa actual)
   selectm(Contexto, P, Candidatos, 1), !,% se obtienen los candidatos
   not (nula(Candidatos)),  % no candidatos significa fallo
   ordcan(Candidatos, OrCand),
   pert(X, OrCand),             % selecciona a un m'etodo
   not (lastmeth(X)),      % que no haya sido previamente seleccionado
   updatem(X),             % establece X como 'lastmeth(X)'
   getmeth(P,X,Cabeza, Cuerpo),   % obtiene al metodo indexado por X en P
```

```
   diffms(Gamma, Cabeza, NewGamma), % actualiza al sub contexto lineal
   %  activa al m'etodo:
   proves(Theta, NewGamma, Cuerpo, Out).   % comienza secci'on cri'tica
                                     % de Enfocamiento sobre la meta Cuerpo
```

La lectura de las cláusulas correspondientes a cada operador asíncrono es directa (ver 4.2.1, pág 42).

Notablemente, el paso de progresión es codificado en la última cláusula. Primero se obtiene el conjunto de métodos que pueden ser activados por el contexto actual haciedo referencia a ellos solamente por su cardinalidad, el cual constituye un índice en lugar de usar toda una lista para incorporarlos (con el fin de ahorrar espacio, es más cómodo usar una lista de índices que una lista de métodos). En la selección de los métodos candidatos, se toma en cuenta el 'origen' de su activación; es decir, en qué caso de los siguientes, los átomos de su cabeza están contenidos en el Contexto:

1. únicamente en el acumulador Γ,
2. como submulticonjunto de $\Gamma \cup \Theta$
3. exclusivamente en Θ

Siguiendo la anterior clasificación se procede a marcar a los métodos que son activables por los 2 últimos casos con el fin de retardar su aplicación lo más posible, activando primero a los métodos del primer caso. Tal lista de candidatos no debe ser vacía (lo cual indicaría que no es demostrable la fórmula original). Es entonces seleccionado un método de esa lista, (que no haya sido el último seleccionado, con el fin de evitar posibles bucles[2]). Posteriormente es actualizado el subcontexto (lineal) Γ y se procede a activar al método, comenzando la sección crítica de Enfocamiento sobre la meta Cuerpo.

Las cláusulas auxiliares a este paso de progresión son:

```
/* ------ Auxiliares al paso de Progresi'on: -------*/

/* Selecciona del programa P a los me'todos que pueden ser activados por
   Contexto actual. Se sirve u'nicamente del i'ndice que indica su
   cardinalidad (u'nica) dentro de P
*/
signated([], Indx, Indx):- !.
signated(_X, Indx, Y) :-  Y is -Indx.

selectm(Theta, Contexto, [(method( Cabeza ) <- _)|T], [Y|R], Indx) :-
    subconj(Cabeza, Contexto), % condici'on necesaria
    interms(Cabeza, Theta, Void), % verifica su "origen"
    signated(Void, Indx, Y), %incorpora el \'indice seg\'n el caso
    NIndx is Indx + 1,
    selectm(Theta, Contexto, T, R, NIndx).
selectm(Theta, Contexto, [H|T], R, Indx) :- % no es aplicable
    NIndx is Indx + 1,
```

[2] Nótese, sin embargo, que tal condición es frágil, pues no evita bucles alternados: que dos métodos sean alternadamente seleccionados indefinidamente.

```
    selectm(Theta, Contexto, T, R, NIndx).
selectm(_,_, [], [], _) :- !.  % caso trivial: ya no hay ma's me'todos

/* Establece al me'todo con i'ndice X como u'ltimo seleccionado */
updatem(X) :- retract(lastmeth(_)), asserta(lastmeth(X)), !.

/* Obtiene al me'todo con cardinalidad N del programa P */
getmeth([(method( Cabeza ) <- Cuerpo)|_T], 1, Cabeza, Cuerpo) :- !.
getmeth([H|T], N, C,B) :- N1 is N - 1, getmeth(T,N1,C,B).

/* Ordena la lista de Candidatos: primero los que son originados por a'tomos
   exclusivamente en Gamma (lineales) y despue's los modalizados (en Gamma y
   Theta o' solo en Theta).
*/
separa([H|T], R, [Hn|R1]) :- % Candidatos, no-modalizados , modalizados
  H < 0, !, Hn is -H,
  separa(T, R, R1).    % modalizados -> segunda lista
separa([H|T], [H|R], R1) :-
  separa(T, R1, R).    % no-modalizados -> primer lista
separa([], [], []) :- !.

ordcan(A,Z) :- separa(A, X,Y), append(X,Y,Z), !.
```

A su vez, las cláusulas para la etapa síncrona son:

