TESIS DOCTORAL

NEGACION CONSTRUCTIVA PARA PROGRAMACION LOGICA-ECUACIONAL

Memoria presentada por María José Ramírez Quintana
para la obtención del grado de Doctor en Informática
por la Universidad Politécnica de Valencia

Valencia, Noviembre de 1993

NEGACION CONSTRUCTIVA PARA PROGRAMACION LOGICA-ECUACIONAL

Tesis Doctoral en Informática presentada por

María José Ramírez Quintana
Licenciada en Ciencias Físicas por la Universidad de Valencia

Directores:

Prof. Dr. Giorgio Levi	Prof. Dr. Isidro Ramos Salavert
Università di Pisa	U. Politécnica de Valencia

Tribunal de Lectura:

Presidente:	Prof. Dr. Fernando Orejas Valdés	U. Politécnica de Cataluña
Vocales:	Prof. Dr. Mario Rodriguez Artalejo	U. Complutense de Madrid
	Prof. Dr. Moreno Falaschi	Università di Padova
	Prof. Dr. Juan José Moreno Navarro	U. Politécnica de Madrid
Secretario:	Prof. Dra. María Alpuente Frasnedo	U. Politécnica de Valencia

a José Luis, Alicia y ...

Indice

Lista de Figuras

Resumen

Dentro del campo de los lenguajes ecuacionales y lógico-ecuacionales, una de las extensiones en la que últimamente se han centrado las investigaciones es la incorporación de la negación. Esta extensión no sólo incrementa la potencia expresiva de los lenguajes sino que es necesaria para poder expresar algunos problemas cuya formulación más natural hace uso de la negación, como es el caso de las especificaciones de algunos tipos de datos. El problema de la negación del predicado igualdad se ha abordado con técnicas y planteamientos muy distintos. Así, por ejemplo, algunas aproximaciones abordan el problema de resolver la desigualdad con respecto a la teoría ecuacional vacía. Otras consideran sistemas de ecuaciones y disecuaciones a resolver con respecto a teorías ecuacionales de Horn satisfaciendo diversas restricciones. Finalmente, otras propuestas (entre las que podemos situar la presentada en esta tesis) consideran teorías ecuacionales que incluyen negación, con respecto a las que se resuelven sistemas de ecuaciones o sistemas de ecuaciones y disecuaciones (como es nuestro caso) usando diversos mecanismos operacionales basados o en la reescritura de términos o en algún procedimiento de resolución de ecuaciones (tal y como "narrowing"). Quizás algunas de las causas que han contribuido a esta diversidad de planteamientos son la propia complejidad del predicado igualdad y la existencia de problemas abiertos dentro del campo de las especificaciones positivas.

El principal argumento discutido en esta tesis es que es posible resolver 'constructivamente' la negación del predicado igualdad. El término constructivo hace referencia a una de las aproximaciones propuestas para resolver la negación en programación lógica y que fue acuñada bajo el apelativo de 'Negación Constructiva'. Esta denominación alude directamente a la forma en que se tratan los literales negados: en esta aproximación, las respuestas a un literal negado son 'construidas' a partir de las respuestas de su versión positiva. Siguiendo esta idea, en la tesis se define una extensión del algoritmo de "narrowing" capaz de resolver conjunciones de ecuaciones y disecuaciones con

respecto a una teoría ecuacional de Horn que permite el uso de disecuaciones en las condiciones de las cláusulas. Uno de los atractivos de nuestra aproximación es que el mecanismo computacional subyacente es único ya que la resolución de una disecuación $t \neq s$ se obtiene como una adecuada transformación de una subderivación para su versión positiva $t = s$ que define la regla de negación constructiva ecuacional. Otra de sus características, en relación a otras propuestas, es la presencia en las respuestas computadas de variables cuantificadas universalmente, que surgen como resultado del proceso de transformación aplicado cuando se resuelven subobjetivos negados.

La organización de la tesis es la siguiente. En los capítulos uno y dos se estudia el problema de la negación en programación lógica, ecuacional y lógica-ecuacional, revisando las distintas aproximaciones formuladas y centrando el interés en los dos últimos paradigmas. En concreto, se analiza la problemática asociada a las teorías ecuacionales normales, es decir, teorías ecuacionales que pueden contener negación en las condiciones de las cláusulas. Los siguientes tres capítulos constituyen el núcleo de este trabajo. En el capítulo tercero se define formalmente una semántica operacional para el lenguaje. El algoritmo, basado en un procedimiento de "innermost narrowing" estándar, no sólo resuelve conjunciones de ecuaciones y disecuaciones sino que también las simplifica (mediante ciertas reglas que se proponen al final del capítulo). Describimos el procedimiento de "narrowing" aumentado con negación constructiva ecuacional como un cálculo jerárquico compuesto por tres relaciones. En el capítulo cuatro se define la semántica por teoría de modelos y por punto fijo. Para ello, se introduce la noción de teoría completada de forma que el significado de una teoría ecuacional normal se da en términos de los modelos de su compleción. Construimos un modelo asociado a la teoría ecuacional normal que es también modelo de la compleción. Para que esta propiedad se cumpla, la teoría normal debe satisfacer la restricción de ser estratificada, lo que garantiza que la compleción de la misma es consistente. En el quinto capítulo se estudian las propiedades de corrección y completitud del cálculo presentado en el capítulo tres que expresan la equivalencia entre las semánticas operacional y declarativa. Finalmente, se establecen algunas conclusiones y se presentan posibles líneas futuras de continuación de este trabajo. Asimismo, se incluye como un anexo un prototipo Prolog que implementa el método de negación constructiva ecuacional, y algunos experimentos realizados con el sistema.

Algunas notas de caracter bibliográfico: una versión preliminar de parte del material presentado en los capítulos dos y tres se presentó en [133, 135] y parte de los capítulos tres, cuatro y cinco en [134]. Distintos aspectos del prototipo del anexo se han presentado en [116, 117, 125] y también en [126, 127].

Quiero expresar mi más profundo agradecimiento a los profesores Giorgio Levi e Isidro Ramos por su continuo apoyo durante la elaboración de esta tesis. Agradezco a todos los miembros del Dipartimento di Informatica de la Università di Pisa la amable acogida que me dispensaron durante mi estancia en Pisa y muy en particular a Moreno Falaschi por su amistad, apoyo y colaboración en buena parte de este trabajo. Un agradecimiento también muy especial para María Alpuente por su incondicional y continuo apoyo, su amistad y su atenta revisión del trabajo que me ha permitido profundizar en muchas partes del mismo. Gracias también a Vicente Gisbert por sus interesantes sugerencias y comentarios. Un reconocimiento especial a Javier Piris y Armando Mora por su colaboración en las tareas de experimentación. Mi agradecimiento a Asunción Casanova, Javier Oliver, María José Ramis, Germán Vidal y al resto de miembros tanto del grupo de Programación Lógica e Ingeniería del Software como del Departamento de Sistemas Informáticos y Computación. Finalmente, mi más sincero agradecimiento a mi familia por su comprensión y ayuda sin las cuales no habría sido posible la realización de esta tesis.

Capítulo 1

Introducción

La teoría de los sistemas de reescritura de términos es la principal herramienta usada para el tratamiento operacional de la lógica ecuacional (incondicional [34, 70] y condicional [40, 41, 83, 84]). Sin embargo, la reescritura no es un método adecuado para resolver ecuaciones en una teoría ecuacional. Dado un conjunto de reglas, es decir, de ecuaciones dirigidas (incondicionales o condicionales) que definen una teoría ecuacional $\mathcal{E}$, decimos que una ecuación $s = t$ tiene una solución en $\mathcal{E}$ si para alguna instanciación de las variables en s y t la igualdad se cumple. Los métodos para resolver ecuaciones consisten en encontrar tales valores (sustituciones) para las variables. Uno de estos métodos es la paramodulación, usada, por ejemplo, en algunos demostradores de teoremas basados en resolución. Su principal inconveniente es que resulta altamente ineficiente. Para teorías ecuacionales que puedan ser presentadas como un sistema de reescritura confluente se han desarrollado otros métodos mejores. Quizás el más popular ha sido "narrowing" [48, 64, 71, 72]. Otra aproximación alternativa se basa en el uso de reglas de transformación, como es el caso del procedimiento "top-down" presentado por Martelli, Moiso y Rossi en [106], en el que se definen dos operaciones, la descomposición y la reestructuración, que reducen el espacio de búsqueda al no considerar ciertos pasos de "narrowing". Ambos procedimientos son completos en el sentido de que encuentran una solución si es que ésta existe. No obstante, cuando una ecuación es insatisfacible, estos procedimientos pueden no parar. La no terminación de estos métodos es debida a la propia semidecidibilidad del problema de la satisfacibilidad ecuacional.

La resolución de ecuaciones y la detección de ecuaciones insatisfacibles son fundamentales en algunas aplicaciones. Por ejemplo, recientemente se han propuesto diversos lenguajes de programación que usan ecuaciones condicionales como un medio para combinar las principales características de la programación lógica y funcional. Algunos de estos lenguajes son RITE [42], RAP [72], SLOG [53], QUTE [141], BABEL [118, 119] y el lenguaje en [136, 137]. En esta aproximación a la integración lógico-ecuacional, un programa es un conjunto de reglas de reescritura condicionales y un objetivo es una ecuación (o conjunto de ecuaciones) a ser resuelta(s). Por lo tanto, la resolución de ecuaciones es una operación básica en los intérpretes de tales lenguajes. Análogamente, los procedimientos para completar 'a la Knuth-Bendix' una teoría ecuacional condicional, como el presentado por Kaplan en [86], también hacen uso de métodos para resolver las ecuaciones que aparecen en las condiciones y, en algunos casos [86], de métodos para detectar insatisfacibilidad.

Así como la programación lógica se basa en la noción de relación, la programación funcional (o ecuacional) se basa en la noción de función. No obstante, la programación lógica y la funcional son muy similares. En ambos paradigmas de programación, secuencias de símbolos se sustituyen por otras hasta que se alcanza una forma normal o canónica. Este hecho viene marcado, en programación funcional, por una expresión que no contiene redexes (subtérminos reducibles), mientras que en programación lógica es la derivación de la cláusula vacía la que señala la terminación de la computación.

Las ecuaciones son un caso especial de fórmulas lógicas con la igualdad como único símbolo de predicado. El resultado en el que se basan las especificaciones algebraicas (o ecuacionales) con modelo inicial es que existe una única álgebra inicial en la clase de todas las álgebras que satisfacen un conjunto de ecuaciones. Por otra parte, es bien conocido que el subconjunto de la lógica de predicados de primer orden constituido por las cláusulas de Horn tiene la propiedad de intersección de modelos, la cual puede verse, con una traducción adecuada, como la propiedad de álgebra inicial en el paradigma ecuacional. Este hecho, junto con la introducción de las ecuaciones condicionales, considerada como la extensión de la lógica ecuacional en la que no es necesario restringirse a fórmulas atómicas, sugiere que las cláusulas de Horn son un formalismo adecuado para definir las especificaciones ecuacionales [154]. Las teorías ecuacionales de Horn constituyen la clase más amplia de teorías ecuacionales que admiten una semántica por modelo mínimo. Otra razón para estar interesados en las cláusulas de Horn es que, en estas cláusulas, podemos expresar las definiciones recursivas de funciones de forma más natural [95].

Las cláusulas de Horn ecuacionales han sido tratadas y estudiadas ampliamente por diversos autores. Son especialmente adecuadas como expresión de la parte ecuacional cuando se desea integrar, de manera uniforme, los paradigmas de programación lógica y funcional [12, 14] ya que, en este caso, ambos paradigmas tienen los mismos mecanismos operacionales básicos y una estructura similar para los modelos. El formalismo que subyace en esta aproximación es la lógica de las cláusulas de Horn con igualdad [63, 64, 77, 78]. Jaffar, Lassez y Maher [77, 78] mostraron que las principales propiedades semánticas de los programas lógicos también se cumplen para los programas lógico-ecuacionales. Así, ya que la parte ecuacional de estos programas es también un programa en cláusulas de Horn, la propiedad de intersección de modelos se cumple y, por lo tanto, la parte ecuacional genera una relación de congruencia más fina sobre el conjunto de términos básicos. Esto permite determinar las consecuencias lógicas del programa lógico-ecuacional sobre el cociente de la base de Herbrand módulo esta congruencia. Por otra parte, existe una caracterización por punto fijo de cada programa de tal forma que su modelo mínimo se construye, de manera "bottom-up", como el menor punto fijo de una función T, monótona y continua, asociada al programa [54, 64]. Desde un punto de vista operacional, el mecanismo computacional usado en muchos de estos lenguajes, como por ejemplo FUNLOG [148] y EQLOG [59], es la SLDE-resolución, una extensión de la regla de inferencia SLD-resolución que incorpora unificación módulo un conjunto $\mathcal{E}$ de ecuaciones ($\mathcal{E}$-unificación [55, 56]). Una aproximación alternativa es la seguida en la definición de los lenguajes EUROPA [5], LEAF [13] y K-LEAF [18, 57]. En estos lenguajes, las ecuaciones se resuelven usando resolución estándar en lugar de un procedimiento de resolución ecuacional. Para ello, el programa se somete a un preproceso de transformación ("flatening") que lo convierte en un conjunto de cláusulas de Horn planas, es decir, en cláusulas en las que la composición funcional se reemplaza por conjunciones lógicas de literales planos. Esta técnica fue introducida en programación lógica por Barbutti, Bellia, Levi y Martelli [13] y Tamaki [149], principalmente.

En realidad, el procedimiento de "narrowing" y la resolución son dos reglas de inferencia que se basan en el mismo mecanismo [17]: la unificación de una porción del objetivo (un subobjetivo en resolución, un subtérmino de uno de los miembros de una ecuación en "narrowing") y una porción de una regla (la cabeza de una cláusula en resolución, la parte izquierda de la cabeza de una regla de reescritura en "narrowing"), seguida de la aplicación del unificador al objetivo y a la regla para, finalmente, crear un nuevo objetivo poniendo juntas las partes sobrantes del objetivo y de la regla. La principal diferencia entre el "narrowing" y la resolución reside en que, en el proceso de unificación, el "narrowing" toma del objetivo cualquier subtérmino del literal seleccionado, mientras que la resolución toma todo el literal y, por lo tanto, todos los términos que aparecen como argumento. El "narrowing" puede simularse por resolución si tanto los objetivos como las reglas son aplanadas ya que, tras el aplanamiento, los términos que aparecían como argumentos de literales se transforman en literales a los que la resolución puede aplicar unificación. De hecho, SLD-resolución aplicada sobre un programa y objetivo aplanados es semánticamente equivalente a un "narrowing" refinado y más eficiente que un "narrowing" ordinario [17].

En los últimos años ha surgido una nueva clase de lenguajes de programación basados en la resolución de restricciones y en la programación lógica: los llamados lenguajes de programación lógica con restricciones (CLP) [73, 74]. CLP($\mathcal{X}$) es un esquema genérico que ve un lenguaje de programación lógica como un lenguaje de restricciones sobre un dominio de discurso. Las restricciones sobre la estructura $\mathcal{X}$ son relaciones entre elementos de $\mathcal{X}$. Empleando dominios computacionales diferentes se obtienen distintos lenguajes lógicos. Por ejemplo, los lenguajes de programación lógica convencionales (tales como Prolog) pueden verse como lenguajes de programación con restricciones cuyas restricciones son ecuaciones que se resuelven en el universo de Herbrand por medio de unificación sintáctica. En Prolog II [76], las restricciones son ecuaciones y disecuaciones que se resuelven en el álgebra de los árboles racionales. En Prolog III [25, 26], las restricciones son también ecuaciones y disecuaciones que se resuelven en el álgebra de términos o en dominios predefinidos, como los números racionales y los booleanos.

Una de las principales ventajas de este esquema es que cualquier lenguaje definido como una instancia del mismo hereda las propiedades semánticas fundamentales de las cláusulas definidas (existencia de modelo mínimo, caracterización por punto fijo del modelo mínimo y equivalencia entre las semánticas operacional y declarativa).

La semántica operacional del esquema se basa en la existencia de un procedimiento para decidir la satisfacibilidad de los sistemas de restricciones. La semántica declarativa de CLP($\mathcal{X}$) se basa en la definición de una estructura $\mathcal{R}$ axiomatizada por una teoría $\mathcal{J}$. Para garantizar la equivalencia entre semánticas, el esquema CLP impone algunas restricciones tanto a $\mathcal{R}$ como a $\mathcal{J}$. Así, $\mathcal{R}$ debe ser *compacta respecto a la solución*, es decir, cada elemento del dominio debe poder ser especificado por un número (posiblemente infinito) de restricciones, y el complemento de cualquier restricción debe poder ser especificado por un número (posiblemente infinito) de restricciones. Si queremos razonar sobre la negación, la teoría $\mathcal{J}$ tiene que ser *completa respecto a la satisfacción*, es decir, cada restricción es o bien demostrablemente satisfacible o bien demostrablemente insatisfacible. Si esta última condición no se cumple, el lenguaje no hereda los resultados del esquema referentes al fallo finito y a la negación como fallo.

Algunos ejemplos de lenguajes definidos dentro del esquema CLP son CLP($\mathcal{R}$) [80], CLP(Σ^*) [155], Cosylog [16], CHIP [44] y CLP($\mathcal{H}/\mathcal{E}$) [3].

Höhfeld y Smolka [62] presentaron un nuevo marco que generaliza el esquema CLP para hacerlo aplicable a especificaciones en cláusulas definidas sobre lenguajes de restricciones arbitrarios. En este esquema, el lenguaje de restricciones puede tener más de una interpretación. Las restricciones se definen sin hacer ninguna asunción sobre su sintaxis. En realidad, una restricción representa los valores que las variables pueden tomar en las interpretaciones. Por lo tanto, las interpretaciones del lenguaje de restricciones no deben satisfacer ninguna condición a priori. Esta característica hace que el esquema pueda aplicarse, entre otros, a algunos lenguajes que no cumplen las condiciones del esquema CLP. En particular, ha sido utilizado para la definición del lenguaje

FALCON [29], para definir los fundamentos semánticos de LOGIN [2] y para definir un mecanismo de deducción ecuacional de primer orden [89].

La primera propuesta para la programación lógica-funcional con restricciones fue realizada por Darlington, Guo y Pull en [29], al presentar un nuevo esquema para definir lenguajes lógico-funcionales sobre estructuras de restricciones arbitrarias. Estos autores definen el esquema CFLP($\mathcal{X}$) como CLP(FP($\mathcal{X}$)), es decir, como la instancia CLP($\mathcal{X}$) en la que la estructura $\mathcal{X}$ se enriquece con nuevas funciones definidas por medio de un lenguaje funcional y donde las restricciones incluyen ecuaciones, resueltas por "narrowing", entre expresiones FP($\mathcal{X}$). Posteriormente, en [104] se definió un esquema general para programación lógica-funcional de primer orden con restricciones, CFLP($\mathcal{X}$), que juega un papel similar al del esquema CLP($\mathcal{X}$) pero para lenguajes lógico-funcionales perezosos con disciplina de constructores. Como en el esquema CLP, las estructuras en las que va a tener lugar la computación están parametrizadas por una signatura constituida por símbolos de función predefinidos y símbolos de predicado. Para definir nuevos símbolos de función y de predicado se utilizan reglas de reescritura condicionales con restricciones. La semántica operacional para CFLP($\mathcal{X}$) es un procedimiento de "narrowing" condicional perezoso restringido, en el que la unificación se ha sustituido por resolución de restricciones.

Otra extensión interesante, tanto de la programación lógica como de la programación ecuacional, consiste en incorporar negación. A continuación analizamos la problemática asociada a esta nueva extensión, bastante reciente en lo que respecta a la programación ecuacional.

1.1 Negación

La potencia expresiva de los lenguajes ecuacionales puede incrementarse extendiendo el formalismo para permitir en las reglas el uso directo tanto del símbolo $=$ como del $\neq$. Por ejemplo, las especificaciones axiomáticas de tipos de datos se expresan de forma más concisa y apropiada usando disecuaciones explícitamente. En algunos casos incluso resultan imprescindibles, como cuando se pretende especificar funciones y tipos de datos completos. Similarmente, la negación del predicado igualdad es relevante también para el esquema de programación lógica con restricciones (CLP) [73, 74].

Sin embargo, el estudio de las teorías ecuacionales y de los sistemas de reescritura que incluyen información negativa en forma de disecuaciones en las condiciones de las cláusulas y reglas, sólo ha sido abordado muy recientemente. Como apunta Kaplan en [85], algunas de las razones que han influido en este hecho son:

* este tipo de especificaciones ecuacionales no presentan un comportamiento declarativo directo, es decir, así como las especificaciones (condicionales) positivas tienen una semántica por modelo mínimo (o, alternativamente, por álgebra inicial), las especificaciones que incluyen disecuaciones no admiten, en general, un modelo mínimo sino modelos minimales (llamados "quasi-initial" en [86]), como mostraremos en el Capítulo 4.

* las especificaciones con negación son al menos tan complicadas como las positivas y éstas últimas siguen siendo objeto de estudio. Nótese que las investigaciones en el campo de las especificaciones positivas continúan y todavía no se han resuelto completamente todos los problemas asociados a las mismas (por ejemplo, ver [35] para un resumen de distintos problemas abiertos dentro del campo de la reescritura).

Esta situación no corresponde a lo que ha ocurrido en programación lógica, donde el problema de la incorporación de la negación fue tratado mucho más tempranamente. Como consecuencia, actualmente se tiene un conocimiento bastante profundo de la problemática asociada a la negación y de sus posibles soluciones. A continuación mostraremos cómo este conocimiento se ha usado en algunos casos para intentar dar solución a la negación en programación ecuacional.

1.1.1 Negación en programación lógica

La programación lógica con cláusulas definidas no parece ser un formalismo bastante expresivo para ciertas aplicaciones, como es el caso de los sistemas expertos. En particular, en este campo de aplicación se hace necesario poder expresar la negación. Es importante comprender que la negación incrementa la potencia expresiva de la programación lógica. Esto podría parecer paradójico ya que, como es bien conocido, los programas lógicos sin negación tienen la potencia completa de la teoría de la recursión. Sin embargo, en muchas ocasiones la computación se realiza sobre dominios finitos y esto cambia drásticamente la situación. El conocimiento de este hecho ha provocado estudios profundos de las extensiones de la programación lógica para incorporar el uso de la negación [7, 23, 50, 97, 103, 130, 132, 144].

La SLD-resolución utilizada en programación lógica (cláusulas de Horn definidas) es un ejemplo de método de razonamiento correcto en el sentido de que para toda fórmula libre de variables φ, $P \vdash \varphi$ implica que $P \models \varphi$, donde $P \vdash \varphi$ denota que φ puede probarse a partir del programa P. Sin embargo, la SLD-resolución es una forma muy restringida de razonamiento ya que con ella sólo podemos deducir hechos positivos. Para poder inferir información negativa se necesitan reglas especiales.

Una posibilidad es considerar la siguiente regla de inferencia

$$\frac{A \text{ no puede probarse desde } P}{\neg A}$$

donde A es un átomo básico y P un programa. Esta regla es conocida como *asunción de mundo cerrado* (CWA) y fue introducida por Reiter [139].

La CWA es un ejemplo de regla de inferencia no monótona. Una regla de inferencia es no monótona si al añadir nuevos axiomas, otros axiomas que eran válidos dejan de serlo.

La CWA es una regla usada habitualmente en el contexto de las bases de datos: en las bases de datos relacionales la información que no está explícitamente presente en la base se considera falsa. En programación lógica la situación es algo más complicada ya que la información contenida en un programa no puede determinarse inspeccionando simplemente el programa, por existir cláusulas no unitarias. Por lo tanto, cada aplicación particular es la que debe determinar la conveniencia o no de usar la CWA. Desafortunadamente, la CWA no es un método de razonamiento efectivo (ver [6] donde se demuestra que existen programas para los que el conjunto de átomos negados inferidos por la CWA no es recursivamente enumerable).

Una forma posible de resolver este problema consiste en considerar que $\neg A$ queda probado cuando intentamos probar A (usando SLD-resolución) y este intento falla finitamente.

Un árbol de derivación SLD falla finitamente si no contiene la cláusula vacía, lo que significa que todas sus ramas son derivaciones SLD falladas. Dado un programa P, su conjunto de fallo finito es el conjunto de todos los átomos básicos A tal que existe un árbol de derivación SLD fallado finitamente con $\leftarrow A$ como raíz.

Podemos ahora reemplazar la CWA por una nueva regla de inferencia:

$$\frac{A \text{ pertenece al conjunto de fallo finito de } P}{\neg A}$$

denominada *regla de negación como fallo.*

Clark [23] fue el primero en estudiar esta regla con detalle. Ya que el conjunto de fallo finito de un programa P es un subconjunto del complementario del conjunto de éxitos de P, la regla de negación como fallo es menos potente que la CWA. Sin embargo, en la práctica, se puede implementar fácil y eficientemente. Supongamos que A es un átomo básico de la base de Herbrand de un programa P y que tenemos como objetivo $\leftarrow \neg A$. El sistema intenta probar el objetivo $\leftarrow A$. Si $\leftarrow A$ tiene éxito entonces $\leftarrow \neg A$ falla, mientras que si falla finitamente entonces $\leftarrow \neg A$ tiene éxito.

Otra forma de inferir información negativa desde un programa lógico es usar el concepto de compleción de un programa. Un programa lógico sólo contiene la parte 'si' de las definiciones de los símbolos de predicado. Esto no nos permite concluir hechos negativos ya que su propia base de Herbrand es un modelo del programa. Ni siquiera podemos deducir información negativa de un programa normal, aunque éste permite literales negados en el cuerpo de sus cláusulas. La *compleción* de un programa [23] consiste en añadir al programa las partes 'sólo-si' de las definiciones de los símbolos de predicado, y una teoría de la igualdad (CET). Este proceso de compleción es otra forma de capturar la idea de que la información que no está en el programa es falsa. El concepto de respuesta correcta puede extenderse en este contexto considerando que una respuesta es correcta si el objetivo, instanciado con la sustitución respuesta, es una consecuencia del programa completado. La contrapartida procedural a este concepto declarativo es el concepto de respuesta computada. El mecanismo operacional elegido usualmente para computar respuestas es la SLDNF-resolución, esto es, SLD-resolución aumentada con la regla de negación como fallo.

Desafortunadamente, las implementaciones de la negación como fallo sin ningún tipo de restricción carecen de propiedades elementales como la corrección. Por ejemplo, consideremos la implementación del metapredicado `not` en Prolog:

```
not(G) :-call(G), !, fail.
not(G).
```

y el siguiente programa P:

```
eq(X, X).
p :-not(eq(X, 1)), eq(X, 2).
```

que consta del axioma reflexivo de la igualdad y de una cláusula que define el predicado `p`.

Consideremos ahora el objetivo `?- p.` que tiene como solución `X=2`. Este objetivo deriva un nuevo objetivo:

`?- not(eq(X,1)),eq(X,2).`

La primera observación que podemos hacer es que, dada la presencia del metapredicado corte en la definición del `not`, el orden de selección de los literales dentro del objetivo es relevante; de hecho, si seleccionáramos el literal `eq(X,2)` en primer lugar, encontraríamos la respuesta. Ahora bien, según la regla de selección estándar utilizada en Prolog, el próximo literal a resolver es `not(eq(X,1))`. La regla de negación como fallo genera una subderivación con objetivo inicial `eq(X,1)` que tiene éxito con respuesta computada `X=1`. Ante este hecho, la regla de negación como fallo llega a la conclusión errónea de que `not(eq(X,1))` es falso, al asumir de forma incorrecta que $P \models \forall$ `X eq(X,1)`.

La solución habitual para asegurar la corrección de la SLDNF-resolución consiste en restringir la aplicación de la regla de negación como fallo al caso básico, es decir, a literales sin variables. Esta condición se conoce como *condición de seguridad* ("safeness condition") sobre la selección de los literales. Sin embargo, esta solución introduce el problema del tropiezo ("floundering") cuando no existen literales negativos seleccionables. Este problema se mantiene incluso cuando se introducen métodos de retraso en la selección de un literal negado [156] debido a que éste puede no llegar a ser básico, como se muestra en el siguiente ejemplo:

```
es_menor(antonio).
es_mayor(sara).
puede_trabajar(X) :- not(es_menor(X)).
```

Es evidente que `puede_trabajar(sara)` es una consecuencia lógica del programa. Sin embargo, no existe ninguna respuesta computada para el objetivo `?- puede_trabajar(X).` debido a que se produce un tropiezo en el subobjetivo `not(es_menor(X))`.

El problema del tropiezo de un objetivo normal G puede resolverse imponiendo las siguientes restricciones:

* todas las cláusulas del programa normal P deben ser *admisibles*, es decir, cada variable en la cláusula ocurre o bien en la cabeza o bien en un literal positivo del cuerpo de la misma,

* el objetivo G tiene que ser *permitido* ("allowed"), es decir, si G es de la forma $\leftarrow L_1, \ldots, L_n$ cada variable en G ocurre en un literal positivo de $L_1, \ldots, L_n$,

* cada cláusula que define un símbolo de predicado que ocurre en un literal positivo del objetivo G o en un literal positivo en el cuerpo de una cláusula del programa P es tal que cada variable en la cláusula ocurre en un literal positivo de su cuerpo.

Para facilitar la comparación entre la asunción de mundo cerrado y la noción de programa completado definimos la asunción de mundo cerrado para un programa P, CWA(P), como:

CWA(P)= $P \cup ext(P) \cup CET \cup DCA$

donde $ext(P) = \{\neg A : A$ es un literal positivo básico y $P \not\models A\}$, CET es la teoría de la igualdad de Clark y DCA es el axioma de cierre de dominio. Con esta notación, CWA(P) y comp(P) tienen en común que ambas son ejemplo de razonamiento por defecto, asumiendo que si alguna información positiva no puede de alguna forma probarse desde P

entonces es falsa. Pero para CWA(P) la noción de 'prueba' utilizada es la de la lógica de primer orden, mientras que para comp(P) la 'prueba' consiste en usar una cláusula del programa cuya cabeza unifica con el átomo. A simple vista, esta noción de prueba es más restringida en el sentido de que más átomos básicos deberán ser falsos bajo comp(P) que bajo CWA(P). En otras palabras, CWA(P) debería ser una consecuencia de comp(P). Sin embargo, esto no es siempre así ya que comp(P) añade en las partes 'sólo-si' de la definición completada de un predicado p nuevas sentencias universales, que pueden usarse para demostrar cosas sobre otros predicados distintos de p. Además, CWA(P) incluye el axioma de cierre de dominio, el cual hace que nos restrinjamos a modelos de Herbrand, lo que no ocurre con comp(P) (comp(P) puede tener modelos que no son de Herbrand). De hecho, cuando CWA(P) y comp(P) son compatibles, es decir, cuando comp(P) $\cup$ CWA(P) es consistente, es CWA(P) la que implica comp(P) puesto que CWA(P) es categórica (determina un modelo único) [144].

Tal y como hemos mencionado, el principal problema con la negación como fallo es que, en general, no pueden manejarse los subobjetivos negados no básicos. Además, estos subobjetivos negados sólo pueden usarse como test y no para encontrar valores para las variables que aparecen en ellos.

Una forma de resolver este problema consiste en extender la regla de negación como fallo de forma que los subobjetivos negados sean tratados igual que los positivos. Este es el espíritu de la aproximación presentada por Chan [21], que denominó *Negación Constructiva* porque permite construir la respuesta de los subobjetivos negados de forma similar a como se hace para los positivos. La idea básica planteada es que la respuesta a un objetivo negado es la negación del conjunto de respuestas de su versión positiva. Para evaluar $\neg Q$ se ejecuta una subderivación con Q como objetivo inicial. Si $A_1 \vee \ldots \vee A_k$ son las respuestas a Q entonces su negación son las respuestas a $\neg Q$, es decir, la regla de negación constructiva simplemente establece que $\neg Q \equiv \neg(A_1 \vee \ldots \vee A_k)$.

Esta regla de negación constructiva constituye la base de un nuevo mecanismo operacional: la SLD-CNF-resolución.

Desde un punto de vista de programación, el principal atractivo de esta aproximación es su compatibilidad con los sistemas Prolog.

Como en la negación como fallo, los subobjetivos negados se resuelven mediante subrefutaciones de su versión positiva, lo que permite explotar la eficiencia de las implementaciones Prolog.

El problema con la primera versión del procedimiento descrito por Chan en [21] es que, cuando el subobjetivo positivo tiene infinitas respuestas o tiene un árbol de derivaciones con ramas infinitas, entonces el procedimiento es incapaz de devolver ninguna respuesta del subobjetivo negado. Para solucionar este inconveniente Chan reformuló la regla de negación constructiva en [22]. Básicamente, el nuevo esquema consiste en dos pasos:

a) Usar un procedimiento de evaluación para reducir el subobjetivo no negado Q a una fórmula simplificada F.

b) Aplicar un procedimiento de negación y normalización a F para obtener una fórmula simplificada de $\neg Q$.

Tanto el procedimiento de resolución como el mecanismo de simplificación propuestos son específicos del universo de Herbrand.

El proceso de negación de las respuestas introduce de forma natural disecuaciones cuantificadas. Por lo tanto, no existe ningún impedimento para extender la sintaxis de los cuerpos de las cláusulas del programa y de los objetivos para incluir disecuaciones. Con esta modificación, los programas lógicos pueden verse como programas lógicos con restricciones, de la forma $s = t$ y $\forall Z\ s \neq t$, sobre el dominio de Herbrand. Similarmente, las respuestas son conjunciones de estas restricciones. Estos son los motivos que llevan a Stuckey [147] a caracterizar la negación constructiva en el marco del esquema CLP, lo que extiende su aplicabilidad a muchos más lenguajes y, en particular, para el dominio de Herbrand. Al igual que Chan, Stuckey extiende la sintaxis de los programas lógicos con restricciones. Así, un subobjetivo complejo se define recursivamente como:

* un átomo, o

* $\neg\exists(c \quad B_1, \ldots, B_n)$ donde cada $B_i, 1 \leq i \leq n$, es un subobjetivo complejo y c es una restricción.

Un cuerpo complejo es una conjunción de subobjetivos complejos, mientras que la expresión $(c \quad B)$ es un objetivo complejo, donde c es una restricción y B es un cuerpo complejo. Sea $\mathcal{A}$ la estructura sobre la que se define el lenguaje. Informalmente, el mecanismo operacional que define Stuckey para manejar objetivos complejos es el siguiente: Si el objetivo actual es $(c \quad D_1, \ldots, D_n)$ y se selecciona para resolver un subobjetivo D_j de la forma $\neg\exists(c_q \quad G)$, entonces seleccionamos primero alguna porción c' de la restricción c para la que se cumple que $\mathcal{A} \models c \to c'$ (esta porción se pasará a la subderivación negativa para G con objeto de simplificar la restricción resultante de tal derivación). El objetivo $(c_q \wedge c' \quad G)$ se ejecuta hasta que alcanza alguna frontera finita (conjunto finito de nodos del árbol de derivaciones para G tal que cualquier derivación para G es fallada finitamente o pasa exactamente por un nodo de la frontera) elegida por la regla de computación. Los nodos de esta frontera se expresan como una disyunción que se niega y simplifica y cuyo resultado se transforma en una disyunción de objetivos. Finalmente, cada uno de estos objetivos sustituye al subobjetivo seleccionado en el objetivo actual. Si el subobjetivo seleccionado del objetivo actual es un átomo, entonces se mantiene el mecanismo operacional estándar definido para CLP. Stuckey demuestra la corrección y la completitud de este mecanismo operacional con respecto a las consecuencias trivaluadas de la compleción de un programa. Como ya hemos mencionado, una de las diferencias entre esta aproximación y la de Chan es que la simplificación de los nodos de la frontera cuando se seleccionan subobjetivos con negación no es particular al universo de Herbrand.

Un método particularmente interesante de explicar la semántica de la negación fue sugerido por Apt, Blair y Walker [7] al identificar una clase de programas lógicos con negación llamados *programas estratificados*. Algunas de las propiedades más notables de los programas estratificados son que su compleción es consistente y que su significado viene dado por un único modelo. Los programas estratificados se caracterizan por impedir ciertas combinaciones de negación y recursión. La estratificación define un orden bien fundado $<_p$ sobre el conjunto de predicados del programa. El orden $<_p$ se define de la siguiente forma: $p <_p q$ si p aparece en la cabeza de una cláusula y q en un literal negado en el cuerpo de la misma, mientras que $p \leq_p q$ si q aparece en un literal positivo. Un programa es estratificado si el orden $<_p$, implícitamente dado por la forma en que se escriben las cláusulas del programa, es un orden parcial bien fundado (o equivalentemente, noetheriano). Este orden se extiende de forma natural a una relación de prioridad entre átomos básicos $<$ y a una relación de preferencia entre interpretaciones $<<$. Un modelo N es preferible a otro modelo M, $N << M$, si y sólo si la incorporación a N de un átomo A de prioridad baja está siempre justificada con la eliminación de M de un átomo B de mayor prioridad, es decir, $A < B$. De acuerdo a la relación de preferencia, un modelo es perfecto si no existen modelos preferibles a él. Przymusinski [130] extendió la noción de programa lógico estratificado a las bases de datos deductivas, probando que cada base de datos estratificada tiene modelos perfectos y que cada programa lógico estratificado tiene exactamente un modelo perfecto. Además, este modelo perfecto (único) coincide con el modelo minimal y soportado considerado por Apt, Blair y Walker [7] lo que demuestra la independencia del modelo de la estratificación elegida.

Posteriormente, Przymusinski [132] generalizó esta aproximación a programas y a bases de datos *localmente estratificados*. Esta última clase extiende significativamente la clase de los programas y bases de datos estratificados, incluyendo muchos programas y bases de datos que no cumplen la condición de ser estratificados. Un programa (análogamente, una base de datos) es localmente estratificado si es posible definir un orden noetheriano sobre la base de Herbrand. Por ejemplo [132], el siguiente programa que define los números pares

$$\texttt{even(0).}$$
$$\texttt{even(s(X))} \Leftarrow \neg\texttt{even(X)}.$$

donde `s` representa la función sucesor sobre el conjunto de los números naturales, no es estratificado porque `even` $<_p$ `even` y, por lo tanto, el orden $<_p$ no es un orden parcial. Sin embargo, es un programa localmente estratificado porque cualquier secuencia decreciente de átomos básicos es de la forma

$$\texttt{even(s(s(...)))} < \texttt{even(s(...))} < \ldots < \texttt{s(0)} < \texttt{0}$$

y, por consiguiente, es finita.

Cada modelo de un programa localmente estratificado está subsumido por un modelo perfecto. En particular, cada programa localmente estratificado tiene al menos un modelo perfecto.

1.1.2 Negación en programación ecuacional y lógica-ecuacional

La negación en programación ecuacional se ha tratado desde enfoques muy diversos.

Colmerauer [24] fue el primero en discutir el problema de resolver ecuaciones y disecuaciones en el marco de la programación lógica al dar la definición semántica del lenguaje PROLOG II. En este trabajo, Colmerauer mostró que se pueden realizar ciertas transformaciones sobre los sistemas de ecuaciones y disecuaciones para obtener sistemas 'irreducibles' (llamados *formas resueltas* en [24]) que tienen al menos una solución en el álgebra de los árboles racionales. Posteriormente, Lassez, Maher y Marriott en [100] desarrollaron esta misma aproximación demostrando los mecanismos fundamentales y probando que algunos sistemas no pueden ser reducidos para tener una solución por unificación.

Por otra parte, algunos sistemas de disecuaciones se han usado, por ejemplo, en el aprendizaje por contraejemplos [101] o para resolver un problema clásico de la teoría de los sistemas de reescritura como es la completitud suficiente [27]. La completitud suficiente coincide con la propiedad de 'no basura' ("no junk") en las especificaciones algebraicas, es decir, una especificación es suficientemente completa si no define términos que no se puedan construir usando los símbolos de función que aparecen en la especificación.

Un intento de unificar todos los problemas anteriormente citados en un mismo marco es el de considerar problemas ecuacionales en general, es decir, sistemas conteniendo ecuaciones, disecuaciones, conjunciones, disyunciones y variables cuantificadas universalmente. De hecho, los problemas de unificación son problemas ecuacionales sin cuantificación universal, sin disecuaciones y sin disyunciones. Similarmente, el problema de la validez es un problema ecuacional en el que todas las variables están cuantificadas universalmente.

Una aproximación para resolver problemas ecuacionales es la presentada en [90], al proponer un conjunto de reglas para transformar los problemas ecuacionales en formas resueltas, aunque en este trabajo no se da ningún resultado referente a la completitud de dichas reglas. Comon y Lescanne [28] retoman la idea anterior y presentan un sistema de varias reglas, completo y correcto, para transformar los problemas ecuacionales en otros más simples (preservando las soluciones) con la idea de que el último problema exprese directamente la solución. En realidad, Comon y Lescanne pretenden con esta aproximación dar una generalización de los algoritmos de unificación. Para ello, incluyen ciertos controles para asegurar la terminación de la aplicación de las reglas. Estos controles son bastante débiles para que el algoritmo sea lo más general posible y para que se pueda, refinando el control, obtener cualquier algoritmo de unificación (por ejemplo, el de Robinson, el de Martelli y Montanari, ...). Una aproximación diferente es la de Maher [105] ya que en este trabajo el problema se resuelve mediante un procedimiento de eliminación de cuantificadores que hace uso del algoritmo de forma resuelta. El método propuesto es válido para las álgebras de los árboles finitos, infinitos y racionales para los casos de signatura finita e infinita. No obstante, todos los trabajos mencionados hasta ahora investigan la resolución de ecuaciones y disecuaciones con respecto a la teoría ecuacional vacía.

Otras propuestas diferentes toman en consideración teorías ecuacionales arbitrarias. Así, el problema de la $\mathcal{E}$-disunificación, es decir, el proceso de resolver disecuaciones en el cociente del universo de Herbrand módulo una teoría ecuacional $\mathcal{E}$, se estudia en [49] y en [20]. Fernández [49] demuestra que la $\mathcal{E}$-disunificación es semidecidible cuando la teoría ecuacional $\mathcal{E}$ puede presentarse como un sistema de reescritura convergente básico. Asimismo, define un procedimiento correcto y completo de $\mathcal{E}$-disunificación basado en "narrowing" y en reducibilidad básica. Su principal inconveniente es que, en general, es ineficiente, al heredar la complejidad computacional del test de reducibilidad básica. No obstante, para algunos casos especiales, como los sistemas de reescritura lineales por la izquierda y los sistemas con constructores, el procedimiento es eficiente. Una solución diferente es la propuesta por Bürckert en [20]. Bürckert demostró que en cualquier teoría en la que es posible representar las soluciones de un conjunto de ecuaciones por un conjunto finito de sustituciones es también posible representar el conjunto de soluciones de un sistema que contiene ecuaciones y disecuaciones por un conjunto finito de *sustituciones con excepciones*; en otras palabras, la $\mathcal{E}$-disunificación es decidible cuando la $\mathcal{E}$-unificación es decidible y finitaria y el conjunto de símbolos de función es finito. Conceptualmente, el procedimiento propuesto consiste en reducir la $\mathcal{E}$-disunificación a $\mathcal{E}$-unificación. Técnicamente, dado un sistema de ecuaciones y disecuaciones S, el procedimiento a seguir consiste en encontrar un par de sustituciones (σ, ψ) tal que σ sea una solución a las ecuaciones en S y ψ una solución a las disecuaciones de S expresadas como ecuaciones. La solución de S es la diferencia entre ambas, es decir, la sustitución $(\sigma - \psi)$ que se llama sustitución con excepción.

En [85, 112, 113] se aborda el problema de resolver objetivos con respecto a un sistema de reescritura condicional que puede contener disecuaciones en el antecedente de las reglas. En [112], Mohan y Srivas siguen una aproximación por reescritura definiendo un nuevo mecanismo de reescritura condicional, denominado EI-reescritura. La igualdad en el antecedente de las reglas de reescritura se interpreta como 'demostrablemente igual' y se asume que se cumple si los dos miembros de la igualdad se reescriben a un término común. Análogamente, la desigualdad en el antecedente de las reglas se interpreta como 'no demostrablemente igual' y se asume que se cumple, por defecto, si puede demostrarse que dos términos básicos no se reescriben a un término común. La EI-reescritura es correcta y completa para especificaciones que satisfacen una propiedad similar a la de completitud suficiente. Kaplan [85] define también una nueva relación de reescritura extendiendo la relación de reescritura condicional positiva y que tiene en cuenta las disecuaciones en las condiciones de las reglas. Para que esta relación esté bien definida y sea significativa, el sistema de reescritura debe cumplir la condición de ser 'reduciente' (una condición similar a la de sistema 'simplificante' definida para los sistemas de reescritura condicionales positivos [86]). En estos dos trabajos se impone la condición de que no aparezcan variables ni en la parte derecha de la cabeza ni en las condiciones de las reglas de reescritura que no estén contenidas en la parte izquierda de la cabeza de las mismas. Mohan y Srivas relajan esta última condición en [113] al permitir nuevas variables en la parte condicional de las reglas de reescritura. Operacionalmente, esta generalización se traduce en el uso de "narrowing" y de técnicas de $\mathcal{E}$-unificación para comprobar la satisfacibilidad (o insatisfacibilidad) de los antecedentes de las reglas condicionales. El tratamiento de la negación en estas propuestas se aproxima más a la *negación como fallo* de la programación lógica que a una deducción explícita de las disecuaciones.

Mohan, Srivas y Kapur presentan una nueva aproximación en [114, 115] en la que las desigualdades se deducen directamente usando un sistema de inferencia completo que permite inferir explícitamente disecuaciones a partir de otras ya existentes. Estas reglas de inferencia para las disecuaciones se construyen como la contrapartida de las reglas de inferencia ecuacionales (reflexiva, simétrica y transitiva) y generalizando la regla de sustitutividad de funciones para las disecuaciones. Estas nuevas reglas sirven de base para definir un procedimiento de semidecisión dirigido por el objetivo y basado en "narrowing" para deducir consecuencias disecuacionales de un sistema S. Para probar que una disecuación $s \neq t$ es consecuencia de un sistema de ecuaciones y disecuaciones S, el procedimiento parte de una disecuación de S y de la ecuación $s = t$ 'skolemizada' e intenta llegar a una contradicción.

Wos y McCune [157] tratan también estos sistemas de ecuaciones y disecuaciones pero en el campo del razonamiento automático. La solución propuesta consiste en definir dos nuevas reglas de inferencia para tratar las disecuaciones: la paramodulación negativa y la hiperparamodulación negativa, que son, respectivamente, las contrapartidas negativas de la paramodulación y la hiperparamodulación, usadas estas últimas para razonar únicamente con ecuaciones. En este trabajo se aplican técnicas refutacionales para derivar una contradicción después de añadir al sistema la negación 'skolemizada' del objetivo. Tales técnicas también se han aplicado, por ejemplo, para tratar el problema de las palabras [66].

Kuchen, López, Moreno y Rodriguez [96] presentan una extensión del lenguaje lógico-funcional perezoso BABEL que incorpora restricciones desigualdad para resolver ecuaciones y construir respuestas. Estas respuestas desigualdad incrementan la potencia expresiva del lenguaje ya que pueden reemplazar en algunos casos a respuestas positivas infinitas. La semántica operacional propuesta combina "narrowing" perezoso con resolución de restricciones desigualdad.

Bachmair y Ganzinger [11] estudian los programas lógicos con funciones representados formalmente por cláusulas de primer orden con igualdad. La aproximación seguida se basa en reescritura de términos ordenada, una técnica en la que uno asume la existencia de un orden bien fundado sobre los términos que sirve para orientar las ecuaciones en reglas de reescritura. Para estos programas lógico-ecuacionales, Bachmair y Ganzinger presentan un sistema de inferencia refutacionalmente completo, construido en torno a una versión restringida de la paramodulación llamada superposición [10]. Asimismo, demuestran que cualquier programa lógico consistente para el que existe un orden total sobre expresiones básicas tiene un modelo perfecto. Esto permite no restringirse a programas estratificados ya que si existe un orden total bien fundado $>$, cualquier programa de primer orden consistente puede transformarse en un programa (posiblemente infinito) el cual es estratificado bajo redundancia con respecto a $>$. Esta transformación del programa consiste, básicamente, en saturar las cláusulas usando el sistema de inferencia S que presentan. Una cláusula es redundante (en un programa P) si no contribuye a la construcción del modelo perfecto. Un programa P es saturado (bajo S) si cada cláusula básica que puede obtenerse a partir de una instancia básica de una cláusula no redundante de P usando una regla de S es una instancia básica de una cláusula de P o bien es redundante en P. La conexión con la programación lógica sin igualdad es directa ya que un programa lógico sin igualdad consistente es saturado.

Como en programación lógica, otra forma de abordar el estudio del significado de un programa ecuacional con negación es usar la aproximación propuesta por Fitting [50] para los programas lógicos estilo Prolog. Esta es la base de la semántica propuesta por Mohan [111] para los sistemas de reescritura condicionales con negación. Mohan da una descripción a tres valores de la relación de reescritura donde $T\langle p, q\rangle$ significa que p se reescribe a q, mientras que $F\langle p, q\rangle$ significa que se sabe que p no se reescribe a q. Si $p \to q$ no es ni $T\langle p, q\rangle$ ni $F\langle p, q\rangle$ entonces la reducción desde p a q no se sabe si es cierta o falsa, es decir, $p \to q$ tiene el tercer valor de verdad. Siguiendo esta idea, Mohan define una aplicación monótona Φ asociada al sistema de reescritura de términos, dando una caracterización por punto fijo de la semántica del mismo basada en Φ.

Desde un punto de vista operacional, la aproximación que presentamos en esta tesis consiste en definir un algoritmo basado en "narrowing" que resuelve constructivamente la negación con respecto a una teoría ecuacional que admite, a su vez, negación en las condiciones de las cláusulas. El método que describimos en los capítulos siguientes difiere de otras propuestas en que es capaz de dar respuesta tanto a objetivos básicos como no básicos, y en que las disecuaciones se tratan constructivamente, de forma que las respuestas a $t \neq s$ se construyen a partir de las respuestas a $t = s$. Desde un punto de vista declarativo, enriquecemos la semántica declarativa estándar de la programación ecuacional definiendo el concepto de compleción de una teoría ecuacional. Esta nueva noción nos permite razonar sobre la negación en algunas teorías ecuacionales que cumplen ciertas condiciones sintácticas que garantizan la consistencia de su compleción.

Capítulo 2

Teorías ecuacionales normales

Este capítulo está dedicado a la definición de las teorías ecuacionales normales, una extensión de las teorías ecuacionales de Horn en la que se permite la presencia explícita en las condiciones de las cláusulas tanto del símbolo de igualdad como del de desigualdad, incrementando con ello la potencia expresiva del formalismo ecuacional estándar.

La característica más destacada de las teorías ecuacionales normales es que el antecedente (o cuerpo) de cada cláusula ecuacional es una conjunción de ecuaciones y disecuaciones (posiblemente vacía) mientras que el consecuente (o cabeza) es una ecuación. Estas cláusulas conteniendo disecuaciones se corresponden con el uso del operador 'Si_entonces_sino' en las especificaciones positivas, aunque las ecuaciones condicionales con negación expresan la relación condicional entre términos de manera mucho más directa y declarativa que la aproximación 'Si_entonces_sino'. Por lo tanto, las teorías normales constituyen un formalismo mucho más natural para expresar tipos de datos y especificaciones algebraicas.

Este capítulo está organizado como sigue:

En la sección primera se introducen algunas definiciones y conceptos generales sobre ecuaciones, sistemas de reescritura de términos, teorías ecuacionales y unificación universal, que serán utilizados tanto en este capítulo como en los posteriores. Esta sección tiene como objeto facilitar la posible consulta de las definiciones técnicas usadas en la tesis.

La sección segunda describe la sintaxis de las teorías ecuacionales normales. De forma análoga a como se hace con las teorías ecuacionales de Horn, definimos el sistema de reescritura asociado con una teoría normal dada y revisamos la aproximación por reescritura formulada por Kaplan. Asimismo, abordamos el estudio de la canonicidad de estas teorías, presentando condiciones suficientes que garantizan la terminación y la confluencia de la relación de reescritura asociada al conjunto de ecuaciones definidas por una teoría ecuacional normal.

2.1 Preliminares

Los símbolos V y Σ denotan, en lo que sigue, conjuntos numerables de variables y de símbolos de función, respectivamente. El conjunto Σ se llama signatura. $\mathcal{T}(\Sigma)$ denota el conjunto de términos básicos (i.e. sin variables) sobre Σ y $\mathcal{T}(\Sigma \cup V)$ denota el conjunto de términos con variables. El conjunto $\mathcal{T}(\Sigma)$ también se conoce como el universo de Herbrand $\mathcal{H}$ sobre Σ. Una *ecuación*, $t = s$, es un par de términos donde $t, s \in \mathcal{T}(\Sigma \cup V)$. Un *objeto sintáctico* es bien una sustitución, un término, una ecuación o un sistema de ecuaciones (una conjunción de ecuaciones). $\mathcal{V}(\mathcal{O})$ denota el conjunto de variables de un objeto sintáctico $\mathcal{O}$. Un objeto sintáctico se llama un Σ-objeto si cada uno de los símbolos de función que ocurren en él pertenece a Σ. Una secuencia de términos (o, en general, literales) $t_1, \ldots, t_n$ se denotará por $\tilde{t}$.

2.1.1 Sustituciones

Una Σ-sustitución θ es una aplicación de V en $\mathcal{T}(\Sigma \cup V)$ (extendida de manera natural a $\theta : \mathcal{T}(\Sigma \cup V) \to \mathcal{T}(\Sigma \cup V)$). Las Σ-sustituciones pueden ser extendidas a otros objetos sintácticos de la manera usual. Para cada Σ-sustitución θ se define el conjunto de variables afectadas por θ o *dominio de* θ como $D(\theta) = \{x : \theta(x) \neq x\}$ y el conjunto de *variables introducidas por* θ como $I(\theta) = \bigcup_{x \in D(\theta)} \mathcal{V}(\theta(x))$. Sea $W \subseteq V$ un conjunto de variables. Se define la *restricción de* θ *a* W como $\theta\uparrow_W (x) = \theta(x)$ si $x \in W$ y $\theta\uparrow_W (x) = x$ en cualquier otro caso. Decimos que $\sigma = \theta[W]$ si $\sigma\uparrow_W = \theta\uparrow_W$.

Una sustitución $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ puede ser representada como un sistema de ecuaciones $\hat{\theta} = (x_1 = t_1, \ldots, x_n = t_n)$, donde x_i es una variable y t_i es un término en $\mathcal{T}(\Sigma \cup V)$. $\hat{\theta}$ es la *representación ecuacional* de θ. La sustitución vacía se denota por ϵ y su representación ecuacional por $\emptyset$. Una sustitución σ es un *renombramiento de variables* si y sólo si es una biyección de V a V. Equivalentemente, σ es un renombramiento si y sólo si existe su inversa, es decir, una sustitución σ^{-1} tal que $\sigma\sigma^{-1} = \sigma^{-1}\sigma = \epsilon$. Decimos que una sustitución $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ es básica si todos los términos t_i son básicos.

Sea $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ una sustitución y $\mathcal{O}$ un objeto sintáctico. La *instancia* de $\mathcal{O}$ por θ, denotada por $\mathcal{O}\theta$, es el objeto sintáctico que se obtiene a partir de $\mathcal{O}$ reemplazando simultáneamente cada ocurrencia de la variable x_i en $\mathcal{O}$ por el término $t_i (1 \leq i \leq n)$. Análogamente, un objeto sintáctico $\mathcal{O}$ es una *instancia* de otro objeto sintáctico $\mathcal{O}'$ si existe una sustitución θ tal que $\mathcal{O} = \mathcal{O}'\theta$. Asimismo, un objeto sintáctico $\mathcal{O}$ es una *variante* de otro objeto sintáctico $\mathcal{O}'$ si $\mathcal{O}$ se obtiene a partir de $\mathcal{O}'$ por un renombramiento de variables consistente, es decir, si existen dos sustituciones de cambio de nombre θ y σ tales que $\mathcal{O} = \mathcal{O}'\theta$ y $\mathcal{O}' = \mathcal{O}\sigma$.

Se dice que dos términos t y s son *unificables* si y sólo si existe una sustitución θ tal que $t\theta = s\theta$. Decimos que una sustitución θ es un *unificador más general* (mgu) de t y s si y sólo si para cualquier otro unificador θ' de t y s existe una sustitución δ tal que $\theta' = \theta\delta$. La definición de unificador se extiende de forma natural a otros objetos sintácticos. Por ejemplo, si S es un conjunto de ecuaciones, una sustitución θ es un unificador de S si es un unificador de todas y cada una de las ecuaciones de S.

2.1.2 Ocurrencias

Con objeto de disponer de una notación para tratar con los subtérminos de un término, y considerando la representación usual de un término como un árbol etiquetado, definimos el concepto de *ocurrencia* como una secuencia de enteros que denotan un camino de acceso en un término. Sea N_+^* el conjunto de las secuencias finitas de enteros positivos, Λ la secuencia vacía y $\cdot$ la operación de concatenación de secuencias. Para un término t, definimos su conjunto de ocurrencias,

$O(t)$, como un subconjunto finito de N_+^* de la siguiente forma:

a) $\Lambda \in O(t)$,

b) $u \in O(t_i) \Rightarrow i \cdot u \in O(f(t_1, \ldots, t_n)) \, \forall i : 1 \leq i \leq n$.

Si $u \in O(t)$ entonces se define el subtérmino de t a la ocurrencia u, t/u, y la sustitución en t del subtérmino a la ocurrencia u por el término t', $t[t']_u$, como:

a) $t/\Lambda = t$,

b) $f(t_1, \ldots, t_n)/i \cdot u = t_i/u$,

c) $t[t']_\Lambda = t'$,

d) $f(t_1, \ldots, t_n)[t']_{i \cdot u} = f(t_1, \ldots, t_i[t']_u, \ldots, t_n)$.

El conjunto de ocurrencias no variables de un término t, $\bar{O}(t)$, se define como $\{u \in O(t) : t/u \notin V\}$.

2.1.3 Sistemas de Reescritura de Términos

Definición 2.1.1 Regla de reescritura

Una regla de reescritura es una ecuación dirigida $l \to r$ tal que $l \notin V$ y $\mathcal{V}(r) \subseteq \mathcal{V}(l)$.

Si $l \to r$ es una regla de reescritura y σ es un renombramiento de variables, entonces la regla de reescritura $l\sigma \to r\sigma$ se llama una *variante* de la regla $l \to r$. Un *sistema de reescritura de términos* (SRT) es un conjunto finito de reglas de reescritura.

Definición 2.1.2 Linealidad por la izquierda

Una regla de reescritura $l \to r$ es lineal por la izquierda ("left-linear") si l no contiene ocurrencias múltiples de la misma variable. Un SRT lineal por la izquierda es aquél que sólo contiene reglas de reescritura lineales por la izquierda.

La *relación de reescritura* $\rightarrow_R$ asociada a un SRT R se define de la siguiente forma:

Definición 2.1.3 Relación de reescritura

Dado un SRT R, decimos que un término t se reescribe al término s, $t \rightarrow_R s$, si existe una variante de una regla de reescritura $l \rightarrow r$ de R, una ocurrencia $u \in O(t)$ y una sustitución σ tales que $t/u = l\sigma$ y $s = t[r\sigma]_u$. El término $l\sigma$ se llama un redex (expresión reducible) y puede reemplazarse por su " contractum" $r\sigma$.

Con frecuencia, la relación $\rightarrow_R$ también se llama *relación de reducción en un paso*. El cierre reflexivo-transitivo de $\rightarrow_R$ se denota por $\rightarrow^*_R$. Si $t \rightarrow^*_R s$ decimos que t se *reduce* a s. La *relación de convertibilidad*, denotada como $=_R$, es la relación de equivalencia generada por la relación $\rightarrow_R$, i.e. $\longleftrightarrow^*_R$. Si $\mathcal{E}$ es el conjunto de ecuaciones que corresponde al SRT R, es decir, $\mathcal{E} = \{l = r : l \rightarrow r \in R\}$, entonces $=_R$ coincide con $=_\mathcal{E}$, donde $=_\mathcal{E}$ denota la relación de congruencia más fina cerrada bajo instanciación generada por $\mathcal{E}$.

Dados dos términos t_1 y t_2, decimos que son "*joinable*" o que *tienen un reducto común*, denotado por $t_1 \downarrow_R t_2$, si existe un término t_3 tal que $t_1 \rightarrow^*_R t_3$ y $t_2 \rightarrow^*_R t_3$. En lo que sigue omitiremos el subíndice R cuando éste quede claro a partir del contexto.

Definición 2.1.4 Forma normal

Decimos que un término t está en forma normal (o que es irreducible) si no se reescribe a ningún otro término, es decir, si $\nexists s : t \rightarrow s$.

Definición 2.1.5 SRT terminante

Un SRT se llama terminante o noetheriano si no existen secuencias infinitas de reescrituras $t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow \ldots$. En otras palabras, cada secuencia de reescrituras termina eventualmente en un término en forma normal.

Definición 2.1.6 SRT localmente confluente

Un SRT es localmente confluente si para todo s, t_1, t_2 *tales que* $s \to t_1$ *y* $s \to t_2$ *se cumple que* $t_1 \downarrow t_2$.

Definición 2.1.7 SRT confluente

Un SRT es confluente o tiene la propiedad de Church-Rosser si para todo s, t_1, t_2 *tales que* $s \to^* t_1$ *y* $s \to^* t_2$ *se cumple que* $t_1 \downarrow t_2$.

Definición 2.1.8 SRT confluente básico

Un SRT es confluente básico si es confluente para la evaluación de términos básicos.

Definición 2.1.9 SRT canónico

Un SRT es canónico (convergente o completo) si es confluente y terminante.

Sean $l_1 \to r_1$ y $l_2 \to r_2$ dos variantes (sin variables comunes) de dos reglas de reescritura de un SRT R. Supongamos que existe una ocurrencia $u \in \bar{O}(l_1)$ tal que l_1/u y l_2 son unificables con mgu σ, es decir, $(l_1/u)\sigma = l_2\sigma$, entonces el par $(l_1[r_2]_u\sigma, r_1\sigma)$ es un *par crítico* de R y decimos que l_1 y l_2 *se superponen*. Un par crítico (t, s) es *convergente* si $t \downarrow s$. El Lema del Par Crítico [68, 94] establece que un SRT es localmente confluente si y sólo si todos sus pares críticos son convergentes. Un corolario inmediato de este lema establece que un SRT terminante es confluente si y sólo si todos sus pares críticos son convergentes [94].

En general, las propiedades de terminación y confluencia son indecidibles. No obstante, existen situaciones para las que ambas o alguna de ellas es decidible. Así, por ejemplo, para los SRTs terminantes la confluencia es decidible: el algoritmo de *Knuth-Bendix* intenta generar a partir de un SRT R no confluente, otro SRT R' canónico y equivalente al primero. Para ello, el procedimiento busca pares críticos entre las reglas de R. Si el par encontrado no es convergente, entonces lo orienta para convertirlo en una regla de reescritura (preservando la propiedad de terminación del sistema de reescritura) que se añade al sistema. El procedimiento acaba cuando no existen más pares críticos, en cuyo caso el sistema resultante es canónico y equivalente al de entrada, o cuando un par crítico no puede orientarse, en cuyo caso se concluye que el sistema de entrada no es confluente. Para orientar los pares críticos se hace uso de *órdenes de simplificación* sobre los términos. Un orden de simplificación entre términos '$>$' es un orden parcial bien fundado, compatible y cerrado bajo sustitución, subtérmino y eliminación, es decir,

* si $t > s$ entonces $f(\ldots,t,\ldots) > f(\ldots,s,\ldots)$ para todo término $f(\ldots)$
* si $t > s$ entonces $t\sigma > s\sigma$ para toda sustitución σ
* $f(\ldots,t,\ldots) > t$
* $f(\ldots,t,\ldots) > f(\ldots)$

La terminación de un SRT R queda garantizada si existe un orden de simplificación $>$ tal que $l\sigma > r\sigma$ para toda sustitución σ y toda regla $l \to r$ de R. La concepción de órdenes de simplificación adecuados ha constituido una de las actividades investigadoras más activas dentro del campo de los sistemas de reescritura [30, 33, 38, 140]. Los SRTs que no cumplen la propiedad de terminación han sido estudiados, entre otros, por Dershowitz, Kaplan y Plaisted [36, 37] y Kennaway, Klop, Sleep y De Vries [87]. El interés teórico de la reescritura infinita está intimamente relacionado con la facilidad de algunos lenguajes de programación funcional, tales como Miranda [152], de tratar con términos potencialmente infinitos, así como con la correspondencia entre reescritura infinita y reescritura de grafos de términos.

El algoritmo de compleción de Knuth-Bendix, además del uso que hemos mencionado, puede usarse como un intérprete para programas lógicos [31] o como una herramienta para la síntesis de programas [32]. Esta última aplicación de los SRTs también se ha abordado usando directamente técnicas de reescritura [138].

Definición 2.1.10 SRT no ambiguo

Un sistema de reescritura es no ambiguo ("non-ambiguous" o "non-overlapping") si no contiene pares críticos.

En ausencia de la propiedad de terminación, un SRT puede ser no ambiguo y no ser confluente. Aún más, la condición de no ambigüedad no es suficiente para garantizar confluencia, como muestra el siguiente ejemplo [68]:

$$\begin{array}{ll} r_1: & f(X,X) \rightarrow a \\ r_2: & f(X,g(X)) \rightarrow b \\ r_3: & c \rightarrow g(c) \end{array}$$

Este SRT es no ambiguo ya que las partes izquierdas de las dos primeras reglas no solapan al fallar su unificación por el test del "occur-check". Sin embargo, el término f(c,c) tiene dos formas normales, a y b, obtenidas con las siguientes secuencias de reducciones:

$$\begin{array}{l} f(c,c) \stackrel{r_1}{\rightarrow} a \\ f(c,c) \stackrel{r_3}{\rightarrow} f(c,g(c)) \stackrel{r_2}{\rightarrow} b \end{array}$$

Definición 2.1.11 SRT ortogonal

Un SRT es ortogonal si es lineal por la izquierda y no ambiguo.

El resultado más relevante referente a estos últimos sistemas es que todo SRT ortogonal es confluente, independientemente de si posee o no la propiedad de terminación [67, 68]. La confluencia de los sistemas ortogonales implica que las formas normales pueden computarse por reescritura. Los sistemas de reescritura ortogonales han sido estudiados ampliamente (consultar, por ejemplo, el trabajo de Klop [93] que incluye los aspectos más destacados de la teoría de estos sistemas).

Otra propiedad importante de los SRTs es la *reducibilidad básica* ("ground reducibility"). La reducibilidad básica se ha aplicado, por ejemplo, para demostrar que una especificación algebraica es suficientemente completa (ver Definición 2.1.16) y para demostrar teoremas inductivos a partir de especificaciones algebraicas [120].

Definición 2.1.12 Reducibilidad básica

Un término t es reducible básico en un SRT R si todas sus instancias básicas, $t\sigma$, pueden reescribirse por R.

Finalmente, dada la necesidad de modularización de las especificaciones de los tipos abstractos de datos, podría resultar ventajoso que alguna de las propiedades de un SRT pudiera ser inferida a partir de su validez para diversas 'partes' de tal SRT. La definición más simple de 'partes' de un SRT es la obtenida por el concepto de la suma disjunta de SRTs. Por ello, en los últimos años se han realizado diversos estudios tendentes a establecer la modularidad de ciertas propiedades de los SRTs [108, 150, 151].