```
/*******************
*  ETAPA SINCRONA  *
*******************/

proves(Theta, Gamma, aTrue, Gamma):- !. % Axioma one (1)
  % regresa contexto actual (residual) cerrando el circuito In - Out

proves(Theta, Gamma, bang(F), Out) :- !,
   provea(Theta, Gamma, [F], Out).    % termina regi'on de Enfocamiento

proves(Theta, Gamma, (F * G), Out) :- % Regla Exchange expl'icita
  proves(Theta, Gamma, F, Delta),
  proves(Theta, Delta, G, Out).
proves(Theta, Gamma, (F * G), Out) :-
   proves(Theta, Gamma, G, Delta),
   proves(Theta, Delta, F, Out).

proves(Theta, Gamma, F, Out) :- !,      % Reacci'on Abajo:
   provea(Theta, Gamma, [F], Out).      % termina secci'on de
                                        % Enfocamiento
```

Nótese la utilidad de la estrategia de contexto de entrada y salida en la regla $\otimes$: para demostrar la primer subfórmula, se le proporciona todo el contexto para que una vez consumidas las fórmulas que le permitan una prueba exitosa, regrese las residuales para ser aprovechadas por la demostración de la otra subfórmula; evitando con ello elegir de antemano la partición al contexto.

El punto de entrada para el prototipo es el siguiente:

```
/*********************
 * Punto de Entrada :*
 *********************/
prove(X) :-
         implout(X,X1),    % preprocesamiento
   negin(X1,X2),
         normalize(X2, G),     % obtiene la forma normal LINLOG<P,G>
         asserta(lastmeth(0)),    % 'ultimo m'etodo seleccionado = ninguno
         provea([], [], [G], R),  % comienza la b'usqueda para G
         nula(R).                 % el contexto de salida debe R ser vac'io.
```

5.3.5 Comentarios adicionales

Para hacer más amigable el uso del prototipo, se implantaron un conjunto de cláusulas para construír el árbol de prueba en código LaTeX; para ello, un parámetro acumulante para las cláusulas **provea** y **proves** fue incorporado, el cual al final de la búsqueda contiene el árbol (conformado por las reglas utilizadas y el estado del cual parte). Tal árbol es estonces recorrido por la cláusula **latextree** (en el módulo **latexpt1.pl**) la cual despliega (en la salida estándar) el árbol LaTeX correspondiente[3].

[3] Debido a que LaTeX impone un cota en el uso recursivo de las macros utilizadas, el árbol es construído por bloques, donde cada bloque comienza por una *etiqueta* de rama y a continuación sigue una serie de nodos del árbol, con una altura de 5 máximo permitida a cada rama.

Capítulo 6

Pruebas y Resultados

En éste capítulo se presentan 3 ejemplos representativos para el uso del prototipo implantado.

6.1 Demostración automática de Fórmulas

A continuación, la prueba de una fórmula proposicional en Lógica Lineal.

$$\vdash (!a^{\perp} \otimes !b^{\perp}) \& (!(a^{\perp} \& b^{\perp}) \parr ?a \parr ?b)$$

Notablemente, al inicio son desplegados los componentes de la consulta LINLOG: los métodos que forman al programa, y la meta elemental. Se muestra la utilidad en el despliegue del árbol de búsqueda en macros LaTeX.

```
yes
?- formula(8,F),prove(F).
F = why(a)#why(b)<->why(a+b)

method([u2,a]) <- mTrue
method([u3,b]) <- mTrue
method([u1]) <- bang(u2)*bang(u3)
method([u4]) <- a
method([u4]) <- b
method([u6,a]) <- mTrue
method([u7,b]) <- mTrue
method([u5]) <- bang(u6&u7)