2.1.4 Sistemas de Reescritura de Términos Condicionales

Los *sistemas de reescritura de términos condicionales* (SRTC) han supuesto una extensión importante del campo de la reescritura de términos. Originalmente, los SRTCs surgen a partir de la teoría de los tipos abstractos de datos para expresar especificaciones que contienen ecuaciones condicionales positivas de la forma

$$t = s \Leftarrow t_1 = s_1, \ldots, t_n = s_n$$

En el caso incondicional, la transformación de ecuaciones en reglas de reescritura es sencilla ya que es suficiente con orientar las ecuaciones de izquierda a derecha, previa comprobación de que la parte izquierda no es una variable y de que las variables que ocurren en la parte derecha también están en la parte izquierda (estas restricciones no siempre se imponen). Sin embargo, en el caso condicional la transformación es más compleja ya que, además, se debe elegir cuál es la interpretación del símbolo $=$ en las condiciones de las reglas. Dershowitz, Okada y Sivakumar [41] hacen la siguiente distinción, que extiende la clasificación introducida por Bergstra y Klop [15]:

a) *sistemas semi-ecuacionales*: la igualdad en las condiciones se interpreta como $\longleftrightarrow^*$, i.e. el cierre reflexivo, simétrico y transitivo de la relación de reescritura. Es decir, las reglas son de la forma

$$t \to s \Leftarrow t_1 = s_1, \ldots, t_n = s_n$$

b) *sistemas naturales*: la igualdad en las condiciones se interpreta como reducibilidad a un reducto común. Es decir, las reglas son de la forma

$$t \to s \Leftarrow t_1 \downarrow s_1, \ldots, t_n \downarrow s_n$$

Esta definición es una de las más usadas en la literatura (ver [40, 82, 84]).

c) *sistemas normales*: la igualdad en las condiciones se interpreta como $\to_!^*$, donde $t \to_!^* s$ si $t \to^* s$ y s es una forma normal básica. Es decir, las reglas son de la forma

$$t \to s \Leftarrow t_1 \to_!^* s_1, \ldots, t_n \to_!^* s_n$$

d) *sistemas generalizados*: las condiciones se formulan en un marco matemático general. Es decir, las reglas son de la forma

$$t \to s \Leftarrow P_1, \ldots, P_n$$

donde $P_i, 1 \leq i \leq n$, es una fórmula general escrita en algún lenguaje de primer orden (que puede incluso incluir variables que no aparecen en la cabeza de la regla).

En la clasificación de Bergstra y Klop [15], los sistemas semi-ecuacionales se llaman sistemas de Tipo I, los sistemas naturales son los de Tipo II y los sistemas normales, los de Tipo III_n. Además, en este trabajo, la relación $t \to_!^* s$ se define exigiendo que el término s sea una forma normal básica incluso respecto a la *parte incondicional* del SRTC, es decir, respecto al SRT que se obtiene eliminando todas las condiciones de las reglas. Esto es necesario porque, de no ser así, la relación de reescritura puede no estar bien definida [93].

Habitualmente, en las reglas de reescritura condicionales se impone el mismo requerimiento que en las reglas incondicionales: que no aparezcan variables en la regla que no estén contenidas en la parte izquierda de la cabeza, es decir, que no existan variables extras. El principal motivo es evitar tener que resolver ecuaciones en las condiciones de una regla cuando ésta tiene que ser aplicada, como muestra Kaplan en [84]: si el SRTC contiene una regla como $g(x, y) \to h(z) \Leftarrow f(x, y, z) = s$, para reescribir un término que depende solamente de las variables x e y es necesario encontrar un z tal que $f(x, y, z) = s$. Aunque este requerimiento es bastante restrictivo (muchos problemas se especifican fácilmente usando reglas condicionales que no lo cumplen como, por ejemplo, la especificación de los números de Fibonacci), formatos de reglas más liberales introducen una complicación considerable desde el punto de vista teórico.

Considerando el símbolo $=$ de las condiciones interpretado como $\downarrow$, dado un SRTC R existe una precongruencia más pequeña sobre $\mathcal{T}_\Sigma(X)$ que satisface

$t \to_R t'$ sii existe una ocurrencia u de t, una sustitución σ y
una regla $(l \to r \Leftarrow \bigwedge_{i=1}^{m} u_i = v_i) \in R$
tales que
$t/u = l\sigma, t' = t[r\sigma]_u$ y $\forall i : 1 \leq i \leq m : u_i\sigma \downarrow v_i\sigma$

La relación $\to_R$ puede construirse como la unión de un conjunto infinito de relaciones de reescritura

$$\to_R = \bigcup_{i \geq 0} \{ \stackrel{i}{\to}_R \}$$

definidas como

* $\stackrel{0}{\to}_R$ es la relación vacía
* $t \stackrel{n+1}{\to}_R s$ si y sólo si existe una regla $(l \to r \Leftarrow u_1 = v_1, \ldots, u_m = v_m) \in R$, una ocurrencia no variable u de t, una sustitución σ y m términos t_i tales que $t/u = l\sigma, u_i \stackrel{n}{\to}_R t_i, v_i \stackrel{n}{\to}_R t_i$ y $s = t[r\sigma]_u$

En el caso condicional, la confluencia y la terminación de un SRTC no son suficientes para asegurar la decidibilidad de la relación de reescritura (en [83] puede encontrarse un ejemplo). Por ello, se suele imponer una nueva condición: dado un SRTC R debe cumplirse que

$l\sigma > r\sigma, u_i\sigma, v_i\sigma$, para toda sustitución σ y toda regla $(l \to r \Leftarrow \bigwedge_{i=1}^{m} u_i = v_i) \in R$, siendo $>$ un orden de simplificación. Los SRTCs que la cumplen son llamados *simplificantes* por Kaplan [84, 86]. Nociones relacionadas con el concepto de sistemas simplificantes son los sistemas reductivos [82] y los sistemas decrecientes [39]. De hecho, reductivo $\Rightarrow$ simplificante $\Rightarrow$ decreciente.

Claramente, los sistemas simplificantes son terminantes, como muestra Kaplan en [84]. Para la confluencia, se define una noción adecuada de par crítico condicional (que Kaplan llama par crítico contextual [86]) el cual es la ecuación condicional derivada a partir de dos reglas cuyas partes izquierdas se superponen. El Lema del Par Crítico sigue cumpliéndose para sistemas simplificantes [86]: un SRTC R simplificante es localmente confluente (y, por lo tanto, confluente) si y sólo si $s\sigma \to_R^* w$ y $t\sigma \to_R^* w$ para cada par crítico condicional $s = t \Leftarrow u_1 = v_1, \ldots, u_n = v_n$ y sustitución σ tales que $u_i\sigma =_R v_i\sigma$.

La confluencia también puede asegurarse mediante condiciones sintácticas [15, 18, 41, 118]. Así, por ejemplo, los sistemas semi-ecuacionales y los sistemas normales cuya parte incondicional es, además, ortogonal son confluentes. Algunos métodos semánticos para demostrar la confluencia de los SRTCs pueden encontrarse en [128].

Existen ciertas propiedades, como la propiedad conmutativa y la propiedad asociativa, que no resulta apropiado expresar como reglas de reescritura. En estos casos, lo habitual es mantener tales propiedades como un conjunto de axiomas ecuacionales (no orientados) y plantear la reescritura módulo este conjunto de ecuaciones [9, 81, 124]. En particular, la reescritura módulo un conjunto de ecuaciones $\mathcal{E}$ constituye un marco adecuado para unificar una variedad amplia de modelos de la concurrencia [107].

Para un estudio más profundo de la reescritura de términos, referimos al lector a los trabajos de Klop [92, 93] y Dershowitz [34].

2.1.5 Unificación Universal

Una Σ-teoría ecuacional de Horn $\mathcal{E}$ es un conjunto finito de cláusulas de Horn ecuacionales, de la forma

$$l = r \;\Leftarrow\; e_1, \ldots, e_n$$

con $n \geq 0$ y donde cada e_i es una Σ-ecuación [1].

La Σ-ecuación en el consecuente (cabeza de la cláusula) está implícitamente orientada de izquierda a derecha, mientras que los literales e_i en el antecedente (cuerpo de la cláusula) son Σ-ecuaciones ordinarias (no orientadas). En lo que sigue, omitiremos el prefijo Σ si la signatura queda clara a partir del contexto.

Una teoría ecuacional de Horn $\mathcal{E}$ que cumple $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ y $l \notin V$ puede verse como un SRTC R, donde las reglas son las cabezas de las cláusulas y las condiciones son los respectivos cuerpos. Si todas las cláusulas en $\mathcal{E}$ tienen el cuerpo vacío entonces $\mathcal{E}$ y R se dice que son incondicionales; en caso contrario, se dicen condicionales. La teoría ecuacional $\mathcal{E}$ se dice que es canónica si la relación binaria de reescritura en un paso $\rightarrow_R$ definida por R es noetheriana y confluente.

Definición 2.1.13 $\mathcal{E}$-unificador

Dada una teoría ecuacional $\mathcal{E}$, una sustitución θ es un $\mathcal{E}$-unificador de dos términos t y s si y sólo si $\mathcal{E} \models t\sigma = s\sigma$, o equivalentemente, $t\sigma =_{\mathcal{E}} s\sigma$.

Si $t\sigma =_{\mathcal{E}} s\sigma$ entonces se dice que t y s son $\mathcal{E}$-unificables. Para teorías semi-decidibles, el conjunto $U_{\mathcal{E}}$ de los $\mathcal{E}$-unificadores de dos términos es recursivamente enumerable. En general, es suficiente con obtener una base de $\mathcal{E}$-unificadores a partir de los cuales se puede generar $U_{\mathcal{E}}$ mediante instanciación. Sea $W \subseteq V$ un conjunto de variables y sea $\leq_{\mathcal{E}}$ un orden parcial entre términos definido como $t \leq_{\mathcal{E}} s$ si y sólo si existe una sustitución θ tal que $t =_{\mathcal{E}} s\theta$. Este orden se extiende a sustituciones de la siguiente forma: decimos que σ es una instancia de θ y que θ es más general que σ sobre W, en símbolos $\sigma \leq_{\mathcal{E}} \theta[W]$, si y sólo si existe una sustitución δ tal que $\sigma =_{\mathcal{E}} \theta\delta[W]$, donde $\sigma =_{\mathcal{E}} \theta[W]$ si $\forall x \in W : x\sigma =_{\mathcal{E}} x\theta$. Habitualmente se omite W si es igual a V.

[1] Aunque habitualmente se suele llamar teoría ecuacional al conjunto de ecuaciones que pueden obtenerse a partir de un conjunto $\mathcal{E}$ de cláusulas ecuacionales tomando dichas cláusulas como axiomas y el sistema de deducción de la lógica ecuacional, por abuso hablaremos de la 'teoría ecuacional $\mathcal{E}$' para denotar la teoría axiomatizada por $\mathcal{E}$.

Un conjunto S de sustituciones es un conjunto completo de $\mathcal{E}$-unificadores de dos términos t y s si se satisfacen las siguientes condiciones:

$i)$ $\forall\sigma \in S, D(\sigma) \subseteq \mathcal{V}(t) \cup \mathcal{V}(s)$ y $I(\sigma) \cap W = \emptyset$, siendo W un conjunto de variables tal que $\mathcal{V}(t) \cup \mathcal{V}(s) \subseteq W$ (*protección de* W)

$ii)$ $\forall\sigma \in S$, σ es un $\mathcal{E}$-unificador de t y s (*corrección*)

$iii)$ si θ es un $\mathcal{E}$-unificador de t y s entonces $\exists\sigma \in S$ tal que $\sigma \leq_{\mathcal{E}} \theta[\mathcal{V}(t) \cup \mathcal{V}(s)]$ (*completitud*)

S es un conjunto completo de $\mathcal{E}$-unificadores de máxima generalidad de dos términos t y s si se cumplen las condiciones anteriores junto con el siguiente requerimiento:

$iv)$ $\forall\sigma_1, \sigma_2 \in S$, $\sigma_1 \leq_{\mathcal{E}} \sigma_2[\mathcal{V}(t) \cup \mathcal{V}(s)] \Rightarrow \sigma_1 = \sigma_2$ (*minimalidad*)

Existe siempre un conjunto completo de $\mathcal{E}$-unificadores de dos términos (basta tomar todos los $\mathcal{E}$-unificadores que satisfacen $i)$[2] aunque éste puede ser infinito (por ejemplo, para $a * x = x * a$ en la teoría en la que $*$ es asociativa [129]). Cuando es finito existe siempre un conjunto minimal, obtenido filtrando los elementos redundantes. Sin embargo, esto no es posible en general ya que, en algunas teorías ecuacionales $\mathcal{E}$ de primer orden, existen términos $\mathcal{E}$-unificables para los que no existe un conjunto completo y minimal de $\mathcal{E}$-unificadores, ni siquiera infinito (como demuestran Fages y Huet en [47]). Siekmann y Szabó en [145] analizan diversas teorías ecuacionales desde el punto de vista de la $\mathcal{E}$-unificación, estableciendo una clasificación de las mismas atendiendo a la cardinalidad del conjunto de $\mathcal{E}$-unificadores más generales.

Un procedimiento de $\mathcal{E}$-unificación (es decir, un proceso que resuelve el problema de la unificación de dos términos respecto a una teoría ecuacional $\mathcal{E}$) se dice que es completo si genera un conjunto completo de $\mathcal{E}$-unificadores para cualquier par de términos de entrada. Uno de los

[2] Nótese que esta condición implica que todos los $\mathcal{E}$-unificadores son idempotentes: $\sigma\sigma = \sigma$.

principales problemas de la $\mathcal{E}$-unificación es asegurar la terminación de tales algoritmos. Para la clase de teorías ecuacionales para las que existe un SRT canónico (ver [70]), Fay [48] da un procedimiento de $\mathcal{E}$-unificación basado en la noción de "narrowing", para el que Hullot [71] da un criterio suficiente de terminación. Otros procedimientos de $\mathcal{E}$-unificación han sido desarrollados para otros tipos de teorías, como por ejemplo, para teorías condicionales satisfaciendo diversas restricciones [42, 51, 53, 57, 65, 72, 84]. Algunos problemas tales como la verificación de software, la demostración automática de teoremas o la recuperación de bases de datos, requieren otros algoritmos de $\mathcal{E}$-unificación que trabajan con teorías ecuacionales conmutativas [88], teorías ecuacionales asociativas y conmutativas [1, 46, 146] y teorías ecuacionales asociativas, conmutativas e idempotentes [102].

Algunas técnicas de optimización de los algoritmos de $\mathcal{E}$-unificación tendentes a reducir la talla del árbol de búsqueda sin pérdida de completitud se basan en el uso de símbolos de función irreducibles [64, 72].

Disciplina de Constructores

Sea $\mathcal{E}$ una teoría ecuacional de Horn. Un símbolo de función $f \in \Sigma$ es *irreducible* si y sólo si no existe una cláusula $(l = r \Leftarrow \tilde{B})$ en $\mathcal{E}$ tal que $l \in V$ o f aparece como símbolo de función más externo en el término l. En cualquier otro caso, f es un símbolo de función *definido*.

En teorías en las que se hace la distinción anterior, la signatura queda partida en dos conjuntos disjuntos, $\Sigma = C \uplus F$, donde C es el conjunto de símbolos irreducibles, también llamados *constructores*, y F es el conjunto de los símbolos definidos [52, 69]. La distinción entre constructor y símbolo definido conduce a la distinción entre término y término dato. El primero se construye sobre C, F y V, mientras que el segundo sólo utiliza símbolos de C y V.

Definición 2.1.14 Definición de símbolo de función

El conjunto de cláusulas de la forma $f(t_1, \ldots, t_n) = t \Leftarrow \tilde{B}$ *constituye la definición de* f.

Definición 2.1.15 Teoría ecuacional con disciplina de constructores

Una teoría ecuacional de Horn $\mathcal{E}$ es una teoría con disciplina de constructores si todas sus cláusulas son de la forma $f(t_1, \ldots, t_n) = t \Leftarrow \tilde{B}$, donde $f \in F$ y ningún símbolo de función definido ocurre en $t_1, \ldots, t_n$.

Cuando una teoría ecuacional satisface la disciplina de constructores, la propiedad de confluencia queda asegurada si se imponen además las siguientes restricciones sintácticas [61, 119]:

a) Linealidad por la izquierda.

b) Salida definida: Toda variable en la parte derecha de la cabeza de la cláusula debe ocurrir en la parte izquierda de la cabeza de la misma.

c) No ambigüedad: Dadas dos cláusulas que definen el mismo símbolo de función $f \in F$,

$$\begin{array}{l} f(t_1, \ldots, t_n) = r \Leftarrow \tilde{B} \\ f(t'_1, \ldots, t'_n) = r' \Leftarrow \tilde{B}' \end{array}$$

debe cumplirse alguna de las siguientes condiciones[3]:

c.i) No superposición: $f(t_1, \ldots, t_n)$ y $f(t'_1, \ldots, t'_n)$ no son unificables.

c.ii) Partes derechas básicas: r y r' son términos básicos idénticos.

c.iii) Fusión de partes derechas: $f(t_1, \ldots, t_n)$ y $f(t'_1, \ldots, t'_n)$ unifican con mgu σ tal que $r\sigma$ y $r'\sigma$ son términos idénticos.

c.iv) Incompatibilidad de los cuerpos: $f(t_1, \ldots, t_n)$ y $f(t'_1, \ldots, t'_n)$ unifican con mgu σ tal que $\tilde{B}\sigma$ y $\tilde{B}'\sigma$ son incompatibles.

[3]Si es necesario se consideran variantes de las cláusulas para que no tengan variables comunes.

La incompatibilidad (condición *c.iv*) es una propiedad decidible de pares $\tilde{B}$ y $\tilde{B}'$ de sistemas de ecuaciones que garantiza que $\tilde{B}$ y $\tilde{B}'$ no pueden evaluarse a cierto simultáneamente.

Definición 2.1.16 Teoría suficientemente completa

Una teoría ecuacional $\mathcal{E}$ es suficientemente completa para el conjunto de constructores C si para cada término básico t existe un término básico $s \in \mathcal{T}(C)$ tal que $s =_{\mathcal{E}} t$.

La completitud suficiente es, en general, una propiedad indecidible. Sin embargo, bajo ciertas asunciones, se puede mostrar que una teoría ecuacional $\mathcal{E}$ es suficientemente completa para el conjunto de símbolos de función constructores C si cada término de la forma $f(x_1, \ldots, x_n)$ con $f \in F$ es reducible básico en el SRT correspondiente a $\mathcal{E}$.

Teorías completamente definidas

Definición 2.1.17 Símbolo de función completamente definido *Un símbolo de función f es completamente definido ("everywhere defined") sobre su dominio si no aparece en ningún término básico en forma normal.*

Definición 2.1.18 Teoría ecuacional completamente definida

Una teoría ecuacional de Horn $\mathcal{E}$ es completamente definida si cada símbolo de función $f \in F$ es completamente definido.

En teorías ecuacionales con disciplina de constructores y completamente definidas, todo término básico se reduce a un término dato, es decir, el conjunto de los términos básicos en forma normal coincide con el conjunto de los términos sobre C ($\mathcal{T}(C)$). En [45, 52, 69] se proponen condiciones suficientes que garantizan esta propiedad, como el *principio de definición* de Huet-Hullot [69] que se expresa como la conjunción de las siguientes propiedades:

a) para cada término $t \in \mathcal{T}(C \cup F)$ existe un término $s \in \mathcal{T}(C)$ tal que $t =_{\mathcal{E}} s$

b) para cada par de términos $t, s \in \mathcal{T}(C)$ se cumple que $t =_{\mathcal{E}} s$ si y sólo si $t = s$

Para estas teorías, puede definirse una semántica declarativa tomando como dominio $\mathcal{T}(C)$ ya que el álgebra inicial de una teoría $\mathcal{E}$ que satisface estas propiedades es isomorfa al cociente de $\mathcal{T}(\Sigma)$ por la congruencia definida por $\mathcal{E}$ [60].

2.2 Teorías ecuacionales normales

En esta sección definimos la clase de las teorías ecuacionales normales. Informalmente, una teoría ecuacional normal es una teoría ecuacional cuyas cláusulas pueden contener negación en el cuerpo. Formalmente:

Definición 2.2.1 Cláusula ecuacional normal

Una cláusula ecuacional normal es una cláusula de la forma

$$t = s \Leftarrow e_1, \ldots, e_m, d_1, \ldots, d_n.$$

donde la ecuación de la cabeza está implícitamente orientada de izquierda a derecha, $e_1, \ldots, e_m$ son ecuaciones no orientadas, $d_1, \ldots, d_n$ son disecuaciones no orientadas y $m, n \geq 0$.

Definición 2.2.2 Teoría ecuacional normal

Una teoría ecuacional normal $\mathcal{E}^{\sim}$ es un conjunto finito de cláusulas ecuacionales normales.

Como fácilmente se desprende de las Definiciones 2.2.1 y 2.2.2, cuando los cuerpos de las cláusulas de una teoría ecuacional normal $\mathcal{E}^{\sim}$ no contienen disecuaciones, $\mathcal{E}^{\sim}$ es una teoría ecuacional de Horn.

Definición 2.2.3 Objetivo normal

Un objetivo normal es una cláusula normal sin cabeza.

Ejemplo 1 *Como ejemplo de teoría ecuacional normal, consideremos la siguiente teoría tomada de Padawitz* ([123]), *la cual define los números racionales no negativos como pares de números naturales. Los símbolos de función* $/, +$ *y* $*$ *denotan, respectivamente, la división, suma y resta de racionales, mientras que* $+_{\mathrm{N}}$ *y* $*_{\mathrm{N}}$ *denotan la suma y multiplicación de naturales.*

$$\begin{array}{ll} \mathcal{E}_1^{\sim} = \{ & 0/m = 0 \Leftarrow m \neq 0. \\ & m/c = n/d \Leftarrow m *_{\mathrm{N}} d = c *_{\mathrm{N}} n, c \neq 0, d \neq 0. \\ & (m/c) + (n/d) = ((m *_{\mathrm{N}} d) +_{\mathrm{N}} (c *_{\mathrm{N}} n))/(c *_{\mathrm{N}} d). \\ & (m/c) * (n/d) = (m *_{\mathrm{N}} n)/(c *_{\mathrm{N}} d). \} \end{array}$$

2.2.1 Teorías ecuacionales normales y reescritura

Al igual que con las teorías ecuacionales de Horn, una teoría ecuacional normal $\mathcal{E}^{\sim}$ puede verse como un sistema de reescritura de términos condicional, donde las reglas son las cabezas y las condiciones son los respectivos cuerpos. Como hemos mencionado en la introducción, el estudio de tales sistemas de reescritura (los cuales incluyen disecuaciones en las condiciones de las reglas) ha sido abordado por Kaplan [85] entre otros. En este trabajo se generaliza la clase de reglas de reescritura bajo consideración, introduciendo los llamados *sistemas de reescritura de términos condicionales positivos/negativos (SRTC P/N)* definidos como conjuntos finitos de fórmulas con la forma:

$$l \to r \Leftarrow (\bigwedge_{i=1}^{m} u_i = v_i) \wedge (\bigwedge_{j=1}^{n} u'_j \neq v'_j)$$

que satisfacen la condición siguiente:

$$Var(l) \supseteq Var(r), Var(u_i), Var(v_i), Var(u'_j), Var(v'_j)$$

Para poder dar una semántica operacional por reescritura es necesario un nuevo requerimiento: los SRTC P/N deben ser *simplificantes*, es decir, debe existir un orden de reducción '$>$' tal que, para cualquier sustitución básica σ y para cualquier regla del sistema, se cumple que:

$$l\sigma > r\sigma, u_i\sigma, v_i\sigma, u'_j\sigma, v'_j\sigma \tag{2.1}$$

Desde un punto de vista operacional, uno de los principales inconvenientes de los sistemas P/N reside en el hecho de que no existe una precongruencia más fina, es decir, una relación de reescritura mínima que satisfaga las reglas. Por ejemplo [85], para el sistema:

$$S_1 : \{a \to c \Leftarrow b \neq c\}$$

las siguientes dos relaciones:

$$R_1 : \{a \to c\} \text{ y } \ R_2 : \{b \to c\}$$

son minimales, satisfacen S_1 pero son incomparables entre sí. Sin embargo, cuando se especifica la regla S_1, se está añadiendo cierta información desde un punto de vista operacional: el sistema S_1 deberá interpretarse como R_1 y no como R_2 ya que no existe ninguna regla en el sistema que satisfaga $b = c$. Esta información ha sido tenida en cuenta por Kaplan en la definición de la relación de reescritura asociada a un sistema P/N. Así, si el sistema es simplificante existe una precongruencia única y bien definida $\to_R$ sobre $\mathcal{T}(\Sigma)$, tal que

$t \to_R t'$ sii existe una ocurrencia u de t, una sustitución σ y una regla $(v \to w \Leftarrow (\bigwedge_{i=1}^m u_i = v_i) \wedge (\bigwedge_{j=1}^n u'_j \neq v'_j)) \in R$ tales que
$t/u = v\sigma, t' = t[w\sigma]_u, \forall i : 1 \leq i \leq m : (u_i\sigma \downarrow v_i\sigma)$ y
$\forall j : 1 \leq j \leq n : (u'_j\sigma \not\downarrow v'_j\sigma)$

Según esta definición, la relación asociada al sistema S_1 anterior es R_1 pues la relación R_2 no cumple la condición: "$t \to t'$ sii existe una ocurrencia ..." con $t = b$ y $t' = c$.

Kaplan [85] muestra que para sistemas P/N simplificantes existe una construcción para la relación $\to_R$ que puede definirse como el límite de la siguiente secuencia de relaciones:

$$
\begin{array}{ll}
\to_0 = \emptyset & \\
\to_{k+1} = \{(t,t') : & \text{existe una ocurrencia } u \text{ de } t, \\
& \text{una sustitución } \sigma, \text{ y una regla de } R \\
& v \to w \Leftarrow (\bigwedge_{i=1}^{m} u_i = v_i) \wedge (\bigwedge_{j=1}^{n} u'_j = v'_j) \\
& \text{que satisfacen:} \\
& t/u = v\sigma, t' = t[w\sigma]_u, \\
& \forall i : 1 \leq i \leq m : NF_{\to_k}(u_i\sigma) \cap NF_{\to_k}(v_i\sigma) \neq \emptyset \\
& \forall j : 1 \leq j \leq n : NF_{\to_k}(u'_j\sigma) \cap NF_{\to_k}(v'_j\sigma) = \emptyset\}
\end{array}
$$

donde $NF_{\to_k}(t)$ es el conjunto de la formas normales de t en $\to_k$.

Para poder dar esta construcción de la relación de reescritura es imprescindible la hipótesis de que el sistema es simplificante (condición 2.1). De no ser así, puede que no exista el límite de la secuencia, como se muestra en el siguiente ejemplo [85]:

$$
\begin{array}{ll}
S_2 = \{ & c \to d \Leftarrow a \neq b \\
& a \to b \Leftarrow c \neq d\}
\end{array}
$$

Entonces,

$$
\begin{array}{ll}
\to_0 = \emptyset, & \to_1 = \{c \to d, a \to b\}, \\
\to_2 = \emptyset, & \to_3 = \{c \to d, a \to b\}, \\
\ldots & \ldots
\end{array}
$$

2.2.2 Terminación

La terminación y confluencia de los SRTCs P/N pueden demostrarse extendiendo las técnicas usadas para los SRTCs. Veamos primero cuáles son las condiciones bajo las que un SRTC P/N es terminante.

Al igual que para el caso positivo, la terminación de los sistemas P/N depende de la terminación de la evaluación de las condiciones de las reglas. El método estándar para probar la terminación de un sistema de reescritura se basa en el uso de órdenes de simplificación. Para el caso condicional positivo, Kaplan ha demostrado [86] que un SRTC es terminante si la condición de una regla es más 'simple' (en el sentido definido por un orden de simplificación) que la cabeza de la misma. La extensión de este resultado al caso de sistemas que incluyen negación en las condiciones es directo, como mostramos a continuación.

Definición 2.2.4 *Sea R un SRTC P/N, t un término, $u \in O(t)$, σ una substitución y $(l \to r \Leftarrow u_1 = v_1, \ldots, u_m = v_m, u'_1 \neq v'_1, \ldots, u'_n \neq v'_n)$ una regla de R. Decimos que $t \hookrightarrow_R t'$ si $t/u = l\sigma$ y $t' = u_i\sigma$ o $t' = v_i\sigma$ o $t' = u'_j\sigma$ o $t' = v'_j\sigma$ con $1 \leq i \leq m, 1 \leq j \leq n$.*

La definición anterior viene a decir que para evaluar t usando una regla de R debemos evaluar también t'. Por lo tanto, para demostrar que R es terminante, es suficiente demostrar la terminación de $\to_R \cup \hookrightarrow_R$.

Proposición 2.2.5 *Sea R un SRTC P/N y sea $>$ un orden de simplificación sobre el conjunto de términos $\mathcal{T}(\Sigma)$. Si R es simplificante (con respecto a $>$) entonces $\to_R \cup \hookrightarrow_R$ es terminante.*

DEMOSTRACIÓN. Por reducción al absurdo.

Si R no es terminante entonces existe una secuencia infinita de términos, $t_1, \ldots, t_k, \ldots, k > 0$, tal que $\forall l : 1 \leq l < k : t_l \to_R t_{l+1}$ o $t_l \hookrightarrow_R t_{l+1}$. Sea $(l \to r \Leftarrow u_1 = v_1, \ldots, u_m = v_m, u'_1 \neq v'_1, \ldots, u'_n \neq v'_n) \in R$, t un término, $w \in O(t)$ y σ una sustitución. Entonces, si $t/w = l\sigma$ entonces $t > l\sigma$ ya que $>$ es un orden de simplificación y, por tanto, es cerrado bajo subtérmino. De aquí que:

* Si $t_l \to_R t_{l+1}$, entonces $t_{l+1} = t_l[r\sigma]_w$, $l\sigma > r\sigma$ por ser R simplificante y $t_l > t_{l+1}$ porque $>$ es compatible.

* Si $t_l \hookrightarrow_R t_{l+1}$, donde t_{l+1} es uno de los siguientes términos, $u_i\sigma, v_i\sigma, u'_j\sigma, v'_j\sigma, 1 \leq i \leq m, 1 \leq j \leq n$, entonces $l\sigma > u_i\sigma, v_i\sigma,$
$u'_j\sigma, v'_j\sigma$ poque R es simplificante y $t_l > l\sigma$ porque $>$ es cerrado bajo subtérmino. Por lo tanto, $t_l > t_{l+1}$.

De aquí que $\forall l : 1 \leq l < k : t_l > t_{l+1}$, lo cual contradice el hecho de que $>$ es un orden bien fundado.

Por lo tanto concluimos que $\to_R \cup \hookrightarrow_R$ es terminante.

2.2.3 Confluencia

La cuestión de la confluencia de una teoría ecuacional normal (o, alternativamente, del SRTC P/N asociado) es abordada por Kaplan proporcionando un criterio 'a la Knuth-Bendix' a partir de los pares críticos contextuales existentes en el sistema. Este resultado extiende el criterio de Knuth-Bendix para sistemas ecuacionales incondicionales y condicionales.

Es posible dar una caracterización sintáctica de la confluencia de una teoría ecuacional normal siguiendo el criterio establecido por O'Donnell [122] para ecuaciones condicionales cuya parte condicional incluye predicados arbitrarios. O'Donnell demostró que, bajo ciertas restricciones sintácticas, un conjunto de ecuaciones condicionales es confluente si las condiciones cumplen la propiedad de estabilidad. Intuitivamente, estabilidad significa que si una condición $\tilde{Q}$ en una ecuación condicional $(l = r \Leftarrow \tilde{Q})$ se cumple para una cierta instanciación de las variables y existe una instancia de una ecuación $(l' = r' \Leftarrow \tilde{Q}')$ tal que $\tilde{Q}'$ es cierto y un subtérmino de $\tilde{Q}$ es igual a l', entonces $\tilde{Q}$ sigue cumpliéndose si reemplazamos este subtérmino por r'.

Teorema 2.2.6 [122] *Sea $\mathcal{E}$ un conjunto de ecuaciones de la forma $l = r \Leftarrow \tilde{Q}$ tal que la parte incondicional de $\mathcal{E}$ es ortogonal y todas las condiciones en $\tilde{Q}$ son estables. Entonces, $\mathcal{E}$ es confluente.*

Según el teorema anterior, la demostración de la confluencia de una teoría ecuacional normal $\mathcal{E}^{\sim}$, cuya parte incondicional sea ortogonal, se basa en la demostración de la estabilidad de los símbolos de igualdad y desigualdad que aparecen en las condiciones de las cláusulas de $\mathcal{E}^{\sim}$.

El corolario siguiente establece el resultado de confluencia de una teoría ecuacional normal a partir de la estabilidad de las condiciones.

Corolario 2.2.7 *Sea $\mathcal{E}^{\sim}$ una teoría ecuacional normal cuya parte incondicional es ortogonal. Entonces, $\mathcal{E}^{\sim}$ es confluente.*

DEMOSTRACIÓN. La confluencia de $\mathcal{E}^{\sim}$ depende de la estabilidad de los símbolos $=$ y $\neq$ en las condiciones de las cláusulas. Consideremos una instancia de una cláusula de $\mathcal{E}^{\sim}$, $t = s \Leftarrow \tilde{Q}$, cuya condición $\tilde{Q}$ incluye los símbolos de igualdad y desigualdad. La igualdad es un predicado estable [15]. Para demostrar la estabilidad de la desigualdad, supongamos que la condición $\tilde{Q}$ es cierta, que existe una disecuación d en $\tilde{Q}$ y que existe una ocurrencia no variable u de un subtérmino en d, digamos t_j, tal que $t_j/u = l$, siendo $l = r \Leftarrow \tilde{Q}'$ una instancia de una cláusula de $\mathcal{E}^{\sim}$. Asimismo, podemos suponer, sin pérdida de generalidad, que $u = \Lambda$. Sea d, por ejemplo, de la forma $f(t_1, \ldots, t_n) \neq s$. Entonces, basta demostrar la propiedad:

$$\begin{array}{ll} l = r \ \wedge \ t_j = l & \wedge \ f(t_1, \ldots, t_j, \ldots, t_n) \neq s \Rightarrow \\ & f(t_1, \ldots, r, \ldots, t_n) \neq s \end{array}$$

Ahora bien, por las propiedades de transitividad y sustitutividad de funciones que cumple el predicado de igualdad, sabemos que:

$$l = r \ \wedge \ t_j = l \Rightarrow f(t_1, \ldots, t_j, \ldots, t_n) = f(t_1, \ldots, r, \ldots, t_n) \quad (2.2)$$

Por lo tanto,

$$\begin{array}{ll} l = r \ \wedge \ t_j = l & \wedge \ f(t_1, \ldots, t_j, \ldots, t_n) \neq s \stackrel{2.2}{\Rightarrow} \\ & f(t_1, \ldots, t_j, \ldots, t_n) = f(t_1, \ldots, r, \ldots, t_n) \wedge \\ & f(t_1, \ldots, t_j, \ldots, t_n) \neq s \end{array}$$

Aplicando a la expresión resultante del paso anterior la propiedad transitiva para el predicado desigualdad, que se enuncia como:

$$p \neq q \ \wedge \ q = r \Rightarrow p \neq r \quad (2.3)$$

obtenemos:

$$\begin{array}{ll} f(t_1, \ldots, t_j, \ldots, t_n) & = f(t_1, \ldots, r, \ldots, t_n) \wedge \\ & f(t_1, \ldots, t_j, \ldots, t_n) \neq s \stackrel{2.3}{\Rightarrow} \\ & f(t_1, \ldots, r, \ldots, t_n) \neq s \end{array}$$

Lo cual demuestra que el predicado desigualdad es también estable.