g = u1#why(u4)&u5#why(a)#why(b)
```

Rama root1 :

$$\dfrac{\dfrac{\dfrac{\dfrac{izq1}{\vdash: u1 \Uparrow ?(u4),}[?]}{\vdash:\Uparrow \underline{u1}, ?(u4)}[R\Uparrow]}{\vdash:\Uparrow \underline{(u1 \parr ?(u4))},}[\parr] \qquad \dfrac{\dfrac{\dfrac{der2}{\vdash: u5 \Uparrow \underline{(?(a)\parr ?(b))},}[\parr]}{\vdash:\Uparrow \underline{u5}, (?(a)\parr ?(b))}[R\Uparrow]}{\vdash:\Uparrow \underline{(u5\parr (?(a)\parr ?(b)))},}[\parr]}{\vdash:\Uparrow \underline{((u1\parr ?(u4))\&(u5\parr (?(a)\parr ?(b))))},}[\&]$$

Rama izq1 :

$$\dfrac{\dfrac{\dfrac{\dfrac{izq2}{\vdash u4 :\Uparrow \underline{u2},}[R\Uparrow]}{\vdash u4 :\Downarrow \underline{!(u2)},}[!] \qquad \dfrac{\dfrac{der1}{\vdash u4 :\Uparrow \underline{u3},}[R\Uparrow]}{\vdash u4 :\Downarrow \underline{!(u3)},}[!]}{\vdash u4 :\Downarrow (!(u2)\otimes !(u3))}[\otimes]}{\vdash u4 : u1 \Uparrow}[\text{⟜}]$$

Rama izq2 :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash u4 :\Downarrow 1}[1]}{\vdash u4 : a, u2 \Uparrow}[\text{⟜}]}{\vdash u4 : u2 \Uparrow \underline{a},}[R\Uparrow]}{\vdash u4 : u2 \Downarrow \underline{a},}[R\Downarrow]}{\vdash u4 : u2 \Uparrow}[\text{⟜}]$$

Rama der1 :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{der11}{\vdash u4 : a, u3 \Downarrow \underline{b},}[R\Downarrow]}{\vdash u4 : a, u3 \Uparrow}[\text{⟜}]}{\vdash u4 : u3 \Uparrow \underline{a},}[R\Uparrow]}{\vdash u4 : u3 \Downarrow \underline{a},}[R\Downarrow]}{\vdash u4 : u3 \Uparrow}[\text{⟜}]$$

Rama der11 :

$$\dfrac{\dfrac{\dfrac{}{\vdash u4 :\Downarrow 1}[1]}{\vdash u4 : b, a, u3 \Uparrow}[\text{⟜}]}{\vdash u4 : a, u3 \Uparrow \underline{b},}[R\Uparrow]$$

Rama der2 :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{izq3 \qquad der3}{\vdash b, a :\Uparrow \underline{(u6\&u7)},}[\&]}{\vdash b, a :\Downarrow \underline{!((u6\&u7))},}[!]}{\vdash b, a : u5 \Uparrow}[\text{⟜}]}{\vdash a : u5 \Uparrow \underline{?(b)},}[?]}{\vdash: u5 \Uparrow \underline{?(a)}, ?(b)}[?]$$

Rama izq3 :

$$\dfrac{\dfrac{\overline{\vdash b, a :\Downarrow 1}[1]}{\vdash b, a : u6 \Uparrow}[\circ\!-]}{\vdash b, a :\Uparrow \underline{u6},}[R \Uparrow]$$

Rama der3 :

$$\dfrac{\dfrac{\overline{\vdash b, a :\Downarrow 1}[1]}{\vdash b, a : u7 \Uparrow}[\circ\!-]}{\vdash b, a :\Uparrow \underline{u7},}[R \Uparrow]$$

```
More (y/n)? n

yes
?- nolog.
```

6.2 Interacción entre procesos mediante comunicación asíncrona: *ACL*

ACL [KY94] representa un enfoque para especificar la interacción entre procesos. Su sintaxis se define como sigue: dado un conjunto de 'átomos proceso' A_p y de 'átomos mensaje' A_m, las fórmulas válidas son:

C	$::=$	$A_p \circ\!- G$	Cláusula
G	$::=$	$\bot \mid \top \mid A_m \mid ?A_m \mid A_p \mid G \parr G \mid R$	Meta
R	$::=$	$M \otimes G \mid R \oplus R$	Recepción de mensajes
M	$::=$	$A_m^{\bot} \mid M \otimes M$	multiconjunto de mensajes a recibir

Una cláusula indica qué comportamiento puede presentar un proceso identificado por el átomo que constituye la cabeza (lo cual es un caso particular a la sintaxis más general de LINLOG). Tal comportamiento es expresado por las fórmulas Meta. Dada una configuración inicial (multiconjunto) de átomos procesos y un programa $?\mathcal{P}$ (conjunto de cláusulas), el sistema construye una prueba como se ha discutido.

Las reglas de prueba tiene un significado especial, y están divididas en 4 grupos:

1. **Desempeño en Paralelo:**

   - *Composición Concurrente:*

     $$\frac{\vdash ?\mathcal{P}, \Gamma, G_1, G_2}{\vdash ?\mathcal{P}, \Gamma, G_1 \parr G_2}\parr$$

     Esta regla indica que los procesos G_1 y G_2 comienzan a correr de manera independiente (en paralelo).

   - *Bifurcación:*

     $$\frac{\vdash ?\mathcal{P}, \Gamma, G_1 \qquad \vdash ?\mathcal{P}, \Gamma, G_2}{\vdash ?\mathcal{P}, \Gamma, G_1 \& G_2}\& \ (fork)$$

     Esta regla indica la creación de un proceso hijo (al estilo del `fork` de Unix) con el mismo contexto y que se ejecutará de manera independiente al proceso padre.

2. **Comunicación:**

   - *Envío de Mensajes:*

$$\frac{\vdash ?\mathcal{P}, \Gamma, A_m, B}{\vdash ?\mathcal{P}, \Gamma, A_m \parr B} C_1$$

   Esta regla indica el envío (asíncrono) de un mensaje A_m y mutar el comportamiento como indique B. El mensaje queda disponible en la configuración en espera de ser consumido.

   - *Recepción de Mensajes normales:*

$$\frac{\dfrac{}{(\vdash A_{m_i}, A_{m_i}^{\perp})} Id \qquad \vdash ?\mathcal{P}, \Gamma, A_j}{\vdash ?\mathcal{P}, \oplus_{i=1,n}(A_{m_i}^{\perp} \otimes A_i), A_{m_j}} C_2$$

$$1 \leq j \leq n, \oplus_{i=1,n}(Fi) =_{def} F_1 \oplus F_2 \oplus \ldots \oplus F_n.$$

   Esta regla indica el consumo de un multiconjunto de mensajes, expresado mediante la *dualidad*: consumir un mensaje significa encontrar su átomo dual. Así, $(A_{m_i}^{\perp} \otimes A_i)$ se lee: "leer el mensaje A_{m_i} y al mismo tiempo mutar al comportamiento A_i". Esta regla representa un refinamiento a la regla original, de tal manera que se elimina el nodeterminismo para la partición del contexto: una vez elegida la meta (denotado implícitamente por la regla $\oplus$) cuyo mensaje está disponible para ser consumido, se consume (lo cual denota el axioma Identidad).

   - *Recepción de Mensajes modalizados:*

$$\frac{\dfrac{}{(\vdash A_{m_i}, A_{m_i}^{\perp})} Id \qquad \vdash ?\mathcal{P}, \Gamma, A_j, ?A_{m_j}}{\vdash ?\mathcal{P}, \oplus_{i=1,n}(A_{m_i}^{\perp} \otimes A_i), ?A_{m_j}} C_3$$

   De manera correspondiente, los mensajes modalizados siempre están disponibles para ser utilizados, de tal forma que pueden utilizarse para compartir información (al estilo *broadcasting*).

3. **Finalización:**

   - *Término del programa:*

$$\frac{}{\vdash ?\mathcal{P}, \Gamma, \top} \top$$

   La regla indica la finalización de la ejecución en el contexto actual, lo cual puede interpretarse como término exitoso del programa.

   - *Término del proceso:*

$$\frac{\vdash ?\mathcal{P}, A_p}{\vdash ?\mathcal{P}, \perp, A_p} \perp$$

   La regla indica que el proceso desaparezca.

4. *Activación de cláusulas (métodos):*

$$\frac{\vdash ?\mathcal{P}, \Gamma, G}{\vdash ?\mathcal{P}, \Gamma, A_p} [\circ\!-]$$

Correspondientemente, la regla denota la activación del método $A_p \circ\!- G \in \mathcal{P}$

6.2.1 ACL en LINLOG

Fácilmente podemos codificar la sintaxis de ACL en LINLOG: las cláusulas (resp. métodos) que expresan el comportamiento de los procesos tienen en su cabeza un sólo átomo proceso. La sintaxis de metas también sigue la definición de metas LINLOG: la recepción de mensajes (R) corresponde a parte síncrona de las metas no elementales LINLOG. Adicionalmente, se restringe el uso de los exponenciales (?) únicamente a átomos mensaje; de igual forma, el conectivo $\otimes$es utilizado para la recepción de mensajes (evitando el nodeterminismo en la partición del contexto que se tiene en general) exclusivamente. $\oplus$sigue teniendo su significado original, restringido (al igual que $\otimes$) ante la disponibilidad de algún mensaje (de ahí que consistentemente sean *síncronos*). La recepción y consumo de mensajes (determinado por el uso de el axioma de identidad) se expresa en LINLOG mediante el renombrado del átomo dual correspondiente ($A_{m_i}^{\perp}$) por un átomo positivo nuevo (u_{m_i}) y la definición de un método que conduce a la constante 1: (que representa el consumo del mensaje).

$$\begin{array}{cc} ACL & \textsc{Linlog} \\ \dfrac{\dfrac{}{(\vdash A_{m_i}, A_{m_i}^{\perp})}\mathsf{Id} \quad \vdash ?\mathcal{P}, \Gamma, A_j}{\vdash ?\mathcal{P}, \oplus_{i=1,n}(A_{m_i}^{\perp} \otimes A_i), A_{m_j}} C_2 & \dfrac{\dfrac{\dfrac{}{\vdash \mathcal{P}' : \Downarrow 1} 1}{\vdash \mathcal{P}' : \Downarrow u_{m_i}, A_{m_i}} [\circ\!-] \quad \dfrac{\vdots}{\vdash \mathcal{P}' : \Gamma, A_j}}{\vdash \mathcal{P}' : \Gamma, A_{m_i} \Downarrow (u_{m_i} \otimes A_j)} [\otimes] \\ & u_{m_i} ⅋ A_{m_i} \circ\!-1 \in \mathcal{P}' \end{array}$$

A continuación se muestra la codificación correspondiente para el problema de los Filósofos comelones de Djistra.

$$\begin{array}{c} \text{Filoso} \circ\!-\text{Piensa.} \\ \text{Piensa} \circ\!-\text{Hambre.} \\ \text{Hambre} \circ\!- tenedor_i^{\perp} \otimes tenedor_j^{\perp} \otimes \text{Come.} \\ \text{Come} \circ\!- tenedor_i ⅋ tenedor_j ⅋ \text{Filoso.} \end{array}$$

A su vez, se muestra cómo se podrían codificar la interacción entre un proceso *Cliente*, un proceso *Servidor*. Se considera que el proceso *Servidor* crea dinámicamente a un proceso *hijo* para atender las solicitudes.

$$\begin{array}{c} \text{Cliente} \circ\!- mSolicitud ⅋ rRecibe. \\ \text{rRecibe} \circ\!- mResp^{\perp} \otimes \top. \\ \text{pServidor} \circ\!- rSolicitud. \\ \text{rSolicitud} \circ\!- mSolicitud^{\perp} \otimes (pHijo \& pServidor). \\ \text{pHijo} \circ\!- mResp ⅋ \perp. \end{array}$$