Finalmente, por el Teorema 2.2.6, la teoría ecuacional normal $\mathcal{E}^{\sim}$ es confluente.

La Proposición 2.2.5 y el Corolario 2.2.7 dan condiciones suficientes para garantizar la canonicidad de una teoría ecuacional normal.

Podría formularse una tercera caracterización de la confluencia de una teoría ecuacional normal imponiendo ciertas condiciones sintácticas, de forma análoga a como se hace para teorías ecuacionales de Horn. Nótese que las condiciones a), b), $c.i$), $c.ii$) y $c.iii$) de la Sección 2.1.5 formuladas para teorías ecuacionales sin negación sólo hacen referencia a la cabeza de la cláusula ecuacional y la condición $c.iv$) sólo requiere que los cuerpos de las cláusulas instanciados sean incompatibles lo que incluye la posibilidad de que aparezca la negación en tales cuerpos.

Conjetura 2.2.8 *Sea $\mathcal{E}^{\sim}$ una teoría ecuacional normal con disciplina de constructores. Si $\mathcal{E}^{\sim}$ satisface las propiedades de linealidad por la izquierda, salida definida y no ambigüedad entonces $\mathcal{E}^{\sim}$ es confluente.*

Ejemplo 2 Teoría ecuacional normal confluente

La siguiente teoría ecuacional normal define una función, denotada por elim__, que elimina un elemento de un conjunto. Los elementos del conjunto no están repetidos y supongamos que son de un cierto dominio D. Las operaciones constructoras de datos conjunto son las habituales: empty (conjunto vacío) e ins__ (añadir un elemento a un conjunto). En lo que sigue, X e Y son variables que denotan elementos de D y S denota un conjunto.

$$\begin{array}{ll} elim(X, empty) = empty. & \\ elim(X, ins(S,Y)) = S & \Leftarrow X = Y. \\ elim(X, ins(S,Y)) = ins(elim(X,S),Y) & \Leftarrow X \neq Y. \end{array}$$

Esta teoría satisface todas las condiciones sintácticas antes mencionadas ya que la parte izquierda de la cabeza de la primera cláusula no se superpone con ninguna de las partes izquierdas de las cabezas de las cláusulas restantes, y las dos últimas cláusulas cumplen la condición c.iv) ya que sus condiciones, $X = Y$ y $X \neq Y$, son incompatibles.

Capítulo 3

Semántica operacional

En este capítulo presentamos nuestra aproximación para resolver objetivos normales en una teoría ecuacional normal. El mecanismo operacional que definimos es una extensión del algoritmo de "narrowing" condicional para resolver de manera constructiva los subobjetivos negados. De esta forma, tanto ecuaciones como disecuaciones gozan del mismo "status" y se manejan de forma similar, siguiendo la idea que se propone en programación lógica de que se puede obtener una forma simplificada de una fórmula negada a partir de su versión positiva. En concreto, el algoritmo que presentamos adapta al caso ecuacional la propuesta de Stuckey [147] para la negación constructiva en programación lógica con restricciones. El método que describimos a continuación es correcto y completo para teorías normales canónicas, asumiendo que tales teorías son completamente definidas y satisfacen la disciplina de constructores (ver Sección 2.1.5), como mostraremos en el Capítulo 5. En el Anexo A presentamos un sistema experimental que implementa el método de negación constructiva.

El capítulo está organizado en cuatro secciones. En la primera sección se hace una breve revisión del algoritmo de "narrowing" condicional. La segunda sección describe la *regla de negación constructiva ecuacional* para tratar constructivamente objetivos negados, formulándose, además, un mecanismo operacional basado en "narrowing" para resolver conjunciones de ecuaciones y disecuaciones en una teoría ecuacional normal. La sección tercera define un cálculo, llamado $\mathcal{ICN}cn$, basado en el procedimiento de "innermost narrowing" (un procedimiento de $\mathcal{E}$-unificación correcto y completo para teorías ecuacionales completamente definidas, con disciplina de constructores y canónicas) y que sigue el método descrito en la segunda sección. Finalmente, la última sección presenta algunas optimizaciones para el cálculo.

3.1 El algoritmo de "narrowing" condicional

En esta sección revisamos brevemente el concepto de "narrowing" y su formulación en el caso condicional. Comenzaremos dando una explicación intuitiva de por qué el algoritmo de "narrowing" se llama precisamente así [93].

Aunque el algoritmo de "narrowing" fue definido originalmente como una adaptación de la regla de inferencia de la paramodulación en el contexto de la demostración automática de teoremas, posteriormente fue usado como procedimiento de unificación universal para resolver ecuaciones en una teoría ecuacional. Si $\mathcal{E}$ es una teoría ecuacional, la expresión $[\![t = s]\!]_{\mathcal{E}}$ denota el conjunto de soluciones de la ecuación $t = s$ en $\mathcal{E}$, es decir, $\{\sigma : \mathcal{E} \models t\sigma = s\sigma\}$, donde σ es una sustitución. Sea SUB el conjunto de todas las sustituciones y sea $\sigma\mathcal{X} = \{\sigma\tau : \tau \in \mathcal{X}\}$ si $\mathcal{X} \subseteq SUB$. Ahora bien, ya que para cada σ se cumple que $\sigma[\![t\sigma = s\sigma]\!]_{\mathcal{E}} \subseteq [\![t = s]\!]_{\mathcal{E}}$, existe la posibilidad, en principio, de determinar $[\![t = s]\!]_{\mathcal{E}}$ realizando los dos tipos de acciones siguientes: primero, calcular una componente σ del conjunto de sustituciones y restringir $[\![t = s]\!]_{\mathcal{E}}$ a $\sigma[\![t\sigma = s\sigma]\!]_{\mathcal{E}}$; seguidamente, aplicar una ecuación de $\mathcal{E}$ a una de las partes de la ecuación $t\sigma = s\sigma$. Claramente, este tipo de acción preserva el conjunto de soluciones. Realizando de forma iterativa estos dos pasos, alternando pasos del primer tipo con pasos del segundo tipo, podemos alcanzar el conjunto de soluciones de una ecuación trivial $r = r$,

$$
\begin{array}{l}
[\![t = s]\!]_{\mathcal{E}} \supseteq \sigma[\![t\sigma = s\sigma]\!]_{\mathcal{E}} = \sigma[\![r = s\sigma]\!]_{\mathcal{E}} \supseteq \\
\sigma\sigma'[\![r\sigma' = s\sigma\sigma']\!]_{\mathcal{E}} = \ldots \supseteq \ldots \supseteq \sigma\sigma' \ldots \sigma^{(n)}[\![r = r]\!]_{\mathcal{E}}
\end{array}
$$

El último conjunto de soluciones, $\sigma\sigma' \ldots \sigma^{(n)}[\![r = r]\!]_{\mathcal{E}}$, de esta secuencia de restricciones o secuencia de "narrowing" tiene como elemento más general la sustitución $\sigma\sigma' \ldots \sigma^{(n)}$. A la palabra "narrowing" también se le ha dado un contenido formal: denota un cierto método, basado en la reescritura de términos, para determinar, tal y como se ha descrito, el conjunto $[\![t = s]\!]_{\mathcal{E}}$. Un paso de "narrowing" combina un paso del primer tipo con uno del segundo. De hecho, la relación de "narrowing" se define sobre términos en vez de sobre ecuaciones, como en la definición siguiente en la que suponemos que R es un SRT equivalente a $\mathcal{E}$, es decir, $=_R$ coincide con $=_{\mathcal{E}}$.

Definición 3.1.1 Relación de "narrowing"

Sea t un término y u una ocurrencia no variable de t. Decimos que t se reduce por "narrowing" a t' a la ocurrencia u usando la regla de reescritura $r : t_1 \to t_2$ de R via $\sigma = mgu(t/u, t_1)$ (las variables de r se renombran, si es necesario, para que no haya coincidencia con las variables de t) si $t' = t[t_2]_u\sigma$, y escribimos $t \quad {}_{u,r,\sigma}\; t'$. La relación ${}_{u,r,\sigma}$ se denomina "narrowing".

Los subíndices u y r en la relación ${}_{u,r,\sigma}$ se omiten a menudo, pero no σ. Obsérvese que de la anterior definición se deduce que el "narrowing" es una generalización de la reescritura ya que cualquier paso de reescritura sobre términos sin variables también es un paso de "narrowing".

Definición 3.1.2 Redex de "narrowing"

Un término s se llama un redex de "narrowing" (expresión reducible) si y sólo si existe una nueva variante $(l \to r \Leftarrow \tilde{B})$ de una regla de reescritura en R y una sustitución σ tales que $s\sigma = l\sigma$.

La transformación de "narrowing" definida sobre términos se extiende de forma natural a ecuaciones: si $t \quad {}_{\sigma}\; t'$ entonces $t = s \Rightarrow_\sigma t' = s\sigma$, donde la relación $\Rightarrow_\sigma$ se llama relación de "narrowing" entre ecuaciones.

Fay [48] fue el primero en proponer una estrategia de "narrowing" normalizante para teorías canónicas en la que, tras cada paso de "narrowing", el término resultante se reescribe hasta su forma normal. Habitualmente, el algoritmo de "narrowing" conlleva una gran cantidad de búsqueda redundante, por lo que se han propuesto distintos refinamientos tendentes a reducir el espacio de búsqueda. Así, Hullot [71] presentó la estrategia de "narrowing" básico, en la que los pasos de "narrowing" están restringidos a los subtérminos que no han sido introducidos previamente por instanciación. Nutt, Réty y Smolka [121] definieron un sistema que combina "narrowing" básico y normalizante, mostrando cómo usar este sistema para resolver ecuaciones en el álgebra inicial especificada por un SRT terminante y confluente básico. El algoritmo de "narrowing" fue extendido a teorías condicionales por Kaplan [86], Hussmann [72] y Dershowitz y Plaisted [42, 43], entre otros. Giovannetti y Moiso [58] mostraron cómo la presencia de variables extras en las condiciones de las cláusulas puede conducir a la pérdida de la completitud del "narrowing". Hölldobler [64] fue uno de los primeros en realizar un análisis, extenso y sistemático, de versiones distintas del "narrowing" condicional para SRTCs sin variables extras. También nosotros consideraremos teorías ecuacionales sin variables extras. Otros refinamientos son el "narrowing" perezoso presentado por Reddy [136], el algoritmo de "narrowing" con estrategia "innermost" que presenta Fribourg [53] o el "narrowing" con estrategia de selección introducido por Bosco, Giovannetti y Moiso [17] y en el que, en cada paso, se selecciona un subtérmino y se olvidan todas las otras elecciones posibles.

Recientemente, Middeldorp y Hamoen [109, 110] han analizado los resultados de completitud de diversas formulaciones del "narrowing", tanto incondicional como condicional, para SRTs y SRTCs satisfaciendo diversas restricciones, refutando algunos de los resultados existentes hasta el momento, como por ejemplo, el referente a la completitud del "narrowing" condicional básico establecido por Hölldobler [64] para SRTCs canónicos sin variables extras en las reglas.

El algoritmo de "narrowing" condicional puede formularse, para un sistema de ecuaciones, de la siguiente forma, en la que seguimos la nomenclatura usada por Hussmann en [72].

Un objetivo es una expresión de la forma g *with* θ donde g es un sistema de ecuaciones y θ es una sustitución, la cual se considera restringida implícitamente a las variables del objetivo inicial. Para resolver un sistema inicial g_0 el algoritmo comienza con el objetivo g_0 *with* ϵ e intenta derivar nuevos objetivos (recordando las sustituciones aplicadas en las partes "*with*") hasta que no queden ecuaciones por resolver, es decir, hasta alcanzar un objetivo terminal de la forma $\emptyset$ *with* θ. Cada sustitución θ en un objetivo terminal (restringida a las variables del objetivo inicial) es un $\mathcal{E}$-unificador del sistema g_0 inicial (una respuesta computada) el cual se llama, a menudo, solución. A continuación definimos un paso de derivación de "narrowing" condicional.

Definición 3.1.3 Un paso de la relación de "narrowing" condicional

Dada una teoría ecuacional de Horn $\mathcal{E}$, los hijos de un objetivo g with θ en un árbol de derivaciones de "narrowing" condicional vienen dados por:

a) si g unifica sintácticamente con mgu σ, entonces g with θ tiene un único hijo de la forma

$$\emptyset \, with \, \theta\sigma$$

b) si $e \in g$ y $u \in \bar{O}(e)$ entonces existe un hijo de la forma

$$((g \setminus \{e\}) \cup \{e[r_k]_u\} \cup \tilde{B}_k)\sigma \, with \, \theta\sigma$$

por cada claúsula (estandarizada aparte, es decir, en el conjunto de variables en la claúsula no hay ninguna variable encontrada previamente en la computación [1]) $(l_k = r_k \Leftarrow \tilde{B}_k)$ de $\mathcal{E}$ tal que $\sigma = mgu(e/u, l_k)$. $\setminus$ denota la diferencia de conjuntos.

El mecanismo computacional descrito anteriormente utiliza una estrategia de búsqueda en amplitud ("breadth-first") para visitar el árbol

[1]Middeldorp y Hamoen [109, 110] han refutado la completitud del "narrowing" cuando se usan otras definiciones diferentes para el concepto de estandarización aparte.

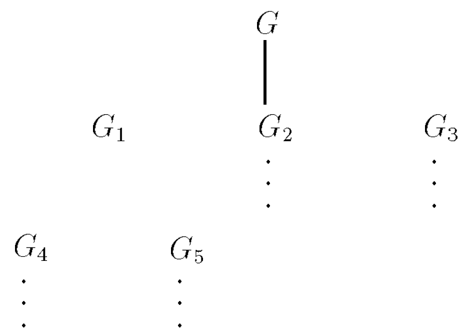

Figura 3.1: *Arbol de derivaciones para un objetivo G.*

de derivaciones para un objetivo. Las hojas de este árbol representan las soluciones del objetivo inicial (la raíz del árbol).

A continuación definimos el concepto de *frontera* de un árbol de derivaciones, que será crucial para la formulación de nuestra regla de negación constructiva ecuacional.

Definición 3.1.4 Frontera de un árbol de derivaciones

Una frontera de un árbol de derivaciones para un objetivo G, o simplemente frontera de G, es el conjunto de nodos a la profundidad uno en el árbol.

En la Figura 3.1, $\{G_1, G_2, G_3\}$ es la frontera del objetivo G y $\{G_4, G_5\}$ es la frontera de G_1.

3.2 El método de Negación Constructiva Ecuacional

El mecanismo operacional que definimos en esta sección está basado en el algoritmo de "narrowing" condicional, tal y como se ha formulado en la Sección 3.1. Nuestro algoritmo resuelve de manera constructiva el problema de la negación en teorías ecuacionales, es decir, es capaz de encontrar las respuestas a una disecuación con respecto a una teoría ecuacional normal $\mathcal{E}^{\sim}$. Intuitivamente, la idea consiste en ir resolviendo parcialmente cada disecuación a partir de la resolución parcial de su correspondiente versión positiva: cuando una disecuación se selecciona para ser resuelta, se realiza una subderivación de "narrowing" para su versión positiva en la que se genera una frontera; la negación de esta frontera es justo una frontera del árbol de derivaciones para la disecuación. Debido a este proceso de negación de la frontera, el algoritmo introduce negación y cuantificación en la respuesta computada. Por lo tanto, la respuesta computada ya no es una sustitución (como ocurre en el caso positivo) sino un sistema complejo de *fórmulas elementales*, que definimos de la forma siguiente:

Definición 3.2.1 Fórmula elemental

Una fórmula elemental es bien una ecuación (posiblemente cuantificada) ($x = t$ o $\forall \tilde{Z}\ x = t$), bien una disecuación (posiblemente cuantificada) ($x \neq t$ o $\forall \tilde{Z}\ x \neq t$), bien $\emptyset$ o bien $\sim \emptyset$ [2]*, donde x es una variable, t es un término distinto de x y $\tilde{Z}$ son algunas de las variables de ($x = t$) o de ($x \neq t$) (posiblemente ninguna).*

Definición 3.2.2 Sistema complejo

Un sistema complejo es una conjunción (posiblemente negada y/o cuantificada) de fórmulas elementales.

NOTA. En lo que sigue, una conjunción $c_1 \wedge \ldots \wedge c_n$ se denotará como $c_1, \ldots, c_n$.

Los siguientes sistemas son ejemplos de sistemas complejos:

$$x = a, x \neq b$$
$$x \neq 0, \forall x' \sim (x = 0, x = s(x'))$$
$$\sim (x = 0, \forall x' \sim (x = s(x')))$$

[2] La fórmula elemental $\emptyset$ es equivalente al valor 'cierto', mientras que la fórmula $\sim \emptyset$ es equivalente al valor 'falso'.

Definición 3.2.3 Respuesta

Una respuesta para un objetivo G es un sistema complejo que afecta a las variables de G y posiblemente a otras variables.

Análogamente, los objetivos derivados adquieren, a lo largo de la computación, una forma que también incluye negación y cuantificación. Así pues, extendemos la sintaxis de los objetivos normales.

Definición 3.2.4 Subobjetivo extendido

Un subobjetivo extendido se define recursivamente como:

* *una ecuación, o*
* *una expresión, $\sim \exists \tilde{Z}(g_1, \ldots, g_n \quad E)$, donde cada $g_i, 1 \leq i \leq n$, es un subobjetivo extendido, E es un sistema complejo y $\tilde{Z}$ denota algunas de las variables que ocurren en $(g_1, \ldots, g_n \quad E)$.*

NOTA. En lo que sigue, usamos la expresión $(g_1, \ldots, g_n \quad E)$ como una forma abreviada de $(g_1 \wedge \ldots \wedge g_n \wedge E)$. Por analogía con la programación lógica con restricciones, usamos el símbolo   en los subobjetivos extendidos como nexo entre la conjunción de subobjetivos y el sistema complejo E, ya que nuestro mecanismo operacional considera el sistema E como una restricción.

Definición 3.2.5 Objetivo extendido

Un objetivo extendido es una conjunción de subobjetivos extendidos.

Llamaremos *subobjetivos negados* a los subobjetivos extendidos de la forma $\sim \exists \tilde{Z}(g_1, \ldots, g_n \quad E)$. El resto de subobjetivos extendidos, las ecuaciones, se llamarán *subobjetivos no negados*.

Un concepto importante dentro de la negación constructiva ecuacional es el de versión positiva de un subobjetivo negado, ya que la resolución de este último se obtiene a partir de la primera.

Definición 3.2.6 Subobjetivo complementario

El subobjetivo complementario de un subobjetivo negado de la forma $\sim \exists \tilde{Z}(g_1, \ldots, g_n \quad E)$ *es el subobjetivo* $(g_1, \ldots, g_n \quad E)$.

Ya que la negación constructiva ecuacional introduce negación y cuantificación en los objetivos derivados, no existe ningún inconveniente en extender la sintaxis de una teoría ecuacional normal para que los cuerpos de las cláusulas sean conjunciones de subobjetivos extendidos. Para ello, basta con expresar cada disecuación $t \neq s$ como el subobjetivo extendido $\sim (t = s \quad \emptyset)$.

Ejemplo 3 Teoría ecuacional normal extendida

La teoría ecuacional normal del ejemplo 1 en la Sección 2.2 puede expresarse de forma extendida como

$$
\begin{array}{lrcl}
\mathcal{E}^{\sim} = \{ & 0/m = 0 & \Leftarrow & \sim (m = 0 \quad \emptyset). \\
 & m/c = n/d & \Leftarrow & m *_{\mathrm{N}} d = c *_{\mathrm{N}} n, \sim (c = 0 \quad \emptyset), \\
 & & & \sim (d = 0 \quad \emptyset). \\
 & (m/c) + (n/d) & = & ((m *_{\mathrm{N}} d) +_{\mathrm{N}} (c *_{\mathrm{N}} n))/(c *_{\mathrm{N}} d). \\
 & (m/c) * (n/d) & = & (m *_{\mathrm{N}} n)/(c *_{\mathrm{N}} d). \}
\end{array}
$$

Similarmente, un objetivo normal g puede expresarse como un objetivo extendido g', donde cada disecuación $t \neq s$ de g tiene la forma $\sim (t = s \quad \emptyset)$ en g'.

A continuación mostramos cómo el algoritmo de "narrowing" condicional puede extenderse para resolver constructivamente subobjetivos negados. Para este propósito, los nodos del árbol de derivaciones para un objetivo G se consideran compuestos de una parte, la *parte objetivo*, que representa la conjunción de subobjetivos por resolver y de otra parte, la *parte restricción*, que representa una respuesta parcial al objetivo. La sustitución de "narrowing" aplicada en cada paso de derivación, representada como un sistema de ecuaciones, se añade como una restricción al correspondiente nodo derivado. Las figuras siguientes muestran, respectivamente, la frontera de los árboles de derivaciones

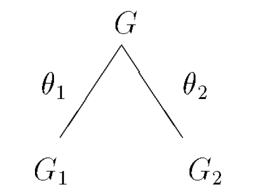

Figura 3.2: *Frontera de G usando el algoritmo de "narrowing".*

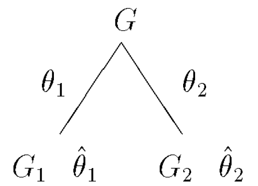

Figura 3.3: *Frontera de G usando el método de negación constructiva ecuacional.*

para un objetivo dado, usando un algoritmo de "narrowing" (Figura 3.2) y usando el método de negación constructiva (Figura 3.3).

Presentamos primero la regla de negación constructiva ecuacional que resuelve subobjetivos negados y, luego, definimos un mecanismo operacional (basado igualmente en "narrowing") que resuelve objetivos extendidos. En lo que sigue, $g \quad E$ denota un nodo del árbol de derivaciones, donde g es un objetivo normal extendido y E es un sistema complejo que representa una respuesta parcial al objetivo inicial.
NOTA. A menudo, representaremos la frontera de un objetivo como la disyunción de los nodos que la forman.

Definición 3.2.7 Regla de negación constructiva ecuacional

Sea $s : \sim \exists \tilde{Z}(g \quad E)$ un subobjetivo negado seleccionado para ser resuelto. La frontera del árbol de derivaciones para s se define como:

* *Si g sólo contiene ecuaciones, no tiene redexes de "narrowing" y no unifica sintácticamente, entonces la frontera de s es vacía.*

* *Si g sólo contiene ecuaciones, no tiene redexes de "narrowing" y unifica sintácticamente con mgu σ, entonces la frontera de s tiene como único nodo $\emptyset \;\; \forall \tilde{Z} \sim (E, \hat{\sigma})$, si el sistema complejo $\sim \exists \tilde{Z}\,(E, \hat{\sigma})$ es resoluble.*

* *Si g tiene redexes de "narrowing" entonces sea $\{(g_1 \quad (E, \hat{\theta}_1)), \ldots, (g_m \quad (E, \hat{\theta}_m))\}$ la frontera del árbol de derivaciones para el subobjetivo complementario de s, $g \quad E$, donde cada g_i es un objetivo extendido y θ_i es la sustitución aplicada en el paso de derivación desde g a g_i. Consideramos ahora la disyunción*

$$\exists \tilde{Z}, \tilde{Y}_1(g_1 \quad E_1) \;\vee\; \ldots \;\vee\; \exists \tilde{Z}, \tilde{Y}_m(g_m \quad E_m)$$

en la que cada E_i denota $(E, \hat{\theta}_i)$ e $\tilde{Y}_i$, $1 \leq i \leq m$, representa las variables de $g_i \quad E_i$ que no aparecen en $g \quad E$. Aplicamos las siguientes reglas de transformación (de forma análoga a como hace Stuckey *para programación lógica con restricciones* [147]*):*

a) <u>Negación de la disyunción</u>: se aplica la ley de De Morgan:

$$\sim \bigvee_{j=1}^{m} \exists\, \tilde{Z}, \tilde{Y}_j\,(g_j \quad E_j) = \bigwedge_{j=1}^{m} \sim \exists\, \tilde{Z}, \tilde{Y}_j\,(g_j \quad E_j)$$

b) <u>Transformación de cada conjunción</u>: a cada una de las conjunciones obtenida tras el paso a), se le aplica la siguiente regla de descomposición:

$$\begin{aligned} \sim \exists\, \tilde{Z}, \tilde{Y}_k\,(g_k \quad E_k) \iff\; & (\emptyset \;\; \forall\, \tilde{Z}, \tilde{Y}_k \sim E_k) \;\vee \\ & (\sim \exists\, \tilde{Z}, \tilde{Y}_k\,(g_k \quad E_k)) \end{aligned}$$

Finalmente, la disyunción resultante al calcular la forma normal disyuntiva de las fórmulas obtenidas tras el paso b) representa la frontera del subobjetivo negado seleccionado s.

Ejemplo 4 Aplicación de la regla de negación constructiva ecuacional

Supongamos la siguiente teoría ecuacional normal que consta de tres cláusulas:

$e_1: \quad g(X) = X.$
$e_2: \quad f(a) = b.$
$e_3: \quad f(X) = X \Leftarrow g(X) \neq a.$

y sea $\sim (f(X) = Y \quad \emptyset)$ *el subobjetivo extendido a ser resuelto. La frontera de su subobjetivo complementario,* $f(X) = Y \quad \emptyset$*, contiene dos nodos ya que* $f(X)$*, el único redex de la ecuación* $f(X) = Y$*, unifica con la parte izquierda de la cabeza de las cláusulas* e_2 *y* e_3[3]*:*

$$\text{Frontera de } (f(X) = Y \quad \emptyset) =$$
$$\{(\underbrace{b = Y \quad X = a}_{e_2}), (\underbrace{X = Y, \sim (g(X) = a \quad \emptyset) \quad \emptyset}_{e_3})\}$$

Esta frontera se transforma de acuerdo a las reglas descritas en la Definición 3.2.7:

* *Tras el paso a) obtenemos la siguiente conjunción:*

$$\sim (b = Y \quad X = a) \wedge \sim (X = Y, \sim (g(X) = a \quad \emptyset) \quad \emptyset)$$

* *Aplicamos el paso b) a cada una de las fórmulas anteriores:*

$$\sim (b = Y \quad X = a) \iff$$
$$(\emptyset \quad \sim (X = a)) \vee (\sim (b = Y \quad X = a))$$
$$\sim (X = Y, \sim (g(X) = a \quad \emptyset) \quad \emptyset) \iff$$
$$(\emptyset \quad \sim \emptyset) \vee (\sim (X = Y, \sim (g(X) = a \quad \emptyset) \quad \emptyset))$$

* *Tomando ahora la forma normal disyuntiva obtenemos la frontera del subobjetivo negado* $\sim (f(X) = Y \quad \emptyset)$*:*

$$\text{Frontera de } (\sim (f(X) = Y \quad \emptyset)) =$$
$$\{(\emptyset \quad \sim (X = a), \sim \emptyset),$$
$$(\sim (b = Y \quad X = a) \quad \sim \emptyset),$$
$$(\sim (X = Y, \sim (g(X) = a \quad \emptyset) \quad \emptyset) \quad \sim (X = a)),$$
$$(\sim (b = Y \quad X = a), \sim (X = Y, \sim (g(X) = a \quad \emptyset) \quad \emptyset) \quad \emptyset)\}$$

[3] Por simplicidad, no hemos realizado el renombramiento de las variables de las cláusulas.

La forma general de cada nodo de la frontera del árbol de derivaciones para un subobjetivo negado $\sim \exists \tilde{Z}(g \quad E)$, obtenida aplicando la regla de negación constructiva ecuacional a la frontera $\{g_1 \quad E_1, \ldots, g_m \quad E_m\}$ de su subobjetivo complementario, es una conjunción de subobjetivos extendidos aún por resolver junto con un sistema complejo. En el primer nodo de esta frontera, de la forma $(\emptyset \quad \forall \tilde{Z}, \tilde{Y}_1 \sim E_1, \ldots, \forall \tilde{Z}, \tilde{Y}_m \sim E_m)$, la conjunción de subobjetivos a resolver es vacía. Por lo tanto, si el sistema $(\forall \tilde{Z}, \tilde{Y}_1 \sim E_1, \ldots, \forall \tilde{Z}, \tilde{Y}_m \sim E_m)$ es resoluble, este sistema es una respuesta computada para el subobjetivo negado. De hecho, ya que cada aplicación de la regla de negación constructiva ecuacional genera una frontera cuyo primer nodo es siempre de esta forma, el sistema en este nodo es una posible respuesta. Así pues, si el subobjetivo negado tiene solución, la negación constructiva ecuacional garantiza que siempre se encuentra una respuesta en un número finito de pasos de derivación. El resto de nodos de la frontera contienen subobjetivos extendidos a ser resueltos y, en sus partes restricción, cierta información acerca de la respuesta para el subobjetivo negado. En la Figura 3.4 mostramos la relación entre las fronteras de un subobjetivo negado $\sim \exists \tilde{Z} (g \quad E)$ y su complementario $g \quad E$. Con objeto de simplificar esta figura, tratamos el caso particular de una frontera del subobjetivo complementario con sólo dos nodos. Asimismo, para facilitar la legibilidad, usamos la notación $\tilde{\exists}$ *subobjetivo* y $\tilde{\forall}$ *subobjetivo* en lugar de la más precisa $\exists \tilde{Z}$ *subobjetivo* y $\forall \tilde{Z}$ *subobjetivo*.

Definimos, en lo que sigue, el método de negación constructiva ecuacional, que constituye nuestra propuesta de mecanismo de resolución de objetivos extendidos. Básicamente, este procedimiento es una extensión del algoritmo de "narrowing" condicional que aplica la regla de la Definición 3.2.7 a los subobjetivos negados. Para resolver un objetivo extendido g, el algoritmo comienza con el objetivo $g \quad \emptyset$, realizando pasos de derivación e intentando alcanzar objetivos de la forma $\emptyset \quad E$. El sistema E en estos últimos objetivos es una respuesta posible para el objetivo inicial.

Definición 3.2.8 Método de negación constructiva ecuacional

Dada una teoría ecuacional normal canónica $\mathcal{E}^{\sim}$, un objetivo $g \quad E$, donde $g = g_1, \ldots, g_n$ es un objetivo extendido, y dado g_j el subobjetivo

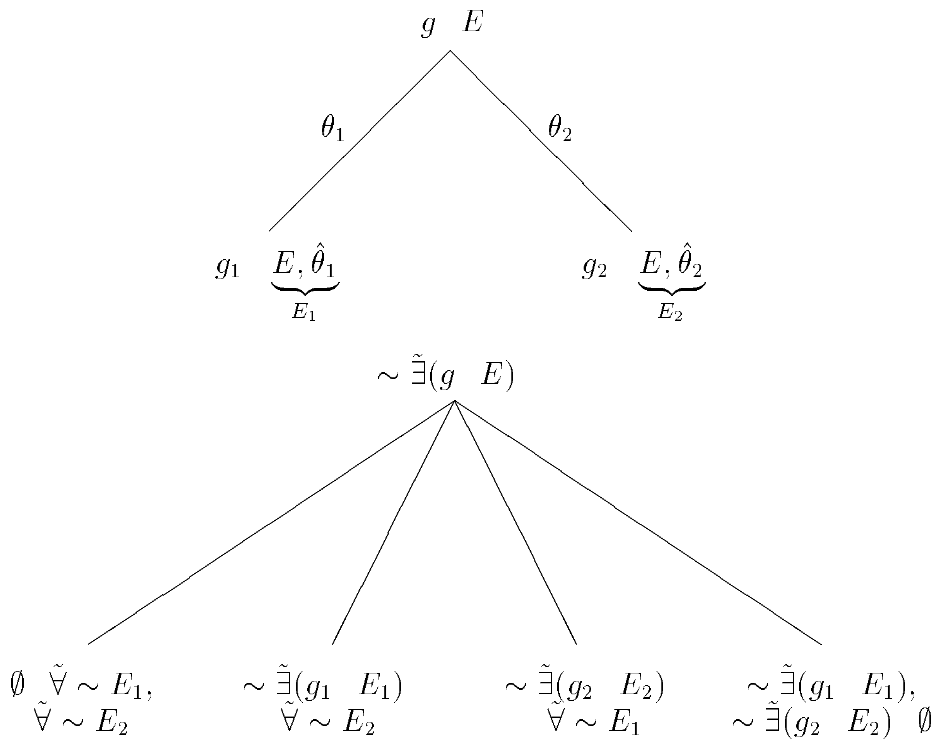

Figura 3.4: *Relación entre las fronteras de un subobjetivo negado y su complementario.*

extendido de g seleccionado para ser resuelto, la frontera de $g \quad E$ se define como:

* *Si g_j es un subobjetivo negado, entonces se aplica la regla de negación constructiva ecuacional:*
  - *Si la frontera de g_j es vacía, entonces el único nodo de la frontera de $g \quad E$ es*

    $$g_1, \ldots, g_{j-1}, g_{j+1}, \ldots, g_n \quad E$$

  - *Si $\{f_1 \quad E_1, \ldots, f_p \quad E_p\}$ es la frontera de g_j, entonces la frontera de $g \quad E$ es*
    $\{(g \setminus \{g_j\}) \cup \{f_1\} \quad (E, E_1), \ldots, (g \setminus \{g_j\}) \cup \{f_p\} \quad (E, E_p)\}$
* *Si g_j es un subobjetivo no negado, entonces se aplica el algoritmo de "narrowing" condicional para generar una frontera de g_j. Finalmente, la frontera de $g \quad E$ contiene un nodo de la forma*

  $$(g \setminus \{g_j\}) \cup \{f_i\} \quad (E, \hat{\theta}_i)$$

  por cada nodo $f_i \quad \hat{\theta}_i$ perteneciente a la frontera de g_j.

Ejemplo 5 Aplicación del método de negación constructiva ecuacional

Consideremos la siguiente teoría ecuacional normal que define la suma de los números naturales:

$$\mathcal{E}^{\sim} = \{ \begin{array}{lcl} 0 + y & = & y. \\ s(x) + y & = & s(x+y). \} \end{array}$$

y el objetivo normal $c: \; x + y \neq 0$ que expresaremos como el objetivo extendido $\sim (x + y = 0 \quad \emptyset)$. El algoritmo comienza con el objetivo inicial $g_0 : \sim (x + y = 0 \quad \emptyset) \quad \emptyset$.