```
/* Interacci'on Cliente-Servidor en ACL */
pCliente <- mSolicitud # rRecibe.
rRecibe <- poolm([mResp], aTrue).
pServidor <- rSolicitud.
rSolicitud <- poolm([mSolicitud], pHijo & pServidor).
pHijo <- mResp # mFalse.

yes
?- proveacl([],[],[],[],[pCliente # pServidor]).
```

```
  No more messages... halting.
More (y/n)? n

yes
?-
```

Rama root3 :

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{root31}{\vdash: pCliente;\,;\Uparrow \underline{rSolicitud},}\,[R\Uparrow]}{\vdash: pServidor, pCliente;\,;\Uparrow}\,[\multimapinv]process}{\vdash: pCliente;\,;\Uparrow \underline{pServidor},}\,[R\Uparrow]}{\vdash:;\,;\Uparrow \underline{pCliente}, pServidor}\,[R\Uparrow]}{\vdash:;\,;\Uparrow \underline{(pCliente \parr pServidor)},}\,[\parr]$$

Rama root31 :

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{root311}{\vdash:; mSolicitud; rRecibe, rSolicitud \Uparrow}\,[\multimapinv]receive_m sg}{\vdash:; mSolicitud; rSolicitud \Uparrow \underline{rRecibe},}\,[R\Uparrow]}{\vdash:;\,; rSolicitud \Uparrow \underline{mSolicitud}, rRecibe}\,[R\Uparrow]}{\vdash:;\,; rSolicitud \Uparrow \underline{(mSolicitud \parr rRecibe)},}\,[\parr]}{\vdash: pCliente;\,; rSolicitud \Uparrow}\,[\multimapinv]process$$

Rama root311 :

$$\cfrac{\cfrac{\cfrac{\cfrac{izq3}{\vdash:;\,; rRecibe \Uparrow \underline{(mResp \parr mFalse)},}\,[\parr]}{\vdash: pHijo;\,; rRecibe \Uparrow}\,[\multimapinv]process}{\vdash:;\,; rRecibe \Uparrow \underline{pHijo},}\,[R\Uparrow] \qquad \cfrac{\cfrac{\cfrac{der3}{\vdash:;\,; rRecibe \Uparrow \underline{rSolicitud},}\,[R\Uparrow]}{\vdash: pServidor;\,; rRecibe \Uparrow}\,[\multimapinv]process}{\vdash:;\,; rRecibe \Uparrow \underline{pServidor},}\,[R\Uparrow]}{\vdash:;\,; rRecibe \Uparrow \underline{(pHijo \& pServidor)},}\,[\&]-fork$$

Rama izq3 :

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\vdash:;\,;\Uparrow \top}\,[\top]-end\ prog}{\vdash:; mResp; rRecibe \Uparrow}\,[\multimapinv]receive_m sg}{\vdash:; mResp; rRecibe \Uparrow \underline{mFalse},}\,[\bot]}{\vdash:;\,; rRecibe \Uparrow \underline{mResp}, mFalse}\,[R\Uparrow]$$

Rama der3 :

$$\cfrac{}{\vdash:;\,; rSolicitud, rRecibe \Uparrow \underline{no\ more\ messages},}\,[?]$$

6.3 Herencia interconstruída en Objetos

El fragmento **LO** [AP91] considera a los objetos como *entidades estructuradas*, de tal forma que cada secuente se ve como una *organización* de objetos independientes.

Los objetos son fórmulas asíncronas exclusivamente constituídas por múltiples literales conectadas por $\parr$, cada literal identifica un componente del objeto (atributo), el cual es involucrado en el cambio de estado. Las transiciones son aplicables de forma concurrente a las subpartes de un objeto (como consecuencia, si varios métodos pueden ser aplicables se elegi alguno de manera no determinística), inspirado por el **principio de organización mediante herencia** (*Organizational Inheritance*): una organización puede resolver alguna tarea específica si alguna de sus suborganizaciones la puede resolver.

Los métodos pueden ser activados via propagación (progresión) por empate parcial (vista parcial del objeto al cual se aplica) si tales métodos pertenecen a alguna superclase del objeto; de otra forma, mediante propagación por empate total deben pertenecer a su clase más específica. De esa forma, LO ofrece un mecanismo de *herencia interconstruída*: los métodos no son vinculados a los objetos (clases) explícitamente por el programador, sino que un método es aplicable cada vez que su cabeza empata alguna parte del estado del objeto. El especializar a un objeto (incorporarle más atributos) no produce que alguno de los métodos de la superclase se tornen inaplicables[1]. De manera notable, gracias al principio de organización, se evita la proliferación de delegados burócratas: la distribución interna de las tareas es realizada precisamente por el principio de organización jerárquica.

Tales propiedades, junto con la *cooperación externa*, (es decir, que las distintas organizaciones de objetos cooperen entre sí) permiten caracterizar agradablemente a la **metáfora sociológica**: considerar a los *programas como sociedades*.

A continuación se muestra un pequeño ejemplo de codificación[2]:

Supóngase el elemento básico de objeto gráfico:

1. $\mathrm{punto} \parr x(X) \parr \mathrm{y(Y)} \parr move(X1, Y1) \circ\!\!- \mathrm{punto} \parr x(X + X1) \parr y(Y + Y1)$.

2. $\mathrm{punto} \parr color(C) \parr setcolor(C1) \circ\!\!- \mathrm{punto} \parr color(C1)$.

Se puede apreciar, que todos los métodos para *punto* (1) también son aplicables a la subclase de *punto coloreado*, los métodos para esta subclase especializada no tienen que mencionar explícitamente a los atributos de la clase base (en este caso, $x(X)$ y $y(Y)$), a menos que fuesen estrictamente necesarios.

[1] Lo cual es similar al mecanismo de delegación propuesto por Henry Baker con la singularidad que los delegados no son especificados explícitamente.

[2] Lamentablemente, es necesario utilizar predicados para un ejemplo a mejor detalle, por lo que se lamenta dejar al lector con la miel en los labios...

Capítulo 7

Trabajo Relacionado

Es de mencionar algunos trabajos al respecto en la literatura. Teniendo como fundamento a las Pruebas Uniformes (ver 3.4.2, pág.33), los lenguajes de programación lógica que han sido propuestos son los siguientes:

- **Forum** Sin duda, Forum es el lenguaje de especificación en este enfoque considerablemente más parecido a LINLOG. Propuesto por Dale Miller [HP96], sus metas fueron aplicar la definición de pruebas uniformes para Lógica Lineal y permitir incorporar tanto abstracciones (programación de alto orden, módulos, tipos de datos abstractos y encapsulación de estado) de los lenguajes λProlog y Lolli como las nociones de concurrencia de LO.

  Forum está basado en el fagmento clásico de conclusión múltiple. Este lenguaje tiene 2 distinciones importantes:

  - es completo para toda la Lógica Lineal, en el sentido que aquellas fórmulas que no sean parte de Forum pueden ser *mapeadas* a Forum mediante una traducción que preserva la probabilidad;
  - el lenguaje completo puede verse como un lenguaje de programación lógica abstracto.

  El diseño de Forum requirió extender la noción de Prueba Uniforme para el entorno de conclusión múltiple. Dale Miller tomó la opción natural, diciendo que una prueba de conclusión múltiple es uniforme si, cuando existe una meta no atómica en la parte derecha, la última regla actúa para descomponer tal meta.

  A diferencia de LINLOG, Forum utiliza algunos conectivos (que considera primitivos) para poder expresar a todos los demás: notablemente, la negación no es considerada como primitiva sino derivada, conduciendo a expresarla como: $A^{\perp} =_{def} A \multimap \perp$. Las reglas de inferencia en Forum son más numerosas (correspondiendo al sistema simétrico tradicional) y notablemente complicadas. El orden de las fórmulas programa (correspondiendo a los métodos LINLOG) son responsabilidad del programador, en el sentido que una presentación incorrecta podría conducir a bucles inútiles debido a la estrategia de búsqueda primero en profundidad.

- **Lolli** Obedeciendo a los problemas de representar la noción de **estado** de *objeto mutable* dentro de la programación lógica[1] y de la noción de **cláusula relativa** dentro del formalismo *Gramática de*

[1] Debido a que en la lógica intuicionista no hay límites en que tan a menudo una cláusula puede ser utilizada en un programa, la cláusula que representa el estado anterior sigue siendo patente durante la prueba de una meta subordinada, el estado de un objeto representado en este entorno se torna indeterminado.

Estructura de Frase Generalizada (GPSG) (consistente fundamentalmente en reglas para *parsing* cuyo uso es restringido a casos particulares); Joshua Hodas y Dale Miller propusieron *Lolli* [Hod92, HM94], el cual refina las lógicas tras Prolog y λProlog. Está basado en la Lógica Lineal Intuicionista y también divide a los contextos en 2 partes: el intuicionista (recursos ilimitados) y el lineal (recursos limitados).

- **Lygon** Lygon es un lenguaje de programación lógica basado en un fragmento de Lógica Lineal clásica de conclusión múltiple. Como en Forum, el fragmento lógico que subyace en este lenguaje fue escogido en base al requerimiento que las Pruebas Uniformes sean completas en ese fragmento. La ejecución de Lygon es en general más no determinística. Fue propuesto por James Harlan y David Pym [HW95], y está siendo desarrollado por Michael Winikoff.

- **LLP** LLP también está cimentado en la Lógica Lineal Intuicionista; es un superconjunto de Prolog y un subconjunto de Lolli, la principal diferencia es que LLP ofrece un compilador (en lugar de intérprete) basado en una extensión a la máquina abstracta de Warren (WAM) para programación Lógica Lineal. Está siendo desarrollado bajo la dirección de Naoyuki Tamura en la Universidad de Kobe [TK96b, TK96a].

Capítulo 8

Conclusión y Trabajo a Futuro

8.1 Conclusión