La subderivación para el subobjetivo complementario de $\sim (x + y = 0 \quad \emptyset)$ tiene dos nodos a la profundidad uno del árbol de derivaciones. Es decir,

$$\begin{array}{ll} \textit{Frontera de} & (x + y = 0 \quad \emptyset) = \\ & \{(y = 0 \quad x = 0), (s(x' + y) = 0 \quad x = s(x'))\}. \end{array}$$

La aplicación de la regla de negación constructiva ecuacional da la siguiente frontera del subobjetivo negado $\sim (x + y = 0 \quad \emptyset)$*:*

$$\begin{array}{ll} \textit{Frontera de} & (\sim (x+y=0 \quad \emptyset)) = \\ & \{(\emptyset \quad x \neq 0, \forall x' \, x \neq s(x')), \\ & (\, \sim (y=0 \quad x=0) \quad \forall x' \, x \neq s(x')), \\ & (\, \sim \exists x'(s(x'+y)=0 \quad x=s(x')) \quad x \neq 0 \,), \\ & (\, \sim (y=0 \quad x=0), \\ & \quad \sim \exists x'(s(x'+y)=0 \quad x=s(x')) \quad \emptyset)\} \end{array}$$

La frontera de g_0 *contendrá exactamente los mismos nodos que la frontera de* $\sim (x + y = 0 \quad \emptyset)$ *al ser éste el único objetivo en* g_0*. Sin embargo, el primer nodo se elimina ya que su parte restricción,* $x \neq 0, \forall x' \, x \neq s(x')$*, no es satisfacible en el universo de Herbrand sobre la signatura de los naturales. Por lo tanto, los tres nodos que conforman la frontera de* g_0 *son:*

$$\begin{array}{ll} (i) & \sim (y=0 \quad x=0) \quad \forall x' \, x \neq s(x') \\ (ii) & \sim \exists x'(s(x'+y)=0 \quad x=s(x')) \quad x \neq 0 \\ (iii) & \sim (y=0 \quad x=0), \sim \exists x'(s(x'+y)=0 \quad x=s(x')) \quad \emptyset \end{array}$$

El siguiente paso de derivación para el primer nodo de esta frontera tiene como subobjetivo a resolver $\sim (y = 0 \quad x = 0)$*. Su frontera tiene un único nodo,* $\emptyset \quad \sim (x = 0, y = 0)$*, ya que en el subobjetivo complementario,* $y = 0 \quad x = 0$*, la ecuación* $y = 0$ *no tiene redexes de "narrowing" y unifica sintácticamente con mgu* $\sigma = \{y/0\}$*. De ahí que la frontera del objetivo seleccionado* (i) *es:*

$$(i.1) \quad \emptyset \quad \forall x' \, x \neq s(x'), \sim (x=0, y=0)$$

cuya parte restricción es una respuesta computada para $x + y \neq 0$ *(este sistema es satisfacible). Nótese que el sistema* $\forall x' \, x \neq s(x'), \sim (x = 0, y = 0)$ *representa un conjunto infinito de soluciones de la forma* $x = 0, y = s(n)$*, con* $n \in N$*.*

En el caso puramente positivo, la respuesta computada por "narrowing" para un objetivo dado (la composición de los mgus usados en la derivación) puede expresarse como un sistema de ecuaciones en el que la igualdad se interpreta sintácticamente. Nosotros también estamos interesados, en esta aproximación, en la obtención de sistemas respuesta cuya satisfacibilidad pueda decidirse sintácticamente. Para ello, es posible utilizar una estrategia de "innermost narrowing" ya que esta estrategia asegura que, para teorías ecuacionales completamente definidas, canónicas y con disciplina de constructores, cualquier respuesta computada por "narrowing" está formada sólo por variables y símbolos de función irreducibles. Este resultado fue presentado por Fribourg [53] demostrando que, para este tipo de teorías, la estrategia "innermost" es correcta y completa. Bajo los requerimientos anteriores sobre las teorías en consideración, los sistemas computados por el método de negación constructiva ecuacional pueden considerarse como sistemas de ecuaciones y disecuaciones sintácticas, cuya satisfacibilidad puede examinarse en el álgebra de los árboles finitos haciendo uso, por ejemplo, del algoritmo definido por Maher [105]. El método de prueba propuesto por Maher es aplicable a las álgebras de los árboles finitos e infinitos, tanto para el caso de signaturas finitas como infinitas, y se basa en un procedimiento de eliminación de cuantificadores. Aunque no siempre es posible eliminar todos los cuantificadores, este método constituye la base de un procedimiento de decisión para las teorías de las álgebras correspondientes. En el Anexo A se describe este procedimiento para el caso de interés (el álgebra de los árboles finitos cuando la signatura es finita).

3.3 El sistema de transición $\mathcal{ICN}cn$

En esta sección definimos un cálculo que resuelve objetivos extendidos con respecto a una teoría ecuacional normal canónica, completamente definida y que satisface la disciplina de constructores. Básicamente, este procedimiento es una variante del algoritmo de "innermost narrowing" condicional, que sigue el método de negación constructiva ecuacional (Definición 3.2.8). Este cálculo sólo aplica pasos de "narrowing" a las ocurrencias "innermost" del objetivo.

Definición 3.3.1 Ocurrencia "innermost"

Una ocurrencia "innermost" de una ecuación e es la ocurrencia en e de un término de la forma $f(t_1, \ldots, t_n)$ *tal que* f *es un símbolo de función definido y todos los términos* $t_i, 1 \leq i \leq n$ *están formados sólo por variables y símbolos de función irreducibles.* $f(t_1, \ldots, t_n)$ *se llama un término "innermost".*

Definición 3.3.2 Redex "innermost"

Un término t *se llama un redex "innermost" si y sólo si* t *es un término "innermost" y es un redex de "narrowing".*

El cálculo de "innermost narrowing" condicional con negación constructiva se presenta como un sistema de transición jerárquico al que llamamos $\mathcal{ICN}cn$ ("$\mathcal{I}$nnermost $\mathcal{C}$onditional $\mathcal{N}$arrowing with $c$onstructive $n$egation").

Definición 3.3.3 $\mathcal{ICN}cn$**-estado**

Un $\mathcal{ICN}cn$*-estado es un par* $\langle S, L\rangle$*, donde* S *es una lista de sistemas complejos y* L *es una lista de objetivos (posiblemente vacía) de la forma* $g \quad E$*, donde* g *es un objetivo extendido y* E *es un sistema complejo que representa una solución parcial al objetivo inicial.*

La primera componente de un $\mathcal{ICN}cn$-estado representa la lista de respuestas computadas al objetivo inicial y la segunda componente representa la lista de los nodos del árbol de derivaciones por visitar. Esta lista se trata como una cola para emular la estrategia de búsqueda en amplitud.

Definición 3.3.4 $\mathcal{ICN}cn$**-estado inicial**

Sea g *un objetivo extendido. Entonces* $\langle \emptyset, g \quad \emptyset\rangle$ *es un* $\mathcal{ICN}cn$*-estado inicial.*

Definición 3.3.5 $\mathcal{ICN}cn$**-estado terminal**

Un $\mathcal{ICN}cn$*-estado terminal es un estado de la forma* $\langle S, L\rangle$ *donde* S *es no vacío y/o* L *es la lista vacía.*

El cálculo $\mathcal{ICN}cn$ se formaliza mediante tres relaciones de transición. La relación $\rightarrow_{\mathcal{ICN}cn}$ constituye el nivel superior del cálculo, realiza el control de la expansión del árbol de derivaciones y acumula las respuestas computadas. El segundo nivel del cálculo está constituido por las relaciones $\rightarrow_f$ y $\rightarrow_{E\mathcal{CN}}$, la primera de las cuales formaliza la selección del subobjetivo a reducir y realiza la expansión del árbol justo hasta la frontera del mismo. Por último, la relación $\rightarrow_{E\mathcal{CN}}$ formaliza la regla de negación constructiva ecuacional de la Definición 3.2.7 haciendo uso de la relación $\rightarrow_f$.

Informalmente, el algoritmo selecciona un subobjetivo extendido g_i del objetivo actual. Si g_i es un subobjetivo no negado se aplica "narrowing" a una ocurrencia "innermost" de g_i. La lista de los nodos derivados se añade entonces a la lista de objetivos por resolver. Si, por el contrario, g_i es un subobjetivo negado, se aplica la regla de negación constructiva ecuacional generando una lista L que representa su frontera. Reemplazando g_i por cada elemento de la lista L en el objetivo actual se obtiene una nueva lista (la frontera del objetivo actual) que se añade igualmente a la lista de objetivos pendientes. Finalmente, cada sistema en la lista S de un $\mathcal{ICN}cn$-estado terminal $\langle S, L\rangle$ es una respuesta computada para el objetivo inicial. Las siguientes reglas definen el cálculo $\mathcal{ICN}cn$.

Definición 3.3.6 Relación de transición $\mathcal{ICN}cn$[4]

Reglas de Unificación para la Negación Constructiva

$$(1)\quad \frac{g \text{ sólo contiene ecuaciones, no tiene redexes “innermost” y} \atop \text{unifica sintácticamente con } mgu\ \sigma\ \wedge\ solvable(\sim \exists\tilde{Z}\ (E,\hat{\sigma}))}{\sim \exists\,\tilde{Z}(g\quad E) \rightarrow_{E\mathcal{CN}} [\emptyset\quad \forall\tilde{Z}\ \sim (E,\hat{\sigma})]}$$

[4]En las siguientes reglas se asume que g es no vacío.

$$(2)\quad \frac{\begin{array}{l} g \text{ sólo contiene ecuaciones, no tiene redexes “innermost” y} \\ \text{unifica sintácticamente con } mgu\ \sigma \wedge\ \sim solvable(\sim \exists \tilde{Z}\ (E, \hat{\sigma})) \end{array}}{\sim \exists\, \tilde{Z}(g \quad E) \rightarrow_{\mathcal{ECN}} [\emptyset \quad false]}$$

Reglas para la Negación Constructiva

$$(3)\quad \frac{\langle g \quad E\rangle \rightarrow_f L \ \wedge\ e\_c\_n(L, \sim \exists\, \tilde{Z}(g \quad E)) = L'}{\sim \exists\, \tilde{Z}(g \quad E) \rightarrow_{\mathcal{ECN}} L'}$$

$$(4)\quad \frac{(1), (2) \text{ y } (3) \text{ no se aplican}}{\sim \exists\, \tilde{Z}(g \quad E) \rightarrow_{\mathcal{ECN}} [\,]}$$

Reglas de expansión de la Frontera

$$(5)\quad \frac{\begin{array}{l} (e, u) = Inner(g) \wedge \\ s\_to\_l(\{\langle i, \sigma\rangle :\ narred(e, u, i, \sigma)\}) = [\langle i_j, \sigma_j\rangle]_{j=1}^{n} \end{array}}{\langle g \quad E\rangle \rightarrow_f \ [(g \setminus \{e\} \cup \{e[r_{i_j}]_u\} \cup \tilde{B}_{i_j})\sigma_j \quad (E, \hat{\sigma}_j)]_{j=1}^{n}}$$

$$(6)\quad \frac{\sim \exists\, \tilde{Z}(g' \quad E') = select(g) \wedge\ \sim \exists\, \tilde{Z}(g' \quad E') \rightarrow_{\mathcal{ECN}} \ [g'_i \quad E'_i]_{i=0}^{n}}{\langle g \quad E\rangle \rightarrow_f \ [g \setminus \{\sim \exists\, \tilde{Z}(g' \quad E')\} \cup \{g'_i\} \quad (E, E'_i)]_{i=0}^{n}}$$

$$(7)\quad \frac{\sim \exists\, \tilde{Z}(g' \quad E') = select(g) \wedge\ \sim \exists\, \tilde{Z}(g' \quad E') \rightarrow_{\mathcal{ECN}} \ [\emptyset \quad false]}{\langle g \quad E\rangle \rightarrow_f \ [\,]}$$

Regla de Unificación para Ecuaciones

$$(8)\quad \frac{\begin{array}{l} g \text{ sólo contiene ecuaciones, no tiene redexes “innermost” y} \\ \text{unifica sintácticamente con } mgu\ \sigma\ \wedge\ solvable((E, \hat{\sigma})) \end{array}}{\langle S, (g \quad E) \bullet L\rangle \rightarrow_{\mathcal{ICN}cn} \langle (E, \hat{\sigma}) \bullet S, L\rangle}$$

Reglas de “Narrowing”

$$(9)\quad \frac{\langle g \quad E\rangle \rightarrow_f L'}{\langle S, (g \quad E) \bullet L\rangle \rightarrow_{\mathcal{ICN}cn} \langle S,\ \ L \circ L'\rangle}$$

$$(10)\quad \frac{(8) \text{ y } (9) \text{ no se aplican}}{\langle S, (g \quad E) \bullet L\rangle \rightarrow_{\mathcal{ICN}cn} \langle S, L\rangle}$$

donde:

a) $Inner(g)$ selecciona de forma no determinista una ocurrencia "innermost" u de una ecuación e en g.

b) $select(g)$ selecciona de forma no determinista un subobjetivo negado de g.

c) $narred(e, u, i, \sigma) \Leftrightarrow ((l_i = r_i \Leftarrow \tilde{B}_i.) \in \mathcal{E}^{\sim} \;\; \wedge \;\; \sigma = mgu(e/u, l_i)\ )$, donde e es una ecuación.

d) la función *solvable*(E) es cierta si E es un sistema complejo satisfacible. Como ya hemos mencionado, la satisfacibilidad del sistema puede ser comprobada por algoritmos como el de Maher [105].

e) la función $e\_c\_n(L, g)$ transforma la lista L que representa la frontera del subobjetivo complementario de g, en otra lista que representa la frontera de g siguiendo el proceso descrito en la Definición 3.2.7.

f) las funciones $\bullet$ (cabeza y resto de una lista), $\circ$ (concatenación de dos listas) y s_to_l (lista de los elementos de un conjunto) tienen las definiciones usuales.

Por último, hay dos casos especiales que cabría comentar y que corresponden, respectivamente, a las situaciones en las que un objetivo extendido inicial g_0 siempre es cierto para cualquier instanciación del mismo y cuando dicho g_0 no se cumple nunca, es decir, no tiene solución. En ambos casos existe un único $\mathcal{ICN}cn$-estado terminal. Cuando el objetivo extendido inicial g_0 es cierto siempre, se produce la transición

$$\langle \emptyset, g_0 \quad \emptyset \rangle \rightarrow^*_{\mathcal{ICN}cn} \langle [\emptyset], [\,] \rangle$$

es decir, la primera componente del $\mathcal{ICN}cn$-estado terminal contiene como único sistema respuesta el sistema complejo vacío ($\emptyset$) que indica que cualquier valuación de las variables de g_0 hace a g_0 cierto. Contrariamente, cuando g_0 no tiene solución se produce la transición

$$\langle \emptyset, g_0 \quad \emptyset \rangle \rightarrow^*_{\mathcal{ICN}cn} \langle [\,], [\,] \rangle$$

es decir, la primera componente del $\mathcal{ICN}cn$-estado terminal es la lista vacía.

3.4 Optimizaciones del cálculo

En esta sección proponemos algunas optimizaciones para el cálculo descrito en la sección anterior que reducen la talla del árbol de derivaciones.

El hecho de que los sistemas respuesta se vayan construyendo, por un lado, a partir de las sustituciones de "narrowing" aplicadas a los subobjetivos no negados y, por otro, con las negaciones de la representación ecuacional de las sustituciones aplicadas en las derivaciones de los subobjetivos complementarios cuando se seleccionan subobjetivos negados, hace que en algunos casos no sea necesario llegar hasta un nodo terminal para detectar que una rama del árbol corresponde a un fallo. Por ejemplo, para un objetivo de la forma $g \quad E$ en el que el sistema E es insatisfacible, resulta innecesario resolver completamente g ya que todas las derivaciones a partir de este objetivo terminarán con fallo. Por lo tanto, podemos usar la parte restricción en cada objetivo para definir un criterio de decisión que nos permita podar el árbol y reducir así su talla. Cada vez que se aplica la regla (9) del cálculo, un simple test de la satisfacibilidad del sistema en la parte restricción de cada nodo derivado por la relación $\rightarrow_f$ es suficiente para decidir si el nodo debe o no ser eliminado. Esta optimización preserva claramente la completitud del cálculo ya que sólo se eliminan los subárboles que no contienen soluciones. El ejemplo siguiente muestra cómo se reduce la talla del árbol de derivaciones con esta optimización.

Ejemplo 6 Optimización basada en el test de satisfacibilidad de los sistemas respuesta

Consideremos la siguiente teoría normal

$$\begin{aligned} \mathcal{E}^{\sim} = \{ \quad & f(a) = a. \\ & f(X) = g(X) \Leftarrow X \neq a. \; \} \end{aligned}$$

y el objetivo normal $f(X) \neq a$. El $\mathcal{ICN}cn$-estado inicial es $\langle [\ ], [\sim (f(X) = a \quad \emptyset) \quad \emptyset] \rangle$. Aplicando las reglas (9), (6), (3) y (5) del cálculo $\mathcal{ICN}cn$ obtenemos el siguiente $\mathcal{ICN}cn$-estado:

$$
\begin{aligned}
\langle [\ ] \quad & , [\sim (f(X) = a \quad \emptyset) \quad \emptyset] \rangle \rightarrow_{\mathcal{ICN}cn} \\
& \langle [\ ], [(\emptyset \quad \sim \emptyset, X \neq a), (\sim (a = a \quad X = a) \quad \sim \emptyset), \\
& (\sim (g(X) = a, \sim (X = a \quad \emptyset) \quad \emptyset) \quad X \neq a), \\
& (\sim (a = a \quad X = a), \sim (g(X) = a, \sim (X = a \quad \emptyset) \quad \emptyset) \quad \emptyset)\ ] \rangle
\end{aligned}
$$

La lista de objetivos pendientes en este nuevo $\mathcal{ICN}cn$-estado consta de cuatro elementos (correspondientes a los cuatro nodos de la frontera del objetivo inicial). Los sistemas complejos en la parte restricción de los dos primeros elementos de esta lista son insatisfacibles ya que $\sim \emptyset$ equivale a $falso$. Por lo tanto, podemos eliminarlos de la lista, reduciéndose a la mitad la talla del árbol de derivaciones.

Este test es particularmente ventajoso si la parte objetivo en cada nodo del árbol de derivaciones contiene subobjetivos negados, dada la explosión exponencial de la talla del árbol producida cuando se aplica la regla de negación constructiva ecuacional (si la frontera del subobjetivo complementario a un subobjetivo negado contiene n nodos, la frontera del subobjetivo negado estará formada por 2^n nodos).

La segunda optimización que proponemos es una estrategia de simplificación del cálculo que extiende las técnicas estándar de simplificación del "narrowing" basadas en la distinción entre símbolos de función definidos e irreducibles [53, 72]. Con esta estrategia es posible evitar la generación de algunas soluciones redundantes, así como reducir el número de aplicaciones del cálculo.

La simplificación de ecuaciones se basa en el hecho de que dos términos cuyos símbolos de función más externos son irreducibles y distintos nunca pueden ser iguales, mientras que si se trata de símbolos irreducibles iguales, ambos términos serán iguales si y sólo si lo son sus argumentos.

Formalmente, si un objetivo en un $\mathcal{ICN}cn$-estado contiene un subobjetivo no negado de la forma $f(t_1, \ldots, t_n) = g(s_1, \ldots, s_n)$ siendo f y g irreducibles entonces:

* Si $f \neq g$, el objetivo es irreducible y puede eliminarse (*regla de rechazo*).

* Si $f = g$, el subobjetivo no negado puede reemplazarse por la conjunción de subobjetivos no negados $(t_1 = s_1, \ldots, t_n = s_n)$ (*regla de descomposición*).

Esta optimización ya se ha utilizado en algunas otras aproximaciones, como en la definición del lenguaje Slog [53] en el que estas reglas se incorporan como cláusulas de programa, o como en [136], donde se incorporan a la propia definición del "narrowing".

Estas dos reglas suelen acompañarse de una tercera, la *regla de sustitución*: si un objetivo en un $\mathcal{ICN}cn$-estado contiene un subobjetivo no negado de la forma $(X = t)$ o $(t = X)$, con $X \in V$, $X \notin \mathcal{V}(t)$ y todos los símbolos de función en t son irreducibles, el subobjetivo puede eliminarse aplicando la sustitución $\{X/t\}$ a la parte objetivo y añadiendo la ecuación $X = t$ a la parte restricción del objetivo.

Es posible enunciar reglas similares para los subobjetivos negados. Si un objetivo en un $\mathcal{ICN}cn$-estado contiene un subobjetivo negado, $\sim \exists \tilde{Z}(g_1, \ldots, g_n \;\; E)$, tal que algún g_i es un subobjetivo no negado $f(t_1, \ldots, t_n) = g(s_1, \ldots, s_n)$ siendo f y g irreducibles, entonces si $f \neq g$ el subobjetivo negado es resoluble y puede eliminarse del objetivo actual (*regla de satisfacción para la negación*). La *regla de descomposición* (cuando $f = g$) se puede aplicar de igual forma que para subobjetivos no negados. En particular, si el subobjetivo negado es de la forma $\sim \exists \tilde{Z}(t = t \;\; E)$ y $t \in \mathcal{T}(C)$, el subobjetivo negado se elimina y la fórmula $\forall \tilde{Z} \sim E$ se añade a la parte restricción del objetivo actual.

En cuanto a la *regla de sustitución* para subobjetivos negados, ésta puede enunciarse como sigue: si un objetivo en un $\mathcal{ICN}cn$-estado contiene un subobjetivo negado de la forma $\sim \exists \tilde{Z}(X = t \;\; E)$ o $\sim \exists \tilde{Z}(t = X \;\; E)$, con $X \in V$, $X \notin \mathcal{V}(t)$ y $t \in \mathcal{T}(C, V)$, este subobjetivo se elimina y la restricción $\forall \tilde{Z} \sim (X = t, E)$ se añade al sistema complejo en la parte restricción del objetivo.

Ejemplo 7 Optimización basada en las reglas de simplificación

Consideremos la siguiente teoría ecuacional normal:

$$\begin{aligned}\mathcal{E}^{\sim} = \{ \quad & f(a) = a. \\ & f(g(X)) = g(X) \Leftarrow X \neq a. \\ & f(g(X)) = X \Leftarrow X = a. \,\}\end{aligned}$$

y el objetivo normal $f(g(g(X))) \neq a$*, el cual es satisfacible para cualquier valuación de* X*.* $\langle[\,], [\sim (f(g(g(X))) = a \quad \emptyset) \quad \emptyset]\rangle$ *es el* $\mathcal{ICN}cn$*-estado inicial. Aplicando las reglas del cálculo* $\mathcal{ICN}cn$*, (9), (6), (3) y (5), obtenemos:*

$$\begin{aligned}\langle[\,] \quad & , [\sim (f(g(g(X))) = a \quad \emptyset) \quad \emptyset\,]\rangle \rightarrow_{\mathcal{ICN}cn} \\ & \langle[\,], [(\emptyset \quad (\sim \emptyset, \sim \emptyset)), \\ & (\sim (g(g(X)) = a, \sim (g(X) = a \quad \emptyset) \quad \emptyset) \quad \sim \emptyset), \\ & (\sim (g(X) = a, g(X) = a \quad \emptyset) \quad \sim \emptyset), \\ & (\sim (g(g(X)) = a, \sim (g(X) = a \quad \emptyset) \quad \emptyset), \\ & \sim (g(X) = a, g(X) = a \quad \emptyset) \quad \emptyset)\,]\rangle\end{aligned}$$

Los tres primeros elementos de la lista de objetivos pendientes se eliminan por la optimización basada en el test de satisfacibilidad del sistema respuesta, reduciéndose el $\mathcal{ICN}cn$*-estado actual a:*

$$\begin{aligned}\langle[\,] \quad & , [(\sim (g(g(X)) = a, \sim (g(X) = a \quad \emptyset) \quad \emptyset), \\ & \sim (g(X) = a, g(X) = a \quad \emptyset) \quad \emptyset)\,]\rangle\end{aligned}$$

Aplicando ahora la regla de satisfacción para la negación al primer subobjetivo negado $\sim (g(g(X)) = a, \sim (g(X) = a \quad \emptyset) \quad \emptyset)$*, éste se elimina ya que es satisfacible por ser* $g(g(X)) = a$ *insatisfacible. El segundo subobjetivo negado,* $\sim (g(X) = a, g(X) = a \quad \emptyset)$*, se elimina igualmente con la misma regla de simplificación, pues también* $g(X) = a$ *es insatisfacible. Por lo tanto, con estas simplificaciones, el* $\mathcal{ICN}cn$*-estado actual queda simplificado a*

$$\langle[\,], [\emptyset \quad \emptyset]\rangle$$

Ahora, con un nuevo paso en el cálculo $\mathcal{ICN}cn$ (a través de la regla (8)) terminamos la derivación al alcanzar un $\mathcal{ICN}cn$-estado terminal que, además, ya no contiene objetivos pendientes a resolver.

$$\langle [\,], [\emptyset \quad \emptyset] \rangle \rightarrow_{\mathcal{ICN}cn} \langle [\emptyset], [\,] \rangle$$

El ejemplo pone de manifiesto la utilidad de estas simplificaciones que reducen claramente el número de pasos del cálculo necesarios para resolver un cierto objetivo inicial.

Capítulo 4

Semántica declarativa

En este capítulo presentamos una semántica declarativa para las teorías ecuacionales normales consideradas en el capítulo anterior. Es bien conocido que el significado de una teoría ecuacional de Horn $\mathcal{E}$ puede determinarse en el dominio de Herbrand [54, 64]. Usando una lógica a dos valores, el significado declarativo de una teoría ecuacional $\mathcal{E}$ viene caracterizado por su E-modelo mínimo de Herbrand, definido como la intersección de todos los E-modelos de Herbrand de $\mathcal{E}$. Asimismo, existe una construcción explícita de este E-modelo mínimo como el menor punto fijo de un operador continuo y monótono asociado a la teoría. La inclusión de la negación en el antecedente de las cláusulas ecuacionales tiene como consecuencias directas la pérdida de monotonía del operador de consecuencias inmediatas y la no existencia de un E-modelo mínimo sino de E-modelos minimales. En el campo de la programación lógica, en el que el significado de un programa se da habitualmente en términos de las consecuencias lógicas del programa completado, la presencia de la negación no sólo hace que el operador de consecuencias inmediatas pierda la propiedad de monotonía, sino que, en algunos casos, puede hacer que el programa completado sea inconsistente. Una posible solución consiste en definir restricciones sintácticas sobre los programas lógicos, surgiendo así la noción de programa lógico estratificado, y en desarrollar una teoría para operadores no monótonos y sus puntos fijos [6, 7] o bien usar una semántica por modelo perfecto [130, 131, 132]. Otra posible solución alternativa consiste en usar la lógica trivaluada de Kleene [91], aproximación seguida por Fitting, Kunen y Stuckey

[50, 97, 98, 147], entre otros.

Para establecer una semántica declarativa para las teorías ecuacionales normales seguiremos la aproximación basada en el uso de una lógica clásica a dos valores. Puesto que el mecanismo operacional descrito en el capítulo anterior es capaz de encontrar las respuestas tanto a ecuaciones como a disecuaciones, el significado de una teoría ecuacional normal se define con respecto a su versión completada, lo que nos permite inferir también información negativa. Impondremos ciertas restricciones sintácticas (análogas a las de la programación lógica) suficientes para garantizar la consistencia de la teoría completada y adaptaremos al caso ecuacional la caracterización por punto fijo para programas lógicos estratificados.

El capítulo queda organizado como sigue. En la sección primera revisamos la aproximación de Hölldobler a la semántica de una teoría ecuacional y la construcción por punto fijo de su E-modelo mínimo de Herbrand. Asimismo, mostramos cómo esta construcción no es válida para teorías ecuacionales normales ya que el operador de consecuencias inmediatas no tiene por qué ser necesariamente monótono. A continuación, definimos una noción análoga a la de programa lógico completado y caracterizamos los modelos de una teoría ecuacional normal completada. Pero en presencia de negación la teoría completada puede ser inconsistente, lo que nos lleva a definir, en la sección tercera, restricciones sintácticas (incluyendo estratificación) que garantizan la consistencia de estas teorías completadas. Finalmente, en la cuarta sección presentamos la caracterización por punto fijo a la semántica de las teorías ecuacionales normales estratificadas, siguiendo la teoría para operadores no monótonos de Apt, Blair y Walker [7].

4.1 Semántica declarativa de las teorías ecuacionales de Horn

En esta sección revisamos la semántica de una teoría ecuacional de Horn en el contexto del universo de Herbrand siguiendo la aproximación de Hölldobler [54, 64].

El *universo de Herbrand* de una teoría ecuacional de Horn $\mathcal{E}$ es el conjunto de todos los términos básicos que pueden construirse a partir de los símbolos de función en $\mathcal{E}$. La *base de Herbrand* de $\mathcal{E}$ ($B(\mathcal{E})$) es el conjunto de todos los pares de términos básicos $\langle t, s\rangle$ que pueden construirse a partir de los elementos del universo de Herbrand. Una *interpretación de Herbrand* I de $\mathcal{E}$ es un subconjunto de $B(\mathcal{E})$ tal que una ecuación $t = s$ es cierta en I si y sólo si $\langle t, s\rangle \in I$. Una *$E$-interpretación de Herbrand* I de $\mathcal{E}$ es una interpretación de Herbrand de $\mathcal{E}$ que satisface los axiomas de la igualdad (ver Definición 4.2.2) para $\mathcal{E}$. Cada E-interpretación de Herbrand I define una congruencia sobre el álgebra de los términos básicos tal que dos términos básicos t y s son congruentes si y sólo si $\langle t, s\rangle \in I$. Una interpretación de Herbrand I de $\mathcal{E}$ es un *modelo de Herbrand* de $\mathcal{E}$ si satisface cada una de las cláusulas ecuacionales de $\mathcal{E}$. Un *E-modelo de Herbrand* de $\mathcal{E}$ es un modelo de Herbrand de $\mathcal{E}$ que satisface los axiomas de la igualdad para $\mathcal{E}$. Ya que sólo vamos a considerar E-interpretaciones y E-modelos de Herbrand en esta sección, omitimos el apelativo Herbrand en lo que sigue.

Una de las características más interesantes de las teorías ecuacionales de Horn es que para éstas se cumple la propiedad de intersección de sus E-modelos.

Proposición 4.1.1 [64] **Propiedad de intersección de los E-modelos**

Sea $Mod_E(\mathcal{E})$ el conjunto de todos los E-modelos de una teoría ecuacional de Horn $\mathcal{E}$. Entonces, $M_{\mathcal{E}} = \bigcap Mod_E(\mathcal{E})$ es un E-modelo de $\mathcal{E}$.

La propiedad anterior nos permite definir la semántica declarativa de una teoría ecuacional de Horn como su E-modelo mínimo. Asimismo, existe una caracterización por punto fijo del E-modelo mínimo de una teoría ecuacional de Horn $\mathcal{E}$ con la que se construye explícitamente este E-modelo [54, 64]. Para ello, asociamos una aplicación $T_{\mathcal{E}}$ a la teoría $\mathcal{E}$.

Sea $\mathcal{E}$ una teoría ecuacional de Horn. El conjunto de todas las interpretaciones de $\mathcal{E}$, $2^{B(\mathcal{E})}$, es un retículo completo bajo el orden parcial de la inclusión de conjuntos. El elemento máximo de este retículo es $B(\mathcal{E})$, mientras que el conjunto vacío es el elemento mínimo. El supremo de cualquier conjunto S de interpretaciones ($lub(S)$) es la unión de las mismas y el ínfimo ($glb(S)$) su intersección.

NOTA. En lo que sigue consideraremos que en las cabezas de las cláusulas de una teoría ecuacional de Horn, la igualdad está orientada de izquierda a derecha, aunque a lo largo de este capítulo usaremos indistintamente ambas notaciones (igualdad en cabeza orientada o no orientada) para referirnos a una teoría ecuacional. Llamaremos teoría inversa de una teoría ecuacional $\mathcal{E}$ al conjunto de cláusulas $\mathcal{E}^{-1} = \{(r \to l \Leftarrow \tilde{B}) : (l \to r \Leftarrow \tilde{B}) \in \mathcal{E}\}$. Por ejemplo, si $\mathcal{E}$ es la teoría ecuacional de Horn:

$$\begin{array}{ll} \mathcal{E} = \{ & a \to b \\ & a \to c\ \} \end{array}$$

entonces $\mathcal{E}^{-1}$ es el conjunto de cláusulas:

$$\begin{array}{ll} \mathcal{E}^{-1}E = \{ & b \to a \\ & c \to a\ \} \end{array}$$

Nótese que el hecho de orientar la igualdad en la cabeza de las cláusulas no cambia la semántica de una teoría ecuacional de Horn, ya que cuando un par $\langle t, s \rangle$ está en el E-modelo mínimo de una teoría ecuacional de Horn entonces el par $\langle s, t \rangle$ también está en este E-modelo, habida cuenta de que todas las E-interpretaciones satisfacen los axiomas de la igualdad y que, por lo tanto, cumplen la propiedad simétrica.

Asimismo, la expresión $\langle t, s \rangle[r]_u$ denotará el par de términos resultantes de reemplazar en la ecuación $t = s$ el subtérmino a la ocurrencia u por el término r, es decir, el par de términos de la ecuación $(t = s)[r]_u$. Diremos que $\langle t, s \rangle[r]_u$ es el resultado de reemplazar en el par $\langle t, s \rangle$ el subtérmino a la ocurrencia u por el término r.

Definición 4.1.2 Operador de consecuencias inmediatas $T_{\mathcal{E}}$

Sea $\mathcal{E}$ una teoría ecuacional de Horn e I una interpretación de $\mathcal{E}$. La aplicación $T_{\mathcal{E}} : 2^{B(\mathcal{E})} \longrightarrow 2^{B(\mathcal{E})}$ se define como:

$$\begin{array}{rl} T_{\mathcal{E}}(I) = & \{\langle t, t \rangle : t \textit{ es un término básico}\} \cup \\ & \{\langle t, s \rangle \in B(\mathcal{E}) : \textit{existe una ocurrencia } u \in O(t = s) \\ & \textit{y una instancia básica} \\ & (l \to r \Leftarrow \tilde{B}) \textit{ de una cláusula en } \mathcal{E} \textit{ tal que} \\ & (t = s)/u = l \textit{ y } \dot{B} \cup \{\langle t, s \rangle[r]_u\} \subseteq I\} \end{array}$$

siendo $\dot{B} = \{\langle u, v\rangle : u = v \in \tilde{B}\}$.