En esta tesis se ha implantado un prototipo buscador de pruebas en Lógica Lineal proposicional en base a las propiedades de su Teoría de Pruebas, cuyo análisis ha conducido al establecimiento de la forma normal LINLOG correspondiente que permite la búsqueda de una clase normal de pruebas conocidas como pruebas enfocadas. Este prototipo permite el estudio y experimentación de la expresividad de la Lógica Lineal (dentro del paradigma computacional de búsqueda de pruebas como cálculo) para ofrecer sustento formal al comportamiento dinámico de los sistemas interactivos, fundamentado en el cuidado de los recursos disponibles. Tres áreas para uso genéricas de aplicación del prototipo han sido presentadas: la demostración automática de fórmulas, la especificación de interacción entre procesos concurrentes mediante comunicación asíncrona (ACL) y soporte a la metáfora de objetos como organizaciones estructuradas (LO). En todas ellas la expresividad de la Lógica Lineal ha sido manifiesta.

La implantación de las cláusulas correspondientes en el prototipo ha sido directa a partir de la especificación del procedimiento de normalización y la búsqueda de pruebas. La estrategia del contexto de Entrada/Salida ha resultado ser muy eficaz y ha conducido el análisis del consumo de los recursos por parte de cada rama en el árbol de prueba. Notable excepción constituye la implantación del paso de progresión: el no determinismo intrínseco en la elección de un método que conduzca a una prueba exitosa (y por ende, a un consumo satisfactorio de los recursos) presenta un contínuo reto para su codificación eficiente, por lo que propuestas para *análisis estático* son necesarias.

Este trabajo es sólo un primer paso hacia el estudio de la Lógica Lineal (particularmente, mediante LINLOG) para propósitos de Especificación Formal, ofreciendo una base para posterior investigación, tanto en el aspecto pragmático (implantación eficiente de un lenguaje de especificación y de programación lógica basados en LINLOG) como en el área semántica: explorar la utilidad de la Lógica Lineal como meta-teoría lógica que derive en propuestas semánticas para los lenguajes de programación lógica y especificación formal a implantar.

8.2 Problemas Abiertos

Fundamentalmente, el ofrecer criterios para la selección de métodos en el paso de progresión mediante análisis estático (lo cual puede verse como una fase posterior de compilación), en donde deban tratarse el no determinismo ('no lo sé'), las decisiones delicadas al elegir métodos activables por el tanque de recursos no limitado, la detección de nuevos bucles en el árbol de prueba (nuevas clases de pruebas p-equivalentes como consecuencia a la elección de métodos) y el posible desperdicio de recursos como consecuencia a una 'mala' elección.

8.3 Trabajo a Futuro

Afortunadamente, queda mucho por realizar:

- Construcción inductiva de nuevos criterios de selección de métodos en el paso de progresión: análisis estático de los programas (consultas LINLOG, en general) que descubran bucles en el árbol de prueba; así como el establecimiento de restricciones en cuanto al ámbito de consumo en las ramas del árbol de prueba.
- Aprovechar las estrategias de las Redes de Prueba *(Proof Nets)* en la implantación del criterio de selección más efectivo, como también anotaciones respecto al significado de las interacciones entre las fórmulas durante el curso de una prueba.
- Detallar reglas consistentes y generales para la implantación de la búsqueda conducida por el árbol sintáctico correspondiente.
- Implantación de un lenguaje de programación lógica basado en LINLOG, que soporte tanto la especificación como desempeño eficiente (aprovechando para ello la concurrencia, paralelismo y distribución). A su vez, aspectos propios de investigación para ofrecer soporte a los requerimientos de *modularidad, escalabilidad* y *localidad* para la construcción de aplicaciones son necesarios.

Bibliografía

[Abr93] Samson Abramsky. Computational interpretations of linear logic. *Theorical Computer Science*, (111):3–57, 1993.

[Ale94] Vladimir Alexiev. Applications of linear logic to computation: An overview. Technical Report TR93-18, University of Alberta, Edmonton, Alberta T6G 2H1, September 1994.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[AP90] Jean-Marc Andreoli and Remo Pareschi. LO and behold! Concurrent structured processes. In *Proceedings of OOPSLA'90*, pages 44–56, Ottawa, Canada, October 1990. Published as ACM SIGPLAN Notices, vol.25, no.10.

[AP91] Jean-Marc Andreoli and Remo Pareschi. Communication as fair distribution of knowledge. In *Proceedings of OOPSLA'91*, pages 212–229, Phoenix, Arizona, October 1991. Published as ACM SIGPLAN Notices, vol.26, no.11.

[Chi95] Jawahar L. Chirimar. *Proof Theoretic Approach to Specification Languages*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1995.