Informalmente, $T_{\mathcal{E}}(I)$ contiene el conjunto de todas las instancias básicas del axioma reflexivo y el conjunto de todos los pares $\langle t, s\rangle$ de términos básicos que pueden construirse desde elementos de I reemplazando una ocurrencia de la parte derecha de la cabeza de una instancia de una cláusula de la teoría por su parte izquierda, previa comprobación de que el cuerpo de dicha cláusula es cierto en I.

El operador $T_{\mathcal{E}}$ de la definición anterior es continuo y monótono [64]. Entonces podemos aplicar los resultados de la teoría del punto fijo para operadores monótonos y obtener el menor punto fijo de $T_{\mathcal{E}}$ $(mpf(T_{\mathcal{E}}))$. Este menor punto fijo se computa como $T_{\mathcal{E}} \uparrow \omega$, donde ω es el primer ordinal infinito y donde las potencias del operador $T_{\mathcal{E}}$ se definen como:

$$T_{\mathcal{E}} \uparrow 0 = \emptyset$$
$$T_{\mathcal{E}} \uparrow n = T_{\mathcal{E}}(T_{\mathcal{E}} \uparrow (n-1)) \text{ si } n \text{ es un ordinal sucesor}$$

Así pues, una teoría ecuacional de Horn $\mathcal{E}$ también puede caracterizarse como el menor punto fijo de la aplicación $T_{\mathcal{E}}$. De hecho es deseable que la semántica declarativa y la caracterización por punto fijo sean idénticas. En programación lógica la correspondencia entre ambas semánticas se deriva del hecho de que una interpretación I es modelo de un programa lógico P si y sólo si se cumple que $T_P(I) \subseteq I$. Sin embargo, esta relación entre los modelos y el operador de consecuencias inmediatas no se cumple directamente en programación ecuacional, dadas las propiedades de la igualdad que deben satisfacer todos los E-modelos de una teoría ecuacional.

Teorema 4.1.3 [64] *Sea $\mathcal{E}$ una teoría ecuacional de Horn. Si I es un E-modelo de $\mathcal{E}$ entonces $T_{\mathcal{E}}(I) \subseteq I$.*

El recíproco del Teorema 4.1.3 no se cumple a no ser que I sea una E-interpretación. Por ejemplo, sea $\mathcal{E}$ la teoría ecuacional de Horn:

$$\mathcal{E} = \{ \begin{array}{l} a \to c \\ b \to d \; \} \end{array}$$

y sea I la siguiente interpretación de $\mathcal{E}$:

$$\begin{array}{ll} I = \{ & \langle a,a\rangle, \langle b,b\rangle, \langle c,c\rangle, \langle d,d\rangle, \langle a,b\rangle, \langle a,c\rangle, \\ & \langle a,d\rangle, \langle b,d\rangle, \langle c,a\rangle, \langle c,b\rangle, \langle d,b\rangle, \langle c,d\rangle \ \} \end{array}$$

Ahora,

$$\begin{array}{ll} T_{\mathcal{E}}(I) = \{ & \langle a,a\rangle, \langle b,b\rangle, \langle c,c\rangle, \langle d,d\rangle, \langle a,b\rangle, \langle a,c\rangle, \\ & \langle a,d\rangle, \langle b,d\rangle, \langle c,a\rangle, \langle c,b\rangle, \langle d,b\rangle \ \} \subseteq I \end{array}$$

Sin embargo, I no es un E-modelo ya que I no contiene, por ejemplo, los pares $\langle b,a\rangle, \langle b,c\rangle, \langle d,a\rangle$ necesarios para que I satisfaga la propiedad simétrica.

No obstante, este resultado sí se cumple si nos restringimos a E-interpretaciones, como muestra el siguiente teorema.

Teorema 4.1.4 *Sea $\mathcal{E}$ una teoría ecuacional de Horn e I una E-interpretación de $\mathcal{E}$. Si $T_{\mathcal{E}}(I) \subseteq I$ entonces I es un E-modelo de $\mathcal{E}$.*

Demostración. Asumiendo que $T_{\mathcal{E}}(I) \subseteq I$, tenemos que demostrar que I es un E-modelo de $\mathcal{E}$. Dado que I es una E-interpretación, entonces I satisface las propiedades de la igualdad (reflexiva, simétrica, transitiva y sustitutividad para funciones). Por lo tanto, sólo tenemos que probar que para todas las instancias básicas de las cláusulas de $\mathcal{E}$, $(l \rightarrow r \Leftarrow \tilde{B})$, se cumple que si $\dot{B} \subseteq I$ entonces $\langle l,r\rangle \in I$. Ya que $\langle r,r\rangle \in I$ por ser I una E-interpretación, $\langle l,r\rangle \in T_{\mathcal{E}}(I)$ reemplazando el término r a la ocurrencia 1 en el par $\langle r,r\rangle$. Finalmente, como $T_{\mathcal{E}}(I) \subseteq I$ entonces $\langle l,r\rangle \in I$.

El menor punto fijo de la aplicación $T_{\mathcal{E}\cup\mathcal{E}^{-1}}$ es una E-interpretación que satisface los Teoremas 4.1.3 y 4.1.4. De aquí que podamos identificar el E-modelo mínimo de una teoría ecuacional de Horn $\mathcal{E}$ como el menor punto fijo de $T_{\mathcal{E}\cup\mathcal{E}^{-1}}$.

Teorema 4.1.5 [64] *Sea $\mathcal{E}$ una teoría ecuacional de Horn. Entonces,*
$$M_{\mathcal{E}} = mpf(T_{\mathcal{E}\cup\mathcal{E}^{-1}})$$

Para teorías ecuacionales de Horn confluentes el menor punto fijo de $T_{\mathcal{E}}$ coincide con el menor punto fijo de $T_{\mathcal{E}\cup\mathcal{E}^{-1}}$, no siendo necesario hacer uso de la teoría inversa para construir su E-modelo mínimo.

Luego podemos concluir que una teoría ecuacional de Horn $\mathcal{E}$ confluente se caracteriza por los siguientes hechos:

a) El operador de consecuencias inmediatas $T_{\mathcal{E}}$ es monótono.

b) Existe un E-modelo mínimo de $\mathcal{E}$ caracterizado como el menor punto fijo de $T_{\mathcal{E}}$.

c) El E-modelo mínimo de $\mathcal{E}$ es la intersección de todos los E-modelos de $\mathcal{E}$.

Para teorías ecuacionales normales podríamos proceder de forma análoga, definiendo una aplicación apropiada asociada a la teoría. Nótese que los conceptos definidos al principio de esta sección (base de Herbrand, interpretación,...) siguen siendo válidos cuando consideramos teorías ecuacionales normales; por ejemplo, dada una teoría ecuacional normal $\mathcal{E}^{\sim}$, una interpretación de Herbrand I de $\mathcal{E}^{\sim}$ es un subconjunto de $B(\mathcal{E}^{\sim})$ tal que una ecuación $t = s$ es cierta en I si y sólo si $\langle t, s\rangle \in I$.
NOTA. Por simplicidad, en lo que resta de este capítulo representaremos una cláusula ecuacional normal $(t = s \Leftarrow e_1, \ldots, e_m, d_1, \ldots, d_n.)$ como $(t = s \Leftarrow \tilde{E}, \tilde{D}.)$, donde $\tilde{E}$ denota la conjunción $e_1, \ldots, e_m$ y $\tilde{D}$ denota la conjunción $d_1, \ldots, d_n$. Asimismo, $\dot{D}$ denota el conjunto $\{\langle u, v\rangle : u \neq v \in \tilde{D}\}$.

Definición 4.1.6 *Sea $\mathcal{E}^{\sim}$ una teoría ecuacional normal e I una interpretación de $\mathcal{E}^{\sim}$. Definimos la aplicación $T_{\mathcal{E}^{\sim}} : 2^{B(\mathcal{E}^{\sim})} \longrightarrow 2^{B(\mathcal{E}^{\sim})}$ como:*

$$\begin{aligned} T_{\mathcal{E}^{\sim}}(I) = \ & \{\langle t, t\rangle : t \text{ es un término básico}\} \cup \\ & \{\langle t, s\rangle \in B(\mathcal{E}^{\sim}) : \text{existe una ocurrencia} \\ & u \in O(t = s) \text{ y una instancia básica} \\ & (l \to r \Leftarrow \tilde{E}, \tilde{D}.) \text{ de una cláusula en } \mathcal{E}^{\sim} \text{ tal que} \\ & (t = s)/u = l, \dot{E} \cup \{\langle t, s\rangle[r]_u\} \subseteq I \text{ y } \dot{D} \subseteq B(\mathcal{E}^{\sim}) \setminus I\} \end{aligned}$$

Esta definición es consistente con la Definición 4.1.2 ya que ambas coinciden cuando la teoría ecuacional normal no contiene negación en el cuerpo de las cláusulas, es decir, cuando la teoría normal es una teoría ecuacional de Horn.

Puesto que disponemos de un operador $T_{\mathcal{E}^\sim}$ asociado con cada teoría ecuacional normal $\mathcal{E}^\sim$, podríamos intentar caracterizar dicha teoría como el menor punto fijo de $T_{\mathcal{E}^\sim}$. Sin embargo, esto no es posible ya que:

a) el resultado establecido por el Teorema 4.1.5 no puede generalizarse a teorías ecuacionales normales. En presencia de negación el operador de consecuencias inmediatas no tiene por qué ser monótono y, por consiguiente, puede no tener puntos fijos.

   Ejemplo 8 *Sea* $\mathcal{E}^\sim = \{a \to c \Leftarrow a \neq b.\}$ *y consideremos las siguientes interpretaciones de* $\mathcal{E}^\sim$*:*

   $$I_1 = \emptyset$$
   $$I_2 = \{\langle a, a\rangle, \langle b, b\rangle, \langle c, c\rangle, \langle a, b\rangle, \langle b, a\rangle\}$$

   entonces

   $$T_{\mathcal{E}^\sim}(I_1) = \{\langle a, a\rangle, \langle b, b\rangle, \langle c, c\rangle, \langle a, c\rangle, \langle c, a\rangle\}$$
   $$T_{\mathcal{E}^\sim}(I_2) = \{\langle a, a\rangle, \langle b, b\rangle, \langle c, c\rangle\}$$

   Luego, $I_1 \subseteq I_2$ *pero* $T_{\mathcal{E}^\sim}(I_1) \not\subseteq T_{\mathcal{E}^\sim}(I_2)$.

b) una teoría ecuacional normal puede tener varios E-modelos minimales y no tener un E-modelo mínimo.

   Ejemplo 9 *Siguiendo con el ejemplo anterior, los conjuntos*

   $$M_1 = \{\langle a, a\rangle, \langle b, b\rangle, \langle c, c\rangle, \langle a, c\rangle, \langle c, a\rangle\} \text{ y}$$
   $$M_2 = \{\langle a, a\rangle, \langle b, b\rangle, \langle c, c\rangle, \langle a, b\rangle, \langle b, a\rangle\}$$

   son ambos E*-modelos minimales de* $\mathcal{E}^\sim$*, pero no existe ningún* E*-modelo mínimo contenido en ellos.*

c) la propiedad de intersección de los E-modelos (Proposición 4.1.1) no siempre se cumple.

Ejemplo 10 *Siguiendo con la teoría ecuacional de los dos ejemplos anteriores, la intersección de M_1 y M_2, es decir, el conjunto $\{\langle a,a\rangle, \langle b,b\rangle, \langle c,c\rangle\}$, no es un E-modelo de $\mathcal{E}^{\sim}$.*

Aproximación algebraica

Kaplan [85] presenta una aproximación por punto fijo alternativa para definir la semántica de un SRTC P/N (ver Sección 2.2.1). A cada SRTC P/N R se le asocia una función Ψ_R tal que, si S es una relación binaria y S^* es su cierre reflexivo y transitivo, entonces $\Psi_R(S)$ es una relación binaria definida como el conjunto de todos los pares $\langle p,q\rangle$ tal que para alguna regla $(l \to r \Leftarrow (\bigwedge_{i=1}^{m} u_i = v_i) \wedge (\bigwedge_{j=1}^{n} u'_j \neq v'_j))$ de R, una ocurrencia $k \in O(p)$ y una sustitución σ se cumple que:

$$p/k = l\sigma, q = p[r\sigma]_u$$
$$\exists r_i : \langle u_i\sigma, r_i\rangle \in S^* \text{ y } \langle v_i\sigma, r_i\rangle \in S^*, \forall i : 1 \leq i \leq m$$
$$\nexists r'_j : \langle u'_j\sigma, r'_j\rangle \in S^* \text{ y } \langle v'_j\sigma, r_j\rangle \in S^*, \forall j : 1 \leq i \leq n$$

En esta aproximación, el significado de la relación de reescritura definida por R se identifica con el menor punto fijo de Ψ_R cuando éste existe. Pero el menor punto fijo de Ψ_R sólo existe cuando se satisfacen ciertas condiciones que aseguran la decidibilidad y la terminación finita de todas las secuencias de reescritura. Luego esta aproximación presenta los mismos problemas que hemos planteado ya que:

a) Ψ_R es no monótono.

b) Además, no se puede distinguir entre aquellos casos para los que sabemos que no se produce una reescritura y aquellos otros para los que no sabemos si puede producirse una reescritura. En particular, esta situación se presenta cuando existen reducciones que no terminan.

Kaplan resuelve el problema imponiendo a los SRTCs P/N las condiciones de ser 'reduciente' (ver Sección 2.2.1) y confluente. Bajo estas restricciones, una de las álgebras minimales ("quasi-initial") del SRTC puede computarse por reescritura como el conjunto de todas las formas normales. Este álgebra, además, corresponde al significado esperado del SRTC P/N.

Nosotros planteamos en este capítulo una solución alternativa ya que estamos interesados en definir el significado de una teoría ecuacional normal con respecto a su versión completada.

4.2 Teorías ecuacionales normales completadas

Puesto que la negación constructiva ecuacional permite deducir negaciones de axiomas ecuacionales, deberíamos disponer de una semántica declarativa apropiada que permita inferir dicha información negativa. En el contexto de la programación lógica la semántica declarativa más ampliamente aceptada para inferir información negativa es *la compleción de programas de Clark* [23]. La noción de compleción de un programa fue introducida por Clark como una descripción declarativa de la regla de fallo finito [8, 75, 79, 99, 142, 143, 153]. Esta semántica declarativa se basa en considerar modelos del programa completado, denotado por $comp(P)$, el cual se obtiene como una transformación adecuada del programa lógico P. En esta sección extendemos la noción de compleción a la programación ecuacional, de forma que la semántica declarativa de una teoría ecuacional normal se define considerando los modelos de la teoría completada. Esta noción de compleción no está relacionada con la compleción de un sistema de reescritura usando, por ejemplo, el algoritmo de Knuth-Bendix. Seguimos aquí la presentación de Apt [6] y Lloyd [103] para la compleción en programación lógica.

Una teoría ecuacional normal puede verse como una colección de sentencias de la forma 'si ... entonces ...'. Pero con esta visión no podemos establecer conclusiones negativas. Tratando las cláusulas como sentencias de la forma '... si y sólo si ...' obtenemos una interpretación más fuerte que nos permite tener también conclusiones negativas. Informalmente, completar una teoría ecuacional normal consiste en reemplazar las definiciones 'si' de las cláusulas por definiciones 'si y sólo si' y añadir una cierta teoría de la igualdad. Como cada término $f(t_1, \ldots, t_n)$ es igual a sí mismo y esto no está explícitamente expresado en las cláusulas ecuacionales que definen el símbolo f, deberemos incluir esta información en la definición completada de f para reemplazar correctamente el símbolo $\Leftarrow$ en las cláusulas de la teoría por símbolos $\Longleftrightarrow$. Para ello introducimos un nuevo símbolo $\equiv$ cuya interpretación es la identidad sintáctica. La compleción realmente corresponde al significado esperado de una teoría ecuacional normal: de una cierta teoría normal $\mathcal{E}^{\sim}$ se infiere la negación de una ecuación si podemos probar que la ecuación no se deriva de $comp(\mathcal{E}^{\sim})$.

Definición 4.2.1 Definición completada de un símbolo de función

Sea $\mathcal{E}^{\sim}$ una teoría ecuacional normal. Consideremos la siguiente secuencia de transformaciones sobre el conjunto de cláusulas de $\mathcal{E}^{\sim}$. Sea

$$f(t_1, \ldots, t_n) = t \Leftarrow \tilde{E}, \tilde{D}.$$

una cláusula de $\mathcal{E}^{\sim}$. Transformamos esta cláusula en

$$\begin{aligned} f(x_1, \ldots, x_n) = w \Leftarrow \exists\, y_1, \ldots, y_k\, (\ & (x_1 = t_1) \wedge \ldots \wedge (x_n = t_n) \wedge \\ & (w = t) \wedge \tilde{E} \wedge \tilde{D}\ ) \end{aligned}$$

donde $y_1, \ldots, y_k$ son las variables de la cláusula original y $x_1, \ldots, x_n, w$ son variables nuevas que no aparecen en $\mathcal{E}^{\sim}$. Ahora, todas las fórmulas obtenidas en el paso anterior con el mismo símbolo de función f en la parte izquierda de la cabeza

$$\begin{gathered} f(x_1, \ldots, x_n) = w \Leftarrow F_1 \\ \vdots \\ f(x_1, \ldots, x_n) = w \Leftarrow F_m \end{gathered}$$

se reemplazan por

$$f(x_1, \ldots, x_n) = w \Leftarrow F_1 \vee \ldots \vee F_m$$

Finalmente, la definición completada de f es la fórmula

$$\begin{aligned} \forall\, x_1, \ldots, x_n, w\, (f(x_1, \ldots, x_n) = w \Longleftrightarrow \quad & F_1 \vee \ldots \vee F_m \vee \\ & (w \equiv f(x_1, \ldots, x_n))) \end{aligned}$$

Ejemplo 11 *La definición completada del símbolo de función + del Ejemplo 5 de la Sección 3.2 es:*

$$\begin{array}{lll}\forall\, x_1, x_2, w\, (+(x_1, x_2) = w & \Longleftrightarrow & \exists\, y\, ((x_1 = 0) \wedge (x_2 = y) \wedge (w = y)) \vee \\ & & \exists\, x, y\, ((x_1 = s(x)) \wedge (x_2 = y) \wedge \\ & & \qquad (w = s(+(x, y)))) \vee \\ & & (w \equiv +(x_1, x_2)))\end{array}$$

A continuación definimos el concepto de compleción de una teoría ecuacional normal. Con objeto de simplificar la notación, en la definición siguiente usaremos el símbolo $\doteq$ para denotar tanto la igualdad, $=$, como la identidad sintáctica, $\equiv$.

Definición 4.2.2 Teoría ecuacional normal completada

Sea $\mathcal{E}^{\sim}$ una teoría ecuacional normal. La teoría completada de $\mathcal{E}^{\sim}$, denotada como comp($\mathcal{E}^{\sim}$), consta de la colección de definiciones completadas de los símbolos de función de $\mathcal{E}^{\sim}$ junto con la siguiente teoría de la igualdad de Clark (C_{ET})[1], en la que los axiomas de libertad para la igualdad (axiomas e) y f) siguientes) se definen sólo sobre los símbolos de función irreducibles:

a) $\forall\, (x \doteq x)$.

b) $\forall\, ((x_1 \doteq x_2) \Rightarrow (x_2 \doteq x_1))$.

c) $\forall\, ((x_1 \doteq x_2) \;\wedge\; (x_2 \doteq x_3) \Rightarrow (x_1 \doteq x_3))$.

d) $\forall\, ((x_1 \doteq y_1) \;\wedge\; \ldots \;\wedge\; (x_n \doteq y_n) \Rightarrow f(x_1, \ldots, x_n) \doteq f(y_1, \ldots, y_n))$ *para cada símbolo de función f.*

e) $\forall\, (f(x_1, \ldots, x_n) \not\doteq g(y_1, \ldots, y_m))$,

* *para todo par de símbolos de función irreducibles y distintos f y g ($n, m \geq 0$) si $\doteq$ es $=$.*
* *para todo par de símbolos de función distintos f y g ($n, m \geq 0$) si $\doteq$ es $\equiv$.*

[1]Cada una de las siguientes reglas debe entenderse como una abreviatura de dos reglas, una para la igualdad y otra para la identidad.

f) $\forall\,(f(x_1,\ldots,x_n) \doteq f(y_1,\ldots,y_n) \Rightarrow (x_1 \doteq y_1) \,\wedge\, \ldots \,\wedge\, (x_n \doteq y_n))$,

- *para cada símbolo de función irreducible f si $\doteq$ es $=$.*
- *para cada símbolo de función f si $\doteq$ es $\equiv$.*

g) $\forall x \in \mathcal{T}(C)\,(t[x] \doteq x)$, *para cada término $t[x]$ que contiene a x y que es diferente de x si*

- *todos los símbolos de función que aparecen en t son irreducibles y $\doteq$ es $=$.*
- *$\doteq$ es $\equiv$.*

La semántica por teoría de modelos de una teoría ecuacional normal se basa en considerar modelos de $comp(\mathcal{E}^{\sim})$[2] en lugar de E-modelos de la propia teoría $\mathcal{E}^{\sim}$. Por lo tanto, bajo la semántica de la teoría completada, una sentencia es cierta con respecto a $\mathcal{E}^{\sim}$ si y sólo si es cierta en todos los modelos de $comp(\mathcal{E}^{\sim})$ y falsa con respecto a $\mathcal{E}^{\sim}$ si y sólo si es falsa en todos los modelos de $comp(\mathcal{E}^{\sim})$. La proposición siguiente muestra cómo cualquier modelo de $comp(\mathcal{E}^{\sim})$ es modelo de $\mathcal{E}^{\sim}$.

Proposición 4.2.3 *Sea $\mathcal{E}^{\sim}$ una teoría ecuacional normal. Entonces $\mathcal{E}^{\sim}$ es una consecuencia lógica de $comp(\mathcal{E}^{\sim})$.*

DEMOSTRACIÓN. Sea M un modelo de $comp(\mathcal{E}^{\sim})$. Tenemos que demostrar que M es un E-modelo de $\mathcal{E}^{\sim}$. Obviamente M satisface los axiomas de la igualdad por ser modelo de C_{ET}. Sea $(f(t_1,\ldots,t_n) = t \Leftarrow \tilde{E}, \tilde{D}.)$ una cláusula ecuacional normal de $\mathcal{E}^{\sim}$ y supongamos que $\tilde{E}$ y $\tilde{D}$ son ciertos en M para alguna asignación de las variables de la cláusula $y_1,\ldots,y_k$. Consideremos la definición completada del símbolo f:

$$\forall\, x_1,\ldots,x_n,w\,(f(x_1,\ldots,x_n) = w \iff F_1 \,\vee\, \ldots \,\vee\, F_m \,\vee\, (w \equiv f(x_1,\ldots,x_n)))$$

[2]Todos los modelos de $comp(\mathcal{E}^{\sim})$ son E-modelos ya que satisfacen los axiomas de la teoría C_{ET}.

y supongamos que F_i es de la forma

$$\exists\, y_1, \ldots, y_k\ (\ (x_1 = t_1)\ \wedge\ \ldots\ \wedge\ (x_n = t_n)\ \wedge\ (w = t)\ \wedge\ \tilde{E}\ \wedge\ \tilde{D}\ )$$

Ahora, sea x_j igual a t_j $(1 \leq j \leq n)$ y w igual a t para la misma asignación de las variables $y_1, \ldots, y_k$ usada antes. Entonces, F_i es cierto en M, ya que $\tilde{E}$ y $\tilde{D}$ son ciertos en M y también porque M debe satisfacer el axioma reflexivo de C_{ET}. De aquí que $\langle f(t_1, \ldots, t_n), t\rangle \in M$.

El concepto de respuesta correcta para una teoría ecuacional normal se define en términos de la teoría completada.

Definición 4.2.4 Respuesta correcta

Sea $\mathcal{E}^{\sim}$ una teoría ecuacional normal, g un objetivo normal y E un sistema repuesta para g con respecto a $\mathcal{E}^{\sim}$. Decimos que E es una respuesta correcta para g con respecto a $comp(\mathcal{E}^{\sim})$ si para toda sustitución θ tal que $C_{ET} \models E\theta$ entonces $\forall g\theta$ es una consecuencia lógica de $comp(\mathcal{E}^{\sim})$.

Para trabajar con las teorías completadas veamos primero sus modelos. Consideramos modelos arbitrarios que estudiamos por medio de una generalización natural del operador de consecuencias inmediatas $T_{\mathcal{E}^{\sim}}$. El motivo por el que no es suficiente restringirse a modelos de Herbrand se pone de manifiesto en el ejemplo siguiente:

Sea $\mathcal{E}^{\sim}$ la teoría ecuacional normal:

$$\mathcal{E}^{\sim} = \{a = b \Leftarrow x \neq b.\}$$

cuya compleción es

$$\begin{aligned} comp(\mathcal{E}^{\sim}) = \{\forall w\, (a = w \iff \exists x \quad & (w = b \wedge x \neq b) \vee \\ & (w \equiv a))\} \end{aligned}$$

$comp(\mathcal{E}^{\sim})$ no tiene modelos de Herbrand pero sí otros modelos como, por ejemplo, $M = \{\langle a, a\rangle, \langle b, b\rangle, \langle c, c\rangle, \langle a, b\rangle, \langle b, a\rangle\}$. Luego, una teoría ecuacional completada puede no tener modelos de Herbrand y ser consistente.

El operador $T_{\mathcal{E}^\sim}$ generalizado actúa sobre interpretaciones basadas en una preinterpretación dada. Para definir este nuevo operador, introducimos la siguiente notación:
Dada una interpretación I, un átomo e de la forma $f(t_1,\ldots,t_n) = g(s_1,\ldots,s_m)$ y una asignación σ de variables con respecto a I, denotamos por $e\sigma$ (o, alternativamente, por $f(t_1,\ldots,t_n)\sigma = g(s_1,\ldots,s_m)\sigma$) el átomo generalizado $f(t_1\sigma,\ldots,t_n\sigma) = g(s_1\sigma,\ldots,s_m\sigma)$. La generalización de una conjunción de átomos $\tilde{B}$, denotada como $\tilde{B}\sigma$, se obtiene generalizando cada uno de los átomos de $\tilde{B}$.

Sea J una preinterpretación[3] e I una interpretación basada en J. Sea B_J el conjunto de todos los pares $\langle t, s\rangle$ que se pueden formar con t y s pertenecientes al dominio de la preinterpretación J. Para una teoría ecuacional normal $\mathcal{E}^\sim$ y un átomo generalizado $t = s$, decimos que $\langle t, s\rangle \in T^J_{\mathcal{E}^\sim}(I)$ si y sólo si o bien t y s son el mismo término o bien para alguna asignación σ sobre I existe una ocurrencia $u \in O(t = s)$ y una cláusula $(l \to r \Leftarrow \tilde{E}, \tilde{D}.)$ en $\mathcal{E}^\sim$ tal que $(t = s)/u = l\sigma$, $\tilde{E}\sigma \cup \{(t = s)[r\sigma]_u\} \subseteq I$ y $\tilde{D}\sigma \subseteq B_J \setminus I$. $T^J_{\mathcal{E}^\sim}$ es una aplicación desde el retículo de las interpretaciones basadas en J en sí mismo. Cuando J es la preinterpretación de Herbrand escribimos simplemente $T_{\mathcal{E}^\sim}$. $T^J_{\mathcal{E}^\sim}$ es generalmente no monotóno, pero cuando $\mathcal{E}^\sim$ es una teoría ecuacional de Horn entonces $T^J_{\mathcal{E}^\sim}$ tiene propiedades similares a las del operador $T_{\mathcal{E}^\sim}$, como mostramos en el lema siguiente.

Lema 4.2.5 *Sea $\mathcal{E}$ una teoría ecuacional de Horn y J una preinterpretación. Entonces $T^J_{\mathcal{E}}$ es continuo y, por lo tanto, monótono.*

Demostración. Sea X un subconjunto dirigido de 2^{B_J}. Nótese que $Y \subseteq lub(X)$ si y sólo si $Y \subseteq I$ para algún $I \subseteq X$. Para demostrar que $T^J_{\mathcal{E}}$ es continuo, tenemos que probar que $T^J_{\mathcal{E}}(lub(X)) = lub(\{T^J_{\mathcal{E}}(Y) : Y \subseteq X\})$ para cada subconjunto dirigido X.

Sea $u = v$ un átomo generalizado. Entonces

[3]La definición de preinterpretación es la usual de la programación lógica, es decir, consta de un dominio D, la asignación a cada constante de un elemento de D y la asignación a cada símbolo de función n-ario de una función de D^n en D.

$\langle u, v\rangle \in T^J_{\mathcal{E}}(lub(X))$
sii $\langle u, v\rangle \in \{\langle t, t\rangle : t \in B_J\}\cup$
$\{\langle t, s\rangle$: existe una ocurrencia $u \in O(t = s)$,
una cláusula $(l \to r \Leftarrow \tilde{B})$ en $\mathcal{E}$,
y una asignación de variables σ con respecto a $lub(X)$ tal que
$(t = s)/u = l\sigma$ y $\dot{B}\sigma \cup \{\langle t, s\rangle[r\sigma]_u\} \subseteq lub(X)\}$
sii $\langle u, v\rangle \in \{\langle t, t\rangle : t \in B_J\}\cup$
$\{\langle t, s\rangle$: existe una ocurrencia $u \in O(t = s)$,
una cláusula $(l \to r \Leftarrow \tilde{B})$ en $\mathcal{E}$
y una asignación de variables σ con respecto a I tal que
$(t = s)/u = l\sigma$ y $\dot{B}\sigma \cup \{\langle t, s\rangle[r\sigma]_u\} \subseteq I$
para algún $I \subseteq X\}$
sii $\langle u, v\rangle \in T^J_{\mathcal{E}}(I)$, para algún $I \subseteq X$
sii $\langle u, v\rangle \in lub(\{T^J_{\mathcal{E}}(I) : I \subseteq X\})$

A continuación, relacionamos los modelos de la compleción una teoría ecuacional normal $\mathcal{E}^{\sim}$ con el operador de consecuencias inmediatas generalizado.

Proposición 4.2.6 *Sea $\mathcal{E}^{\sim}$ una teoría ecuacional normal, J una preinterpretación de $\mathcal{E}^{\sim}$ e I una E-interpretación basada en J. Entonces I es un E-modelo de $\mathcal{E}^{\sim}$ si y sólo si $T^J_{\mathcal{E}^{\sim}}(I) \subseteq I$.*

Para demostrar esta proposición necesitamos primero el siguiente resultado debido a Hölldobler [64].

Lema 4.2.7 [64] *Sea J una preinterpretación e I una E-interpretación basada en J. Si $\{\langle l, r\rangle, \langle t, s\rangle\} \subseteq I$ y $(t = s)/u = l$ entonces $\langle t, s\rangle[r]_u \in I$.*

DEMOSTRACIÓN. (**Proposición 4.2.6**)

$\Longrightarrow$) Supongamos que $\langle t, s\rangle$ es un átomo generalizado y que $\langle t, s\rangle \in T^J_{\mathcal{E}^\sim}(I)$. Entonces tenemos que probar que $\langle t, s\rangle \in I$. Por la definición de $T^J_{\mathcal{E}^\sim}$ distinguimos dos casos:

a) Si $t = s$ es una instancia del axioma reflexivo entonces $\langle t, s\rangle \in I$ ya que I cumple la propiedad reflexiva por ser una E-interpretación.

b) Supongamos ahora que existe una ocurrencia $u \in O(t = s)$, una cláusula $(l \to r \Leftarrow \tilde{E}, \tilde{D}.)$ en $\mathcal{E}^\sim$ y una asignación σ con respecto a I tal que $(t = s)/u = l\sigma$, $\dot{E}\sigma \cup \{\langle t, s\rangle[r\sigma]_u\} \subseteq I$ y $\dot{D}\sigma \subseteq B_J \setminus I$. Ya que I es un E-modelo de $\mathcal{E}^\sim$, $\langle l\sigma, r\sigma\rangle \in I$ y, por la simetría de I, $\langle r\sigma, l\sigma\rangle \in I$. De aquí que, aplicando el Lema 4.2.7, $\langle t, s\rangle \in I$.

$\Longleftarrow$) (la demostración de la proposición en este sentido es una generalización de la prueba del Teorema 4.1.4)

Asumiendo que $T^J_{\mathcal{E}^\sim}(I) \subseteq I$, para probar que I es un E-modelo basta demostrar que para toda asignación σ con respecto a I, si $(l \to r \Leftarrow \tilde{E}, \tilde{D}.)$ es una cláusula en $\mathcal{E}^\sim$ tal que $\dot{E}\sigma \subseteq I$ y $\dot{D}\sigma \subseteq B_J \setminus I$ entonces $\langle l\sigma, r\sigma\rangle \in I$ (nótese que I ya satisface los axiomas de la igualdad por ser una E-interpretación). Ahora bien, sabemos que $\langle r\sigma, r\sigma\rangle \in I$. Por lo tanto, $\langle l\sigma, r\sigma\rangle \in T^J_{\mathcal{E}^\sim}(I)$ reemplazando el término $r\sigma$ a la ocurrencia 1 en $\langle r\sigma, r\sigma\rangle$. Entonces, por la asunción hecha al principio, concluimos que $\langle l\sigma, r\sigma\rangle \in I$.

Para demostrar que los puntos fijos de $T^J_{\mathcal{E}^\sim}$ son modelos de $comp(\mathcal{E}^\sim)$ necesitamos establecer primero algunos resultados previos. $IF(\mathcal{E}^\sim)$ denota la forma intermedia de $\mathcal{E}^\sim$ en la que la definición de un símbolo de función f es de la forma

$$\forall\, x_1, \ldots, x_n, w\, (f(x_1, \ldots, x_n) = w \Leftarrow F)$$

Llamamos $IFF(\mathcal{E}^\sim)$ al conjunto de definiciones completadas de los símbolos de función en $\mathcal{E}^\sim$. Finalmente, $ONLY\!-\!IF(\mathcal{E}^\sim)$ denota el conjunto de fórmulas obtenidas desde $IF(\mathcal{E}^\sim)$ reemplazando cada símbolo $\Leftarrow$ por $\Rightarrow$.

Proposición 4.2.8 *Dada una teoría ecuacional normal $\mathcal{E}^{\sim}$ y una preinterpretación J, una E-interpretación I basada en J es un E-modelo de $IF(\mathcal{E}^{\sim})$ si y sólo si $T^J_{\mathcal{E}^{\sim}}(I) \subseteq I$.*

DEMOSTRACIÓN. Por la Proposición 4.2.6 puesto que $\mathcal{E}^{\sim}$ e $IF(\mathcal{E}^{\sim})$ son semánticamente equivalentes.

Teorema 4.2.9 *Dada una teoría ecuacional normal $\mathcal{E}^{\sim}$ y una preinterpretación J, una E-interpretación I basada en J es un E-modelo de $ONLY - IF(\mathcal{E}^{\sim})$ si y sólo si $I \subseteq T^J_{\mathcal{E}^{\sim}}(I)$.*

DEMOSTRACIÓN.

$\Longrightarrow$) Si I es un E-modelo de $ONLY - IF(\mathcal{E}^{\sim})$ entonces para toda fórmula, $\forall\, x_1, \ldots, x_n, w(f(x_1, \ldots, x_n) = w \Rightarrow F_1 \vee \ldots \vee F_m \vee (w \equiv f(x_1, \ldots, x_n)))$, de $ONLY - IF(\mathcal{E}^{\sim})$ y para toda asignación σ con respecto a I, $\langle f(x_1, \ldots, x_n)\sigma, w\sigma\rangle \in I$ implica que $I \models_\sigma F_1 \vee \ldots \vee F_m \vee (w \equiv f(x_1, \ldots, x_n))$. Supongamos que $\langle f(x_1, \ldots, x_n)\sigma, w\sigma\rangle \in I$. Distinguimos dos casos:

- Si $w\sigma \equiv f(x_1, \ldots, x_n)\sigma$ entonces $f(x_1, \ldots, x_n)\sigma = w\sigma$ es una instancia del axioma reflexivo con lo que

  $$\langle f(x_1, \ldots, x_n)\sigma, w\sigma\rangle \in T^J_{\mathcal{E}^{\sim}}(I)$$

  por la definición de $T^J_{\mathcal{E}^{\sim}}$.

- Sea $c : (f(t_1, \ldots, t_n) \rightarrow t \Leftarrow \tilde{E}, \tilde{D}.)$ una cláusula en $\mathcal{E}^{\sim}$, supongamos que F_i es de la forma $\exists y_1, \ldots, y_k\, ((x_1 = t_1) \wedge \ldots \wedge (x_n = t_n) \wedge (w = t) \wedge \tilde{E} \wedge \tilde{D})$, siendo $y_1, \ldots, y_k$ las variables de c, y que $I \models_\sigma F_i$. Consideremos una asignación τ con respecto a I tal que $x_i\sigma = t_i\tau (1 \leq i \leq n)$, $w\sigma = t\tau$, $\tilde{E}\sigma = \tilde{E}\tau$ y $\tilde{D}\sigma = \tilde{D}\tau$. Entonces $\langle f(t_1, \ldots, t_n)\tau, t\tau\rangle \in I, \dot{E}\tau \subseteq I, \dot{D}\tau \subseteq B_J \setminus I$ y, además, $\langle t\tau, t\tau\rangle \in I$ por ser I reflexiva. Por lo tanto, $\langle f(t_1, \ldots, t_n)\tau, t\tau\rangle \in T^J_{\mathcal{E}^{\sim}}(I)$ reemplazando el término $t\tau$ a la ocurrencia 1 en el par $\langle t\tau, t\tau\rangle$.

$\Longleftarrow$) Sea $c : (f(t_1, \ldots, t_n) \to t \Leftarrow \tilde{E}, \tilde{D}.)$ una cláusula en $\mathcal{E}^{\sim}$ y supongamos que $\langle f(t_1, \ldots, t_n)\tau, t\tau \rangle \in I$ para alguna asignación τ con respecto a I. Por la hipótesis, $\langle f(t_1, \ldots, t_n)\tau, t\tau \rangle \in T^J_{\mathcal{E}^{\sim}}(I)$. Consideremos la fórmula en $ONLY - IF(\mathcal{E}^{\sim})$ correspondiente al símbolo de función f

$$\forall\, x_1, \ldots, x_n, w(f(x_1, \ldots, x_n) = w \Rightarrow \quad F_1 \vee \ldots \vee F_m \vee \\ (w \equiv f(x_1, \ldots, x_n)))$$

Distinguimos dos casos:

- $f(t_1, \ldots, t_n)\tau$ y $t\tau$ son términos idénticos. Sea σ una asignación con respecto a I tal que $w\tau = w\sigma$ y $f(t_1, \ldots, t_n)\tau = f(x_1, \ldots, x_n)\sigma$. Entonces $\langle f(x_1, \ldots, x_n)\sigma, w\sigma \rangle \in I$ y $I \models_\sigma w \equiv f(x_1, \ldots, x_n)$.
- Se cumple que $\dot{E}\tau \subseteq I$ y $\dot{D}\tau \subseteq B_J \setminus I$ y supongamos que F_i es de la forma $\exists y_1, \ldots, y_k\, ((x_1 = t_1) \wedge \ldots \wedge (x_n = t_n) \wedge (w = t) \wedge \tilde{E} \wedge \tilde{D})$, siendo $y_1, \ldots, y_k$ las variables en la cláusula c. Tomemos σ como la asignación con respecto a I tal que $x_i\sigma = t_i\tau (1 \le i \le n), w\sigma = t\tau, \tilde{E}\sigma = \tilde{E}\tau$ y $\tilde{D}\sigma = \tilde{D}\tau$. Entonces, $\langle f(x_1, \ldots, x_n)\sigma, w\sigma \rangle \in I$ e $I \models_\sigma F_i$ ya que I satisface el axioma reflexivo.

Por lo tanto, $I \models_\sigma F_1 \vee \ldots \vee F_m \vee (w \equiv f(x_1, \ldots, x_n))$.

Teorema 4.2.10 *Dada una teoría ecuacional normal $\mathcal{E}^{\sim}$ y una preinterpretación J, una E-interpretación I basada en J es un E-modelo de $IFF(\mathcal{E}^{\sim})$ si y sólo si $I = T^J_{\mathcal{E}^{\sim}}(I)$.*

Demostración. Por la Proposición 4.2.8 y el Teorema 4.2.9, I es un E-modelo de $IFF(\mathcal{E}^{\sim})$ ya que $IFF(\mathcal{E}^{\sim})$ es semánticamente equivalente al conjunto de fórmulas $IF(\mathcal{E}^{\sim}) \cup ONLY - IF(\mathcal{E}^{\sim})$.

El teorema anterior caracteriza qué E-interpretaciones son E-modelo de las definiciones 'si y sólo si' de los símbolos de función. Como estamos interesados en trabajar en el dominio de Herbrand, el siguiente teorema establece las condiciones bajo las cuales una E-interpretación de Herbrand es una modelo de $comp(\mathcal{E}^{\sim})$.

Teorema 4.2.11 Teorema de caracterización
Sea $\mathcal{E}^{\sim}$ una teoría ecuacional normal confluente. Una E-interpretación de Herbrand I es un modelo de $comp(\mathcal{E}^{\sim})$ si y sólo si $I = T_{\mathcal{E}^{\sim}}(I)$.

DEMOSTRACIÓN. Por el Teorema 4.2.10, I es un E-modelo de $IFF(\mathcal{E}^{\sim})$. Por ser I una E-interpretación también es modelo de los axiomas de equivalencia de la igualdad. Por tanto sólo tenemos que probar que I es un modelo de los axiomas de libertad de C_{ET} para la igualdad.

* Axioma e) de C_{ET} (Definición 4.2.2).
  Asumimos que $I = T_{\mathcal{E}^{\sim}}(I)$. Sean f y g dos símbolos de función irreducibles de aridades n y m respectivamente. Supongamos que I no cumple el axioma e) de C_{ET}. Esto significa que

  $$\langle f(t_1, \ldots, t_n), g(s_1, \ldots, s_m)\rangle \in I$$

  Pero entonces

  $$\langle f(t_1, \ldots, t_n), g(s_1, \ldots, s_m)\rangle \notin T_{\mathcal{E}^{\sim}}(I)$$

  porque $f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)$ no es una instancia del axioma reflexivo ni puede obtenerse a partir de ninguna cláusula de $\mathcal{E}^{\sim}$ por ser f y g símbolos de función irreducibles y $\mathcal{E}^{\sim}$ confluente. Por lo tanto, $\langle f(t_1, \ldots, t_n), g(s_1, \ldots, s_m)\rangle \notin I$ e I satisface el axioma e) de C_{ET}.

* Axioma f) de C_{ET} (Definición 4.2.2).
  Asumimos que $I = T_{\mathcal{E}^{\sim}}(I)$. Sea f un símbolo de función irreducible de aridad n, y supongamos que

  $$\langle f(t_1, \ldots, t_n), f(s_1, \ldots, s_n)\rangle \in I$$

  Por la definición de $T_{\mathcal{E}^{\sim}}$ distinguimos dos casos:

– Si $f(t_1,\ldots,t_n) = f(s_1,\ldots,s_n)$ es una instancia del axioma reflexivo entonces $t_i = s_i, 1 \leq i \leq n$, también lo es y, por consiguiente, $\langle t_i, s_i\rangle \in I, 1 \leq i \leq n$.

– En cualquier otro caso, como f es irreducible si

$$\langle f(t_1,\ldots,t_n), f(s_1,\ldots,s_n)\rangle \in T_{\mathcal{E}^\sim}(I)$$

es porque existen n instancias básicas de cláusulas en $\mathcal{E}^\sim$, $(t_i \to s_i \Leftarrow \tilde{E}_i, \tilde{D}_i.)$ (o, alternativamente, $(s_i \to t_i \Leftarrow \tilde{E}_i, \tilde{D}_i.)$) tales que $\dot{E}_i \subseteq I$ y $\dot{D}_i \subseteq B(\mathcal{E}^\sim)\backslash I$. Pero entonces, $\langle t_i, s_i\rangle \in I$, e I satisface el axioma f).

* Axioma g) de C_{ET} (Definición 4.2.2).
Asumimos que $I = T_{\mathcal{E}^\sim}(I)$ y supongamos que para una valuación $[\![x]\!]$ de x en $\mathcal{T}(C)$ se cumple que el par $\langle [\![x]\!], [\![t]\!]\rangle \in I$. Pero entonces

$$\langle [\![x]\!], [\![t]\!]\rangle \notin T_{\mathcal{E}^\sim}(I)$$

porque ni es una instancia del axioma reflexivo ni puede obtenerse a partir de ninguna cláusula de $\mathcal{E}^\sim$ por ser todos los símbolos de función en $[\![x]\!]$ y en $[\![t]\!]$ irreducibles y $\mathcal{E}^\sim$ confluente. Luego, $\langle [\![x]\!], [\![t]\!]\rangle \notin I$ e I satisface el axioma g) de C_{ET}.

El principal inconveniente cuando se trabaja en el dominio de Herbrand con la compleción de una teoría ecuacional normal es que, siendo consistente la teoría normal, su compleción puede no serlo. Por ejemplo, la siguiente teoría ecuacional normal:

$$\mathcal{E}^\sim = \{a = b \Leftarrow a \neq b\}$$

es consistente mientras que su compleción no lo es ya que $comp(\mathcal{E}^\sim)$ sólo contiene la fórmula $\forall w(a = w \iff (w = b \wedge a \neq b) \vee (w \equiv a))$ que cuando w toma el valor b se reduce a $a = b \iff a \neq b$. Una situación similar se presenta en ciertas teorías ecuacionales no confluentes, como por ejemplo $\mathcal{E}^\sim = \{a = b., \;\; a = c.\}$ ya que el par $\langle b, c\rangle$ debería estar en cualquier modelo de $comp(\mathcal{E}^\sim)$ para cumplir la propiedad transitiva de

C_{ET} pero, si esto ocurre, el axioma e) de C_{ET} no se cumple (b y c son dos símbolos de función irreducibles). En programación lógica una de las posibles formas de resolver este problema consiste en imponer ciertas restricciones sintácticas que garantizan la consistencia del programa completado. En la siguiente sección definimos restricciones similares para las teorías ecuacionales normales.

4.3 Teorías ecuacionales estratificadas

En esta sección proponemos algunas restricciones sintácticas, similares a las propuestas en programación lógica [7], suficientes para garantizar la consistencia de la compleción de una teoría ecuacional normal. La idea básica es limitar el uso de la negación impidiendo la recursión por medio de la negación. Llamaremos estratificadas a aquellas teorías ecuacionales normales confluentes que cumplen estas condiciones.

Informalmente, en una teoría ecuacional estratificada primero se definen algunos símbolos de función en términos de ellos mismos y sin el uso de la negación y, después, pueden definirse nuevos símbolos de función en términos de ellos mismos sin el uso de la negación y en términos de los símbolos de función previos, posiblemente usando la negación.

Definición 4.3.1 Teoría ecuacional estratificada

Una teoría ecuacional normal confluente $\mathcal{E}^\sim$ es estratificada si existe una partición

$$\mathcal{E}^\sim = \mathcal{E}_1^\sim \,\dot\cup\, \ldots \,\dot\cup\, \mathcal{E}_n^\sim$$

tal que las dos condiciones siguientes se cumplen para $i = 1, \ldots, n$:

a) Si una ecuación $u = v$ ocurre en la cabeza o en el cuerpo de una cláusula de $\mathcal{E}_i^\sim$, entonces la definición de todos los símbolos de función definidos que aparecen en u y v está contenida en $\bigcup_{j \leq i} \mathcal{E}_j^\sim$.

b) Si una disecuación $u \neq v$ ocurre en el cuerpo de una cláusula de $\mathcal{E}_i^{\sim}$, entonces la definición de todos los símbolos de función definidos que aparecen en u y v está contenida en $\bigcup_{j<i} \mathcal{E}_j^{\sim}$.

Decimos que $\mathcal{E}^{\sim}$ está estratificada por $\mathcal{E}_1^{\sim} \dot{\cup} \ldots \dot{\cup} \mathcal{E}_n^{\sim}$ y que cada $\mathcal{E}_i^{\sim}$ es un estrato ecuacional de $\mathcal{E}^{\sim}$. Un estrato ecuacional define nuevos símbolos de función en términos de ellos mismos (positivamente) y en términos de los símbolos de función de los estratos previos (posiblemente de forma negativa).

Ejemplo 12 *Sea $\mathcal{E}^{\sim}$ la siguiente teoría ecuacional normal:*

$$\begin{array}{l} r = a. \\ q = b \Leftarrow r \neq b. \\ p = a \Leftarrow q \neq a. \end{array}$$

$\mathcal{E}^{\sim}$ está estratificada por $\mathcal{E}^{\sim} = \{r = a.\} \cup \{q = b \Leftarrow r \neq b.\} \cup \{p = a \Leftarrow q \neq a.\}$.

Ejemplo 13 *La teoría ecuacional normal del final de la sección anterior cuya compleción es inconsistente:*

$$\mathcal{E}^{\sim} = \{a = b \Leftarrow a \neq b.\}$$

no es estratificada porque no satisface la condición b).

4.4 Semántica declarativa de las teorías ecuacionales estratificadas

En esta sección damos una caracterización por punto fijo de los E-modelos de Herbrand de las teorías ecuacionales estratificadas. Puesto que el operador de consecuencias inmediatas es no monótono, aplicaremos la teoría del punto fijo para operadores no monótonos definida por Apt, Blair y Walker [7]. Seguiremos la presentación en [6, 7] para programas lógicos estratificados, adaptándola al caso ecuacional.

Como mencionamos en la Sección 4.1, una teoría ecuacional normal puede tener varios E-modelos minimales. Por ejemplo [85], la teoría $\mathcal{E}^{\sim} = \{a = c \Leftarrow a \neq b.\}$ tiene dos E-modelos de Herbrand minimales:

$M_1 = \{\langle a, a\rangle, \langle b, b\rangle, \langle c, c\rangle, \langle a, c\rangle, \langle c, a\rangle\}$ y
$M_2 = \{\langle a, a\rangle, \langle b, b\rangle, \langle c, c\rangle, \langle a, b\rangle, \langle b, a\rangle\}$

No obstante, sólo M_1 corresponde al significado esperado de la teoría $\mathcal{E}^{\sim}$ ya que no hay forma de probar que a es igual a b a partir de la única cláusula de $\mathcal{E}^{\sim}$. Nuestro interés se centra en obtener únicamente estos E-modelos de Herbrand que llamamos *soportados*.

Definición 4.4.1 Modelo de Herbrand soportado

Sea $\mathcal{E}^{\sim}$ una teoría ecuacional normal. Un modelo de Herbrand M de $\mathcal{E}^{\sim}$ es soportado si y sólo si cada par $\langle t, s\rangle$ en M es o bien una instancia del axioma reflexivo (t y s son términos idénticos) o bien existe una instancia básica $(l \to r \Leftarrow \tilde{E}, \tilde{D}.)$ de una cláusula en $\mathcal{E}^{\sim}$ y una ocurrencia $u \in (t = s)$ tal que $(t = s)/u = l$, $\dot{E} \cup \{\langle t, s\rangle[r]_u\} \subseteq M$ y $\dot{D} \subseteq B(\mathcal{E}^{\sim}) \setminus M$.

En el ejemplo anterior sólo M_1 es soportado.

Lema 4.4.2 *Sea $\mathcal{E}^{\sim}$ una teoría ecuacional normal e I un modelo de Herbrand de $\mathcal{E}^{\sim}$. Entonces I es soportado si y sólo si $I \subseteq T_{\mathcal{E}^{\sim}}(I)$.*

Demostración.

$\Longrightarrow$) Directa a partir de la definición del operador $T_{\mathcal{E}^{\sim}}$ (Definición 4.1.6) y de la Definición 4.4.1.

$\Longleftarrow$) Asumimos que $I \subseteq T_{\mathcal{E}^{\sim}}(I)$ y supongamos que I es no soportado. Entonces, por la Definición 4.4.1, existe un par $\langle t, s\rangle \in I$ tal que $(t = s)$ no es una instancia del axioma reflexivo ni existe una instancia básica $(l \to r \Leftarrow \tilde{E}, \tilde{D}.)$ de una cláusula en $\mathcal{E}^{\sim}$ y una ocurrencia $u \in O(t = s)$ tal que $(t = s)/u = l$, $\dot{E} \cup \{\langle t, s\rangle[r]_u\} \subseteq I$ y $\dot{D} \subseteq B(\mathcal{E}^{\sim}) \setminus I$. Pero entonces $\langle t, s\rangle \notin T_{\mathcal{E}^{\sim}}(I)$ lo que contradice la hipótesis.

En lo que sigue estudiamos la semántica de las teorías ecuacionales estratificadas. El significado declarativo de tales teorías viene dado por un E-modelo de Herbrand minimal y soportado. Para estudiar los E-modelos de Herbrand de una teoría ecuacional estratificada $\mathcal{E}^{\sim}$ y su compleción es suficiente considerar los prepuntos fijos y puntos fijos de su operador de consecuencias inmediatas $T_{\mathcal{E}^{\sim}}$. Para ello, revisamos primero la teoría del punto fijo para operadores no monótonos presentada en [7] y revisada en [6].

4.4.1 Operadores no monótonos y sus puntos fijos

Consideremos un retículo arbitrario pero fijo L sobre el que se definen todos los operadores. $\subseteq$ denota la relación de orden en el retículo, mientras que $\emptyset$ denota el elemento mínimo. Definimos las *potencias acumulativas* de un operador T como sigue:

Definición 4.4.3 [6]**Potencias acumulativas de** T

$$
\begin{array}{l}
T \Uparrow 0(I) = I \\
T \Uparrow (n+1)(I) = T(T \Uparrow n(I)) \cup T \Uparrow n(I) \\
T \Uparrow \omega(I) = \bigcup_{n<\omega} T \Uparrow n(I)
\end{array}
$$

Decimos que un operador T es *finitario* si para cada secuencia infinita de elementos de L, $I_0 \subseteq I_1 \subseteq \ldots$, se cumple que:

$$T(\bigcup_{n=0}^{\infty} I_n) \subseteq \bigcup_{n=0}^{\infty} T(I_n)$$

Así, si $A \in T(\bigcup_{n=0}^{\infty} I_n)$ entonces para algún n, $A \in T(I_n)$, lo que explica el nombre de finitario.

Lema 4.4.4 [6] *Si T es finitario entonces para todo $I \in L$ se cumple que*

$$T(T \Uparrow \omega(I)) \subseteq T \Uparrow \omega(I)$$

El lema anterior establece que los operadores finitarios tienen prepuntos fijos que pueden computarse, de forma natural, como la aplicación acumulativa de T hasta el primer ordinal infinito. Los operadores finitarios no tienen, en general, puntos fijos aunque sí los pueden tener bajo ciertas asunciones.

Decimos que un operador T es *creciente* si para todo $I, J, M \in L$, $I \subseteq J \subseteq M \subseteq T \Uparrow \omega(I)$ implica que $T(J) \subseteq T(M)$. Creciente es una forma restringida de monotonía.

Lema 4.4.5 [6] *Si T es creciente entonces para todo $I \in L$ se cumple que*

$$T \Uparrow \omega(I) \subseteq I \cup T(T \Uparrow \omega(I))$$

Corolario 4.4.6 [6] *Sea T un operador finitario y creciente. Entonces*

$$T \Uparrow \omega(\emptyset) = T(T \Uparrow \omega(\emptyset))$$

Así pues, para un operador T finitario y creciente, $T \Uparrow \omega(\emptyset)$ es un punto fijo.

Ahora estudiamos familias de operadores. Sean $T_1, \ldots, T_n$ operadores. Consideremos la secuencia siguiente:

$$\begin{aligned}
&N_0 = I \\
&N_1 = T \Uparrow \omega(N_0) \\
&\vdots \\
&N_n = T \Uparrow \omega(N_{n-1})
\end{aligned}$$

para la que claramente se cumple que $N_0 \subseteq N_1 \subseteq \cdots \subseteq N_n$.

Sea T la unión de los operadores $T_1, \ldots, T_n$, es decir, el operador definido como

$$T(X) = \bigcup_{i=1}^{n} T_i(X)$$

Para determinar bajo qué condiciones N_n es un punto fijo de T, primero introducimos el concepto de localidad. Decimos que una secuencia $T_1, \ldots, T_n$ es *local* si para todo $I, J \in L$ se cumple que:

$$I \subseteq J \subseteq N_n \text{ implica que } T_i(J) = T_i(J \cap N_i), 1 \leq i \leq n$$

Informalmente, la condición de localidad significa que cada T_i se determina por sus valores sobre los subconjuntos de N_i.

Lema 4.4.7 [6] *Supongamos que la secuencia $T_1, \ldots, T_n$ es local y que todos los operadores T_i son finitarios. Entonces $T(N_n) \subseteq N_n$.*

Lema 4.4.8 [6] *Supongamos que la secuencia $T_1, \ldots, T_n$ es local y que todos los operadores T_i son crecientes. Entonces $N_n \subseteq I \cup T(N_n)$.*

Corolario 4.4.9 [6] *Supongamos que la secuencia $T_1, \ldots, T_n$ es local y que todos los operadores T_i son finitarios y crecientes. Entonces $N_n = I \cup T(N_n)$.*

Por lo tanto, para una secuencia local $T_1, \ldots, T_n$ de operadores finitarios y crecientes, N_n es un punto fijo de T cuando $I = \emptyset$.

Por último, veamos bajo qué condiciones N_n es un prepunto fijo minimal de T que contiene a I.

Lema 4.4.10 [6] *Supongamos que la secuencia $T_1, \ldots, T_n$ es local y que todos los operadores T_i son crecientes. Supongamos que $I \subseteq J \subseteq N_n$ y que $T(J) \subseteq J$. Entonces $J = N_n$.*

4.4.2 Teoría de modelos para teorías ecuacionales estratificadas

Para definir la semántica de las teorías ecuacionales estratificadas aplicamos algunos de los resultados del apartado anterior. Primero, definimos una secuencia de E-interpretaciones de Herbrand de $\mathcal{E}^\sim$.

Definición 4.4.11 *Sea $\mathcal{E}^\sim$ una teoría ecuacional estratificada por $\mathcal{E}_1^\sim \dot\cup \ldots \dot\cup \mathcal{E}_n^\sim$. Entonces los siguientes conjuntos son E-interpretaciones de Herbrand de $\mathcal{E}^\sim$.*

$$\begin{aligned}
M_1 &= T_{\mathcal{E}_1^\sim} \Uparrow \omega(\emptyset)\\
M_2 &= T_{\mathcal{E}_2^\sim} \Uparrow \omega(M_1)\\
&\vdots\\
M_n &= T_{\mathcal{E}_n^\sim} \Uparrow \omega(M_{n-1})
\end{aligned}$$

La idea es ir construyendo estas E-interpretaciones de forma que M_n es un E-modelo minimal y soportado de $\mathcal{E}^\sim$. Cada M_i se obtiene a partir de M_{i-1} y usando las instancias básicas sobre $\mathcal{T}(C \cup F_{\mathcal{E}_1^\sim} \cup \ldots \cup F_{\mathcal{E}_i^\sim})$ de las cláusulas en el estrato $\mathcal{E}_i^\sim$. Para garantizar que M_n es un E-modelo de $\mathcal{E}^\sim$ deberemos imponer otra restricción adicional a la teoría ecuacional estratificada, como evidenciamos con el ejemplo siguiente:

Ejemplo 14 *Sea $\mathcal{E}^\sim$ una teoría ecuacional estratificada por*

$$\begin{array}{rl} \mathcal{E}_1^\sim = \{ & c(a) = a. \\ & c(b) = b.\} \\ \mathcal{E}_2^\sim = \{ & f(x) = x \Leftarrow c(x) \neq a.\} \\ \mathcal{E}_3^\sim = \{ & g(x) = x \Leftarrow c(x) \neq b.\} \end{array}$$

Entonces, el par $\langle f(g(b)), g(b)\rangle \notin M_n$ cuando la ecuación $f(g(b)) = g(b)$ es consecuencia lógica de $\mathcal{E}^\sim$.

El problema del ejemplo anterior es debido a que existen instancias básicas de las cláusulas en la teoría ecuacional que no se tienen en cuenta en la construcción de la secuencia $M_1, \ldots, M_n$. Este problema tampoco se resuelve si construimos $M_1, \ldots, M_n$ usando las instancias básicas sobre $\mathcal{T}(\Sigma)$ de las cláusulas de la teoría ya que en este caso M_n podría contener pares que no fueran consecuencia lógica de la teoría. Siguiendo con el ejemplo anterior y tomando las instancias básicas de las cláusulas sobre $\mathcal{T}(\Sigma)$, obtendríamos, por construcción de M_n que el par $\langle f(g(a)), g(a)\rangle \in M_n$ cuando la ecuación $f(g(a)) = g(a)$ no es consecuencia lógica de $\mathcal{E}^\sim$. El problema queda definitivamente resuelto si imponemos la condición de que la teoría ecuacional sea completamente definida ya que, bajo esta condición, cada una de las E-interpretaciones M_i de la secuencia en la Definición 4.4.11 puede obtenerse a partir de M_{i-1} y considerando únicamente las instancias básicas sobre $\mathcal{T}(C)$ de las cláusulas en $\mathcal{E}_i^\sim$. En lo que resta de capítulo consideraremos una teoría ecuacional normal $\mathcal{E}^\sim$ completamente definida y estratificada por $\mathcal{E}^\sim = \mathcal{E}_1^\sim \,\dot{\cup}\, \ldots \,\dot{\cup}\, \mathcal{E}_n^\sim$, y así lo asumiremos aunque explícitamente no lo indiquemos.

Puesto que la teoría ecuacional es completamente definida, $T_{\mathcal{E}^\sim}$ es la unión de los operadores $T_{\mathcal{E}_1^\sim}, \ldots, T_{\mathcal{E}_n^\sim}$. En lo que sigue demostramos que $M_{\mathcal{E}^\sim} = M_n$ (el E-modelo minimal y soportado de $\mathcal{E}^\sim$). Para ello primero probamos que cada uno de los operadores $T_{\mathcal{E}_i^\sim}$ es finitario y creciente.

Lema 4.4.12 *Para cada estrato ecuacional* $\mathcal{E}_i^{\sim}, 1 \leq i \leq n$, $T_{\mathcal{E}_i^{\sim}}$ *es finitario.*

DEMOSTRACIÓN. Tenemos que probar que para cada secuencia infinita $I_0 \subseteq I_1 \subseteq \ldots$,

$$T_{\mathcal{E}_i^{\sim}}(\bigcup_{n=0}^{\infty} I_n) \subseteq \bigcup_{n=0}^{\infty} T_{\mathcal{E}_i^{\sim}}(I_n)$$

Si $\langle t, s\rangle \in T_{\mathcal{E}_i^{\sim}}(\bigcup_{n=0}^{\infty} I_n)$ entonces para algún n, $\langle t, s\rangle \in T_{\mathcal{E}_i^{\sim}}(I_n)$. De aquí que, $\langle t, s\rangle \in \bigcup_{n=0}^{\infty} T_{\mathcal{E}_i^{\sim}}(I_n)$.

Lema 4.4.13 *Consideremos un estrato ecuacional* $\mathcal{E}_i^{\sim}(1 \leq i \leq n)$. *Entonces,* $T_{\mathcal{E}_i^{\sim}}$ *considerado como un operador en el retículo completo* $L = \{I : I \subseteq B(\mathcal{E}^{\sim})\}$ *ordenado por la inclusión de conjuntos es creciente.*

DEMOSTRACIÓN. Asumimos que para alguna $I \subseteq B(\mathcal{E}^{\sim}), I \subseteq J \subseteq N \subseteq T_{\mathcal{E}_i^{\sim}} \Uparrow \omega(I)$ y sea $\langle t, s\rangle \in T_{\mathcal{E}_i^{\sim}}(J)$. Por la definición de $T_{\mathcal{E}_i^{\sim}}$ distinguimos dos casos:

* Si $t = s$ es una instancia básica del axioma reflexivo, entonces $\langle t, s\rangle \in T_{\mathcal{E}_i^{\sim}}(N)$.

* Ahora asumimos que existe una ocurrencia $u \in O(t = s)$ y una instancia básica sobre $\mathcal{T}(C)$, $(l \to r \Leftarrow \tilde{E}, \tilde{D}.)$, de una cláusula en $\mathcal{E}_i^{\sim}$ tal que $(t = s)/u = l$, $\dot{E} \cup \{\langle t, s\rangle[r]_u\} \subseteq J$ y que $\dot{D} \subseteq B(\mathcal{E}^{\sim}) \backslash J$. Como $J \subseteq N$ entonces $\dot{E} \cup \{\langle t, s\rangle[r]_u\} \subseteq N$. Para cualquier disecuación en $\tilde{D}$, digamos $u' \neq v'$, $u' = v'$ no es una instancia básica del axioma reflexivo, $u' = v' \notin I$ y $u' = v'$ no puede construirse usando una instancia básica sobre $\mathcal{T}(C)$ de una cláusula ecuacional normal en $\mathcal{E}_i^{\sim}$ porque $\mathcal{E}_i^{\sim}$ es un estrato ecuacional. Sin embargo, para cualquier interpretación de Herbrand $M \subseteq B(\mathcal{E}^{\sim})$ y una ecuación básica $t' = s'$ si $\langle t', s'\rangle \in T_{\mathcal{E}_i^{\sim}} \Uparrow \omega(M)$ entonces $\langle t', s'\rangle \in M$ o $t' = s'$ es una instancia básica del axioma reflexivo o $t' = s'$ es una ecuación básica construida usando una instancia básica sobre $\mathcal{T}(C)$ de una cláusula ecuacional normal en $\mathcal{E}_i^{\sim}$. Luego, $\langle u', v'\rangle \notin T_{\mathcal{E}_i^{\sim}} \Uparrow \omega(I)$ y $\langle u', v'\rangle \notin N$. Esto implica que $\langle t, s\rangle \in T_{\mathcal{E}_i^{\sim}}(N)$.

Por consiguiente $T_{\mathcal{E}_i^\sim}$ es creciente.

Puesto que cada $T_{\mathcal{E}_i^\sim}$ es finitario y creciente, los Lemas 4.4.4 y 4.4.5 y el Corolario 4.4.6 de la sección anterior se cumplen. Pasemos ahora a estudiar el operador unión de los operadores $T_{\mathcal{E}_i^\sim}$.

Consideremos la secuencia de operadores $T_{\mathcal{E}_1^\sim}, \ldots, T_{\mathcal{E}_n^\sim}$ y sea $T_{\mathcal{E}^\sim}$ la unión de estos operadores. Para demostrar que M_n es un punto fijo de $T_{\mathcal{E}^\sim}$ no podemos aplicar directamente los resultados de la sección anterior porque la secuencia $T_{\mathcal{E}_1^\sim}, \ldots, T_{\mathcal{E}_n^\sim}$ no es local, como mostramos en el siguiente ejemplo.

Ejemplo 15 *Sea $\mathcal{E}^\sim$ una teoría ecuacional estratificada por $\mathcal{E}_1^\sim = \{a = b.\}$ y $\mathcal{E}_2^\sim = \{c = d \Leftarrow a \neq d., \;\; e = b.\}$. Ahora, por la Definición 4.4.11*

$$\begin{array}{rl} M_1 = \{ & \langle a,a\rangle, \langle b,b\rangle, \langle c,c\rangle, \langle d,d\rangle, \langle e,e\rangle, \langle a,b\rangle, \langle b,a\rangle \; \} \\ M_2 = \{ & \langle a,a\rangle, \langle b,b\rangle, \langle c,c\rangle, \langle d,d\rangle, \langle e,e\rangle, \langle a,b\rangle, \langle b,a\rangle, \\ & \langle c,d\rangle, \langle d,c\rangle, \langle e,b\rangle, \langle b,e\rangle, \langle a,e\rangle, \langle e,a\rangle \; \} \end{array}$$

Tomemos la siguiente interpretación $J = \{\langle a,a\rangle, \langle b,b\rangle, \langle e,b\rangle\}$ que claramente cumple $J \subseteq M_2$. De ser local la secuencia $T_{\mathcal{E}_1^\sim}, T_{\mathcal{E}_2^\sim}$, tendría que cumplirse que $T_{\mathcal{E}_i^\sim}(J) = T_{\mathcal{E}_i^\sim}(J \cap M_i), 1 \leq i \leq 2$. Sin embargo, para $i = 1$ tenemos que:

$$\begin{array}{l} T_{\mathcal{E}_1^\sim}(J) = \{\langle a,a\rangle, \langle b,b\rangle, \langle c,c\rangle, \langle d,d\rangle, \langle e,e\rangle, \langle a,b\rangle, \langle b,a\rangle, \langle e,a\rangle\} \\ T_{\mathcal{E}_1^\sim}(J \cap M_1) = \{\langle a,a\rangle, \langle b,b\rangle, \langle c,c\rangle, \langle d,d\rangle, \langle e,e\rangle, \langle a,b\rangle, \langle b,a\rangle\} \end{array}$$

Luego $T_{\mathcal{E}_1^\sim}(J) \neq T_{\mathcal{E}_1^\sim}(J \cap M_1)$.