[Di 92] Roberto Di Cosmo. Introduction to linear logic. Course Notes, University of Pisa, Spring 1992.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987. The seminal paper of Linear Logic.

[Gir89] Jean-Yves Girard. Towards a geometry of interaction. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, pages 69–108. American Mathematical Society, 1989. Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference, June 14–20, 1987, Boulder, Colorado; Contemporary Mathematics Volume 92.

[Gir95] Jean-Yves Girard. Linear logic: Its syntax and semantics. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 1–42. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.

[HM94] Joshua S. Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. Extended abstract in the Proceedings of the Sixth Annual Symposium on Logic in Computer Science, Amsterdam, July 15–18, 1991.

[Hod92] Joshua S. Hodas. Lolli: An extension of λprolog with linear context management. In D. Miller, editor, *Workshop on the λProlog Programming Language*, pages 159–168, Philadelphia, Pennsylvania, August 1992.

[HP96] Joshua S. Hodas and Jeffrey Polakow. Forum as a logic programming language: Preliminary results and observations. Technical report, Harvey Mudd College, 1996.

[HW95] James Harland and Michael Winikoff. Implementation and development issues for the linear logic programming language Lygon. In *Proceedings of the Eighteenth Australian Computer Science Conference*, pages 563–572, Adelaide, Australia, February 1995.

[KY94] Naoki Kobayashi and Akinori Yonezawa. Asynchronous communication model based on linear logic. *Formal Aspects of Computing*, 3:279–294, 1994.

[Lin92] Patrick Lincoln. Linear logic. *ACM SIGACT Notices*, 23(2):29–37, Spring 1992.

[LMSS92] Patrick Lincoln, John Mitchell, Andre Scedrov, and Natarajan Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56:239–311, April 1992. Also in the Proceedings of the 31th Annual Symposium on Foundations of Computer Science, St Louis, Missouri, October 1990, IEEE Computer Society Press. Also available as Technical Report SRI-CSL-90-08 from SRI International, Computer Science Laboratory.

[LS93] Patrick Lincoln and Natarajan Shankar. Proof search in first-order linear logic and other cut-free sequent calculi. Technical Report CSL-93-11, SRI International Computer Science Laboratory, 333 Ravenswood Ave. Menlo Park, CA, 94025, 1993.

[MFW92] José Meseguer, Kokichi Futasugi, and Timothy Winkler. Using rewriting logic to specify, program, integrate and reuse open concurrent systems of cooperating agents. Technical Report SRI-CSL-92-11, SRI International, Computer Science Laboratory, 333 Ravenswood Avenue, Menlo Park, CA, 1992.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall International (UK) Ltd., 66 Wood Lane End, Hemel Hempstead Hertfordshire, 1989.

[MMO92] José Meseguer and Narciso Martí-Oliet. Action and change in rewriting logic. Technical Report SRI-CSL-94-07, SRI International, Computer Science Laboratory, 333 Ravenswood Avenue, Menlo Park, CA, 1992.

[MMO93] José Meseguer and Narciso Martí-Oliet. Rewriting logic as a logic and semantic framework. Technical Report SRI-CSL-93-05, SRI International, Computer Science Laboratory, 333 Ravenswood Avenue, Menlo Park, CA, 1993.

[MNSP91] Dale Miller, Gopalan Nadathur, Andre Scedrov, and Frank Pfenning. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[Sce93] Andre Scedrov. A brief guide to linear logic. In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 377–394. World Scientific Publishing Company, 1993. Also in Bulletin of the European Association for Theoretical Computer Science, volume 41, pages 154–165.

[Sce94] Andre Scedrov. Linear logic and computation: A survey. In L. F. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Proceedings of the International Summer School of Marktoberdorf*, NATO Advanced Science Institutes, series F94, pages ??–?? Springer-Verlag, 1994.

[TK96a] Naoyuki Tamura and Yukio Kaneda. A wam model for a linear logic programming language. In *Proceedings of the 9th Exhibition and Symposium on Industrial Applications of Prolog*, pages 63–70, Oct. 1996. (in Japanese).

[TK96b] Naoyuki Tamura and Yuko Kaneda. Extension of WAM for a linear logic programming language. In T. Ida, A. Ohori, and M. Takeichi, editors, *Second Fuji International Workshop on Functional and Logic Programming*, pages 33–50. World Scientific, Nov. 1996.

[Weg95] Peter Wegner. Models and paradigms of interaction. Tutorial Notes CS-95-21, Brown University, Department of Computer Science, Providence, Rhode Island 02912, September 1995.

[YT89] Akinori Yonezawa and Mario Tokoro, editors. *Object-Oriented Concurrent Programming*. MIT Press, 1989.