No obstante, a pesar de que la secuencia de operadores $T_{\mathcal{E}_1^\sim}, \ldots, T_{\mathcal{E}_n^\sim}$ no es local, M_n es un punto fijo de $T_{\mathcal{E}^\sim}$. Para probarlo necesitamos unos lemas previos.

Lema 4.4.14 *Considerar los estratos ecuacionales $\mathcal{E}_1^\sim, \ldots, \mathcal{E}_n^\sim$ y la secuencia de operadores $T_{\mathcal{E}_1^\sim}, \ldots, T_{\mathcal{E}_n^\sim}$. Entonces, $\forall i : 1 \leq i \leq n : T_{\mathcal{E}_i^\sim}(M_n) \subseteq M_n$.*

DEMOSTRACIÓN. Let $\langle t, s\rangle \in T_{\mathcal{E}_i^\sim}(M_n)$. Por la definición de $T_{\mathcal{E}_i^\sim}$ distinguimos dos casos:

a) Si $t = s$ es una instancia básica del axioma reflexivo entonces $\langle t, s\rangle \in M_n$ por construcción de M_n.

b) Asumimos que existe una ocurrencia $u \in O(t = s)$ y una instancia básica sobre $\mathcal{T}(C)$ $(l \to r \Leftarrow \tilde{E}, \tilde{D}.)$ de una cláusula en $\mathcal{E}_i^\sim$ tal que $(t = s)/u = l, \tilde{E} \cup \{\langle t, s\rangle[r]_u\} \subseteq M_n$ y $\bar{D} \subseteq B(\mathcal{E}^\sim) \setminus M_n$. Como $M_i \subseteq M_n$ entonces $\bar{D} \subseteq B(\mathcal{E}^\sim) \setminus M_i$. También, $\tilde{E} \in M_i$ porque $\mathcal{E}_i^\sim$ es un estrato ecuacional. Luego, $\langle l, r\rangle \in M_i$. Ahora, si $\langle t, s\rangle[r]_u \in M_i$, entonces $\langle t, s\rangle \in M_i$ y $\langle t, s\rangle \in M_n$, si no $\langle t, s\rangle[r]_u \in M_j$, para algún $j > i$. Pero $\langle l, r\rangle \in M_j$ ya que $M_i \subseteq M_j$. De aquí que, al ser M_j una E-interpretación, se cumple que $\langle t, s\rangle \in M_j$ por el Lema 4.2.7. Por lo tanto, $\langle t, s\rangle \in M_n$.

Lema 4.4.15 *Considerar los estratos ecuacionales $\mathcal{E}_1^\sim, \ldots, \mathcal{E}_n^\sim$ y la familia de E-interpretaciones $M_1, \ldots, M_n$. Entonces, $\forall i : 1 \leq i \leq n$: $M_i \subseteq T_{\mathcal{E}^\sim}(M_n)$.*

DEMOSTRACIÓN. Si $\langle t, s\rangle \in M_i$ entonces:

a) o $t = s$ es una instancia del axioma reflexivo. Pero entonces $\langle t, s\rangle \in T_{\mathcal{E}^\sim}(M_n)$ por la definición de $T_{\mathcal{E}^\sim}$.

b) o existe una instancia sobre $\mathcal{T}(C)$, $(l \to r \Leftarrow \tilde{E}, \tilde{D}.)$, de una cláusula en $\mathcal{E}_j^\sim, 1 \leq j \leq i$ tal que $\tilde{E} \subseteq M_j$ y $\bar{D} \subseteq B(\mathcal{E}^\sim) \setminus M_j$ y, o $t \equiv l$ y $s \equiv r$ o $\langle l, r\rangle \in M_j$, $\langle t, s\rangle[r]_u \in M_j$ y $(t = s)/u = l$ (Lema 4.2.7). Ahora bien, claramente $\tilde{E} \subseteq M_n$ y también $\bar{D} \subseteq B(\mathcal{E}^\sim) \setminus M_n$ porque $\mathcal{E}_j^\sim$ es un estrato ecuacional. Por lo tanto,

   (a) en el primer caso, $\langle t, s\rangle \in T_{\mathcal{E}^\sim}(M_n)$ por definición de $T_{\mathcal{E}^\sim}$.

   (b) en el segundo caso, como $(t = s)/u = l$ y $\langle t, s\rangle[r]_u \in M_n$ entonces $\langle t, s\rangle \in T_{\mathcal{E}^\sim}(M_n)$.

Estos dos lemas son la base de la demostración de que M_n es un punto fijo de $T_{\mathcal{E}^\sim}$.

Corolario 4.4.16 *Considerar los estratos ecuacionales* $\mathcal{E}_1^\sim, \ldots, \mathcal{E}_n^\sim$ *y la secuencia de operadores* $T_{\mathcal{E}_1^\sim}, \ldots, T_{\mathcal{E}_n^\sim}$. *Entonces* $M_n = T_{\mathcal{E}^\sim}(M_n)$.

Demostración. Primero probamos que $M_n \subseteq T_{\mathcal{E}^\sim}(M_n)$.

Ya que cada $T_{\mathcal{E}_i^\sim}$ es finitario y creciente, tenemos que:

$$
\begin{array}{l}
M_1 = T_{\mathcal{E}_1^\sim}(M_1) \quad \text{(por el Corolario 4.4.6)} \\
M_2 \subseteq T_{\mathcal{E}_1^\sim}(M_1) \cup T_{\mathcal{E}_2^\sim}(M_2) \\
M_3 \subseteq T_{\mathcal{E}_1^\sim}(M_1) \cup T_{\mathcal{E}_2^\sim}(M_2) \cup T_{\mathcal{E}_3^\sim}(M_3) \\
\vdots \\
M_n \subseteq T_{\mathcal{E}_1^\sim}(M_1) \cup \ldots \cup T_{\mathcal{E}_n^\sim}(M_n)
\end{array}
$$

Luego,

$$
\begin{array}{lll}
M_n & \subseteq T_{\mathcal{E}_1^\sim}(M_1) \cup \ldots \cup T_{\mathcal{E}_n^\sim}(M_n) & \\
 & \subseteq M_1 \cup \ldots \cup M_n & \text{por ser } T_{\mathcal{E}_i^\sim} \text{ finitario} \\
 & \subseteq T_{\mathcal{E}^\sim}(M_n) \cup \ldots \cup T_{\mathcal{E}^\sim}(M_n) & \text{por el Lema 4.4.15} \\
 & \subseteq T_{\mathcal{E}^\sim}(M_n) &
\end{array}
$$

Veamos ahora que $T_{\mathcal{E}^\sim}(M_n) \subseteq M_n$. Directamente, por ser $\mathcal{E}^\sim$ completamente definida, $T_{\mathcal{E}^\sim}(M_n) = \bigcup_{i=1}^n T_{\mathcal{E}_i^\sim}(M_n)$ y $\bigcup_{i=1}^n T_{\mathcal{E}_i^\sim}(M_n) \subseteq M_n$ por el Lema 4.4.14.

Por lo tanto, $M_n = T_{\mathcal{E}^\sim}(M_n)$.

El siguiente lema prueba que M_n es un prepunto fijo minimal de $T_{\mathcal{E}^\sim}$.

Lema 4.4.17 *Considerar la secuencia* $T_{\mathcal{E}_1^\sim}, \ldots, T_{\mathcal{E}_n^\sim}$. *Supongamos que* $\emptyset \subseteq J \subseteq M_n$ *y que* $T_{\mathcal{E}^\sim}(J) \subseteq J$. *Entonces* $J = M_n$.

DEMOSTRACIÓN. Probamos por inducción sobre $j = 0, \ldots, n$, que

$$M_j \subseteq J \tag{4.1}$$

Para $j = 0$ esto es parte de las asunciones. Asumimos que la afirmación se cumple para algún $j < n$. Ahora probamos por inducción sobre k que

$$T_{\mathcal{E}^{\sim}_{j+1}} \Uparrow k(M_j) \subseteq J \tag{4.2}$$

Para $k = 0$ esto es justo (4.1). Asumimos que (4.2) se cumple para algún $k \geq 0$. Así pues tenemos que:

* Si $J \subseteq M_{j+1}$ entonces,

  $$T_{\mathcal{E}^{\sim}_{j+1}}(T_{\mathcal{E}^{\sim}_{j+1}} \Uparrow k(M_j)) \subseteq T_{\mathcal{E}^{\sim}_{j+1}}(J) \subseteq T_{\mathcal{E}^{\sim}}(J) \subseteq J$$

  por las asunciones y porque $T_{\mathcal{E}^{\sim}_{j+1}}$ es creciente.

* Si $M_{j+1} \subseteq J$ entonces,

  $$T_{\mathcal{E}^{\sim}_{j+1}}(T_{\mathcal{E}^{\sim}_{j+1}} \Uparrow k(M_j)) \subseteq T_{\mathcal{E}^{\sim}_{j+1}}(M_{j+1}) \subseteq M_{j+1}$$

  porque $T_{\mathcal{E}^{\sim}_{j+1}}$ es creciente y finitario.

Luego,

$$T_{\mathcal{E}^{\sim}_{j+1}}(T_{\mathcal{E}^{\sim}_{j+1}} \Uparrow k(M_j)) \subseteq J \tag{4.3}$$

De aquí que:

$$\begin{array}{rll} T_{\mathcal{E}^{\sim}_{j+1}} \Uparrow (k+1)(M_j) & \subseteq T_{\mathcal{E}^{\sim}_{j+1}}(T_{\mathcal{E}^{\sim}_{j+1}} \Uparrow k(M_j)) \cup J & \text{(por (4.2))} \\ & \subseteq J & \text{(por (4.3))} \end{array}$$

Así pues, por inducción, (4.2) se cumple para todo $k \geq 0$ y, por lo tanto, $M_{j+1} \subseteq J$. Esto prueba (4.1) para todo $j = 0, \ldots, n$ y concluye la demostración.

El siguiente teorema resume los resultados más importantes de esta sección.

Teorema 4.4.18 *Sea* $\mathcal{E}^\sim$ *una teoría ecuacional completamente definida y estratificada por* $\mathcal{E}^\sim = \mathcal{E}_1^\sim \dot{\cup} \cdots \dot{\cup} \mathcal{E}_n^\sim$*. Entonces*

a) $M_{\mathcal{E}^\sim}$ *es un E-modelo de Herbrand soportado de* $\mathcal{E}^\sim$*.*

b) $M_{\mathcal{E}^\sim}$ *es un E-modelo de Herbrand minimal de* $\mathcal{E}^\sim$*.*

c) $M_{\mathcal{E}^\sim}$ *es un modelo de Herbrand de* $comp(\mathcal{E}^\sim)$*.*

DEMOSTRACIÓN.

a) Por el Corolario 4.4.16, $M_n = T_{\mathcal{E}^\sim}(M_n)$. Pero $M_{\mathcal{E}^\sim} = M_n$. Luego, $M_{\mathcal{E}^\sim} = T_{\mathcal{E}^\sim}(M_{\mathcal{E}^\sim})$. Ahora, por la Proposición 4.2.6, $M_{\mathcal{E}^\sim}$ es un E-modelo de Herbrand de $\mathcal{E}^\sim$ y por el Lema 4.4.2 este modelo es soportado.

b) La minimalidad se sigue de *a*), del Lema 4.4.17 y de la Proposición 4.2.6.

c) Por el Corolario 4.4.16 y el Teorema 4.2.10.

Capítulo 5

Equivalencia entre las semánticas: Corrección y Completitud

Uno de los requisitos principales de un buen lenguaje de programación es el de disponer de una semántica definida de un modo lo más sencillo posible y con un buen soporte formal. La forma estándar de definir la semántica de un lenguaje de programación lógico consiste en dar una semántica operacional y una semántica declarativa (teoría de modelos y/o por punto fijo) con resultados de equivalencia entre ambas.

En este capítulo establecemos la equivalencia entre las semánticas operacional y por punto fijo propuestas en los dos capítulos anteriores. En concreto, el resultado que presentamos es la correción y la completitud básica del algoritmo de "narrowing" con negación constructiva con respecto a la compleción de una teoría ecuacional estratificada. Estas propiedades pueden enunciarse informalmente diciendo que todo sistema respuesta computado por "narrowing" con negación constructiva es una respuesta correcta para el objetivo resuelto y sus soluciones hacen dicho objetivo válido en cualquier modelo de la teoría completada (corrección), y que para cualquier sistema ecuacional básico cierto en cualquier modelo de la teoría completada existe una refutación correspondiente haciendo uso del "narrowing" con negación constructiva (completitud).

El capítulo queda organizado en dos secciones, la primera de las cuales está dedicada a la corrección, mientras que en la segunda presentamos el resultado de completitud.

5.1 Corrección

La corrección es relativamente sencilla de demostrar. Se basa, esencialmente, en la corrección de los pasos de derivación del "innermost narrowing", puesto que el "narrowing" con negación constructiva se reduce a "innermost narrowing" estándar cuando se están resolviendo subobjetivos no negados y hace uso del mismo cuando el subobjetivo tratado es negado.

A diferencia de lo que ocurre en programación lógica y ecuacional estándar, para demostrar la corrección de nuestro mecanismo operacional no podemos considerar sólo una rama con éxito del árbol de derivaciones, sino que debemos tener en cuenta todo el árbol ya que la regla de negación constructiva considera fronteras enteras.

El siguiente lema demuestra que podemos reemplazar un objetivo extendido G por la disyunción de los objetivos que aparecen como nodos de su frontera.

Lema 5.1.1 *Si $\mathcal{E}$ es una teoría ecuacional de Horn completamente definida y con disciplina de constructores y $\{G_1, \ldots, G_n\}$ es una frontera de un objetivo extendido G entonces*

$$comp(\mathcal{E}) \models G \longleftrightarrow (\exists \tilde{Y}_1 G_1) \vee \ldots \vee (\exists \tilde{Y}_n G_n)$$

donde $\tilde{Y}_i$ denota el conjunto de variables en G_i que no están en G.

Demostración. El lema se sigue directamente de la Proposición 4.2.3 y del hecho de que si $\{G_1, \ldots, G_n\}$ es una frontera para G entonces cada paso de derivación para G pasa a través de un nodo frontera, ya que el mecanismo operacional usado para construir la frontera ("innermost narrowing") es correcto y completo para las teorías ecuacionales que satisfacen la hipótesis del lema [53].

El siguiente lema generaliza el resultado del lema anterior para teorías estratificadas y constituye la clave para demostrar la corrección.

Lema 5.1.2 *Si $\mathcal{E}^{\sim}$ es una teoría ecuacional estratificada, completamente definida y con disciplina de constructores, G es un objetivo extendido y $\{G_1, \ldots, G_n\}$ es una frontera de G entonces*

$$comp(\mathcal{E}^{\sim}) \models G \longleftrightarrow (\exists \tilde{Y}_1 G_1) \vee \ldots \vee (\exists \tilde{Y}_n G_n)$$

donde $\tilde{Y}_i$ denota el conjunto de variables en G_i que no están en G.

Demostración. Probamos el lema por inducción sobre el número de llamadas a subderivaciones negativas.

En el caso base, el subobjetivo seleccionado de G es un subobjetivo no negado y $\mathcal{E}^{\sim}$ es una teoría ecuacional de Horn (sin negación). Por lo tanto, el lema se cumple por el Lema 5.1.1.

Para establecer el paso de inducción, el único caso comprometido es cuando el subobjetivo seleccionado es un subobjetivo negado $\sim \tilde{\exists}(g \quad E)$. Por la hipótesis de inducción, la frontera $\{(g_1 \quad E_1), \ldots, (g_m \quad E_m)\}$ para la derivación de $(g \quad E)$ es tal que

$$comp(\mathcal{E}^{\sim}) \models (g \quad E) \longleftrightarrow \exists \tilde{Y}_1'(g_1 \quad E_1) \vee \ldots \vee \exists \tilde{Y}_n'(g_m \quad E_m)$$

donde $\tilde{Y}_i'$ denota el conjunto de variables en $(g_i \quad E_i)$ que no están en $(g \quad E)$.

Ahora, si $F_1, \ldots, F_n$ son las fórmulas obtenidas después de negar la frontera de $(g \quad E)$ entonces,

$$comp(\mathcal{E}^{\sim}) \models \sim \tilde{\exists}(g \quad E) \longleftrightarrow F_1 \vee \ldots \vee F_n$$

Por consiguiente,

$$comp(\mathcal{E}^{\sim}) \models G \longleftrightarrow (\exists \tilde{Y}_1 G_1) \vee \ldots \vee (\exists \tilde{Y}_n G_n)$$

ya que cada $\exists \tilde{Y}_i G_i$ se obtiene desde G reemplazando el subobjetivo seleccionado $\sim \tilde{\exists}(g \quad E)$ por F_i.

Luego el lema se cumple también para el caso general construyendo la frontera iterativamente.

A continuación presentamos el resultado de corrección del cálculo $\mathcal{ICN}cn$ de la Sección 3.3.

Teorema 5.1.3 Corrección
Sea $\mathcal{E}^{\sim}$ una teoría ecuacional estratificada, completamente definida y con disciplina de constructores, y g un objetivo normal expresado como el subobjetivo extendido g_0. Si $\langle [\,], [g_0 \quad \emptyset] \rangle \rightarrow_{\mathcal{ICN}cn} \langle S, L \rangle$ entonces cada sistema E_i en la lista S es una respuesta correcta para g_0.

Demostración. Si $\langle [\,], [g_0 \quad \emptyset] \rangle \rightarrow_{\mathcal{ICN}cn} \langle S, L \rangle$ entonces para cada sistema en la lista S existe una derivación de "narrowing" con éxito en el árbol de derivaciones para $(g_0 \quad \emptyset)$ de la forma:

$$(g_0 \quad \emptyset), (\exists \tilde{Y}_1(g_1 \quad E_1)), \ldots, (\exists \tilde{Y}_{n-1}(g_{n-1} \quad E_{n-1})), (\exists \tilde{Y}_n(\emptyset \quad E_n))$$

tal que cada $\exists \tilde{Y}_k(g_k \quad E_k)$ pertenece a la frontera de $\exists \tilde{Y}_{k-1}(g_{k-1} \quad E_{k-1})$ y $\exists \tilde{Y}_n E_n \in S$. Sea $F_i, 0 \leq i \leq n-1$, una frontera de $\exists \tilde{Y}_i(g_i \quad E_i)$. Entonces por el Lema 5.1.2

$$\begin{array}{rl}
comp(\mathcal{E}^{\sim}) \models & (g_0 \quad \emptyset) \longleftrightarrow F_0 \\
comp(\mathcal{E}^{\sim}) \models & \exists \tilde{Y}_1(g_1 \quad E_1) \longleftrightarrow F_1 \\
 & \vdots \\
comp(\mathcal{E}^{\sim}) \models & \exists \tilde{Y}_{n-1}(g_{n-1} \quad E_{n-1}) \longleftrightarrow G_1 \vee \ldots \vee \exists \, \tilde{Y}_n \, (\emptyset \quad E_n) \\
 & \vee \ldots \vee \; G_p
\end{array}$$

Ahora, $\exists \, \tilde{Y}_n \, E_n$ es satisfacible por pertenecer a S. Por lo tanto, para toda sustitución θ tal que $C_{ET} \models \exists \, \tilde{Y}_n \, (E_n \theta)$, la disyunción $G_1 \vee \ldots \vee \exists \, \tilde{Y}_n \, (\emptyset \quad E_n) \vee \ldots \vee G_p$ es cierta. Luego, por las co-implicaciones en las fórmulas anteriores y porque cada $\exists \tilde{Y}_j(g_j \quad E_j) \in F_{j-1}$, cada respuesta computada $E_i \in S$ es una respuesta correcta para $(g_0 \quad \emptyset)$.

5.2 Completitud

En esta sección presentamos el resultado de completitud básica de nuestro mecanismo operacional. Al igual que para la corrección, para demostrar la completitud debemos considerar todo el árbol de derivaciones y no sólo las ramas de éxito. Para ello definimos el concepto de *éxito total*, una noción dual a la de fallo finito.

Definición 5.2.1 Exito Total

Sea $(g \quad E)$ *un objetivo con* g *un objetivo extendido y* E *un sistema complejo. El* $\mathcal{ICN}cn$*-árbol de derivaciones para* $(g \quad E)$ *es de éxito total si y sólo si todas sus derivaciones terminan en nodos* $(\emptyset \quad E_1), \ldots,$ $(\emptyset \quad E_n)$ *tales que* $C_{ET} \models E_i$*.*

La definición anterior establece que si $(g \quad E)$ tiene un $\mathcal{ICN}cn$-árbol de derivaciones de éxito total y todas las derivaciones terminan en nodos $(\emptyset \quad E_1), \ldots, (\emptyset \quad E_n)$, entonces

$$comp(\mathcal{E}^{\sim}) \models (g \quad E) \longrightarrow \tilde{\exists} E_1 \vee \ldots \vee \tilde{\exists} E_n$$

donde $\tilde{\exists} E_i$ denota que todas las variables que están en E_i que no pertenecen a $(g \quad E)$ están cuantificadas existencialmente.

Análogamente, podemos expresar el concepto de fallo finito, siguiendo la misma notación utilizada en la definición de éxito total.

Definición 5.2.2 Fallo Finito

Sea $(g \quad E)$ *un objetivo con* g *un objetivo extendido y* E *un sistema complejo. El* $\mathcal{ICN}cn$*-árbol de derivaciones para* $(g \quad E)$ *es fallado finitamente si y sólo si existe un cierto número natural* n *tal que cada derivación en el* $\mathcal{ICN}cn$*-árbol conduce a un nodo* $(g' \quad E')$ *en un número de pasos igual o inferior a* n *para el que se cumple que*

$$comp(\mathcal{E}^{\sim}) \models (g \quad E) \longrightarrow \sim \tilde{\exists} E'$$

El éxito total y el fallo finito están interrrelacionados, como mostramos a continuación. Esta correspondencia es en parte debida a la relación que existe entre el $\mathcal{ICN}cn$-árbol de derivaciones para un subobjetivo negado y el de su subobjetivo complementario.

Lema 5.2.3 *Cada derivación para* $(\sim (g \quad E) \quad \emptyset)$ *es fallada finitamente, termina con éxito o en algún paso de la derivación, para cada subobjetivo negado que aparece en la derivación su versión positiva pertenece a una derivación para* $(g \quad E)$*.*

Demostración. Sea $\{(g_1 \;\; E_1), \ldots, (g_n \;\; E_n)\}$ una frontera para el subobjetivo extendido $(\sim (g \;\; E) \;\; \emptyset)$, obtenida a partir de la frontera $\{(g'_1 \;\; E'_1), \ldots, (g'_m \;\; E'_m)\}$ de $(g \;\; E)$. Si $g_i, 1 \leq i \leq n$, es $\emptyset$ entonces o bien la derivación que contiene a $(g_i \;\; E_i)$ termina con éxito si el sistema $\tilde{\exists} E_i$ es satisfacible, o bien falla finitamente si este sistema es insatisfacible. El resto de los nodos de la frontera se construyen de forma que $g_i, 1 \leq i \leq n$, es una conjunción de subobjetivos extendidos de la forma $\sim \exists \tilde{Y}_j (g'_j \;\; E'_j), 1 \leq j \leq m$. Luego $(g'_j \;\; E'_j)$ pertenece a una derivación para $(g \;\; E)$.

Lema 5.2.4 *Sea $(g \;\; E)$ un objetivo. El $\mathcal{ICN}cn$-árbol de derivaciones para $(g \;\; E)$ es de éxito total si y sólo si el $\mathcal{ICN}cn$-árbol de derivaciones para $(\sim (g \;\; E) \;\; \emptyset)$ es fallado finitamente.*

Demostración.

$\Longleftarrow$) Ya que el $\mathcal{ICN}cn$-árbol de derivaciones para $(g \;\; E)$ es de éxito total, por la Definición 5.2.1 todas las derivaciones terminan en nodos $(\emptyset \;\; E_1), \ldots, (\emptyset \;\; E_n)$ tal que se cumple que

$$C_{ET} \models E_i$$

y

$$comp(\mathcal{E}^{\sim}) \models (g \;\; E) \longrightarrow \tilde{\exists} E_1 \vee \ldots \vee \tilde{\exists} E_n$$

Ahora bien, por el Lema 5.2.3, cualquier derivación para $(\sim (g \;\; \emptyset) \;\; \emptyset)$ es o bien a) fallada finitamente, o b) de éxito o c) conduce a un nodo tal que la versión positiva de cada subobjetivo negado en la parte objetivo del mismo pertenece a una derivación (incluyendo las de éxito) en el $\mathcal{ICN}cn$-árbol de derivaciones para $(g \;\; E)$. En el caso b), la derivación termina en un objetivo de la forma $(\emptyset \;\; \sim E'_1 \wedge \ldots \wedge \sim E'_m)$ tal que cada $E'_i, 1 \leq i \leq m$, es la parte restricción de un nodo del $\mathcal{ICN}cn$-árbol de derivaciones para $(g \;\; E)$. Ahora, para cada $i, 1 \leq i \leq n$, existe E'_j tal que

$\sim \tilde{\exists} E'_j \longrightarrow \sim \tilde{\exists} E_i$. Por lo tanto, $\sim E'_1 \wedge \ldots \wedge \sim E'_m$ es insatisfacible, lo cual es una contradicción. Para el caso c) la derivación conduce a un nodo que cumple las mismas condiciones que en a). Luego todas las derivaciones para $(\sim (g \quad E) \quad \emptyset)$ son falladas finitamente.

$\Longrightarrow$) Si $(\sim (g \quad E) \quad \emptyset)$ tiene un $\mathcal{ICN}cn$-árbol de derivaciones fallado finitamente, consideremos las derivaciones que conducen a nodos de la forma $(\emptyset \quad E_1), \ldots, (\emptyset \quad E_n)$ con $E_i = \sim E'_1 \wedge \ldots \wedge \sim E'_m$. Nótese que $(\emptyset \quad E'_1), \ldots, (\emptyset \quad E'_m)$ son nodos de éxito en el $\mathcal{ICN}cn$-árbol de derivaciones para $(g \quad E)$. Ahora, cada $\tilde{\exists} E_i$ es insatisfacible, de aquí que

$$comp(\mathcal{E}^{\sim}) \models (g \quad E) \longrightarrow \tilde{\exists} E'_1 \vee \ldots \vee \tilde{\exists} E_m$$

y $(g \quad E)$ tiene un $\mathcal{ICN}cn$-árbol de derivaciones de éxito total.

Lema 5.2.5 *Sea $(g \quad E)$ un objetivo. El $\mathcal{ICN}cn$-árbol de derivaciones para $(g \quad E)$ es fallado finitamente si y sólo si el $\mathcal{ICN}cn$-árbol de derivaciones para $(\sim (g \quad E) \quad \emptyset)$ es de éxito total.*

DEMOSTRACIÓN.

$\Longrightarrow$) Ya que el $\mathcal{ICN}cn$-árbol de derivaciones para $(\sim (g \quad E) \quad \emptyset)$ es de éxito total existen derivaciones exitosas $(\emptyset \quad (\sim \tilde{\exists} E_1^1 \wedge \ldots \wedge \sim \tilde{\exists} E_1^{n_1})), \ldots, (\emptyset \quad (\sim \tilde{\exists} E_m^1 \wedge \ldots \wedge \sim \tilde{\exists} E_m^{n_m}))$ tal que

$$\begin{aligned} comp(\mathcal{E}^{\sim}) \models (\sim (g \quad E) \quad \emptyset) \longrightarrow \quad & (\sim \tilde{\exists} E_1^1 \wedge \ldots \wedge \sim \tilde{\exists} E_1^{n_1}) \\ & \vee \ldots \vee \\ & (\sim \tilde{\exists} E_m^1 \wedge \ldots \wedge \sim \tilde{\exists} E_m^{n_m}) \end{aligned}$$

y

$$C_{ET} \models \sim \tilde{\exists} E_i^1 \wedge \ldots \wedge \sim \tilde{\exists} E_i^{n_i}$$

Cada sistema E_i^j pertenece a la parte restricción de un nodo del $\mathcal{ICN}cn$-árbol de derivaciones para $(g \quad E)$. Por lo tanto, todas las derivaciones que terminen en nodos de la forma $(\emptyset \quad E')$ tendrán sistemas E' insatisfacibles y el $\mathcal{ICN}cn$-árbol de derivaciones es, por lo tanto, fallado finitamente.

$\Longleftarrow$) Ya que el $\mathcal{ICN}cn$-árbol de derivaciones para $(g \quad E)$ es fallado finitamente, consideremos aquellos nodos de la forma $(\emptyset \quad E_1), \ldots, (\emptyset \quad E_n)$ para los que se cumple que

$$comp(\mathcal{E}^\sim) \models (g \quad E) \longrightarrow \sim \tilde{\exists} E_i (1 \leq i \leq n)$$

Ahora bien, ya que este árbol de derivaciones es fallado finitamente, las ramas del $\mathcal{ICN}cn$-árbol de derivaciones para $(\sim (g \quad E) \quad \emptyset)$ terminarán en nodos de la forma $(\emptyset \quad E'_1), \ldots, (\emptyset \quad E'_m)$ tal que $\forall i : 1 \leq i \leq n, \exists j : 1 \leq j \leq m : \sim \tilde{\exists} E_i \longrightarrow \tilde{\exists} E'_j$. Luego,

$$comp(\mathcal{E}^\sim) \models (\sim (g \quad E) \quad \emptyset) \longrightarrow \tilde{\exists} E'_1 \vee \ldots \vee \tilde{\exists} E'_m$$

y el $\mathcal{ICN}cn$-árbol de derivaciones para $(\sim (g \quad E) \quad \emptyset)$ es de éxito total.

La definición siguiente permite establecer una relación análoga a la que existe para una teoría ecuacional de Horn $\mathcal{E}$ entre el conjunto de éxitos del "innermost narrowing" y el punto fijo de $T_\mathcal{E}$, $T_\mathcal{E} \uparrow \omega\, (\emptyset)$.

Definición 5.2.6 *Sea $\mathcal{E}^\sim$ una teoría ecuacional estratificada. Un símbolo de función definido f de $\mathcal{E}^\sim$ es de nivel k si la definición de f está contenida en el estrato k.*

Lema 5.2.7 *Sea $\mathcal{E}^\sim$ una teoría ecuacional estratificada, completamente definida y con disciplina de constructores, y sea $t = s$ un subobjetivo básico no negado. Supongamos que el nivel máximo de los símbolos de función en $\mathcal{E}^\sim$ es $k \geq 1$. Si $\langle t, s \rangle \in T_{\mathcal{E}_k^\sim} \Uparrow \omega(M_{k-1})$ entonces existe una $\mathcal{ICN}cn$-derivación para $(t = s \quad \emptyset)$ que conduce a un nodo $(g \quad E_g)$ con las siguientes propiedades:*

i) para cada subobjetivo básico no negado $t' = s' \in g$ se cumple que $\langle t', s' \rangle \in M_{k-1}$.

ii) para cada subobjetivo básico negado $\sim (g' \;\; E') \in g$ se cumple que al menos uno de los subobjetivos básicos no negados que contiene $t' = s' \in g'$ es tal que $\langle t', s' \rangle \notin M_{k-1}$.

DEMOSTRACIÓN. Probamos el lema por inducción sobre k.

Caso base: Sea $k = 1$. Entonces $\mathcal{E}^{\sim}$ es una teoría ecuacional de Horn, completamente definida y con disciplina de constructores. Luego el lema se sigue de la completitud del "innermost narrowing".

Paso de inducción: Supongamos que el lema se cumple para todo $j < k$.

Dado que si $\langle t, s \rangle \in T_{\mathcal{E}^{\sim}_k} \Uparrow \omega(M_{k-1})$ entonces $\langle t, s \rangle \in T_{\mathcal{E}^{\sim}_k} \Uparrow n(M_{k-1})$ para algún $n < \omega$, probamos el lema para k procediendo por inducción sobre n.

Si $n = 0$ entonces $\langle t, s \rangle \in M_{k-1}$ por la Definición 4.4.3. Como $M_{k-1} = T_{\mathcal{E}^{\sim}_{k-1}} \Uparrow \omega(M_{k-2})$ por construcción, el lema se cumple aplicando la hipótesis de inducción sobre k y por ser $\mathcal{E}^{\sim}$ estratificada.

Si $n = 1$ entonces o $t = s$ es una instancia del axioma reflexivo o existe una cláusula $(l = r \Leftarrow \tilde{E}, \tilde{D}.)$ en $\mathcal{E}^{\sim}_k$, una sustitución σ y una ocurrencia $u \in O(t = s)$ tales que $(t = s)/u = l\sigma$, $\{\langle t, s \rangle [r\sigma]_u\} \cup \dot{E}\sigma \subseteq M_{k-1}$ y $\dot{D}\sigma \cap M_{k-1} = \emptyset$. Ahora bien,

a) si t es idéntico a s y $t = s$ no contiene redexes "innermost" entonces el lema se cumple porque la única $\mathcal{ICN}cn$-derivación para $(t = s \;\; \emptyset)$ conduce al objetivo $(\emptyset \;\; \emptyset)$.

b) si $t = s$ tiene redexes "innermost" entonces,

b.1) sea $u \in O(t = s)$ tal que $(t = s)/u$ es un redex "innermost" cuyo símbolo de función más externo es de nivel k. Entonces el $\mathcal{ICN}cn$-paso de derivación para $(t = s \;\; \emptyset)$ produce una frontera que contiene el nodo $(((t = s)[r\sigma]_u, \tilde{E}\sigma, \tilde{D}_{ext}\sigma) \;\; E)$ donde $\tilde{D}_{ext}$ es $\tilde{D}$ expresando cada disecuación con sintaxis extendida.

$b.2)$ sea $v \in O(t = s)$ tal que $(t = s)/v$ es un redex "innermost" cuyo símbolo de función más externo es de nivel $j < k$. Entonces se produce la transición

$$\langle [\,], [t = s \quad \emptyset] \rangle \rightarrow_{\mathcal{ICN}cn} \langle [\,], F \rangle$$

donde cada elemento de F es de la forma

$$((t = s)[r']_u, \tilde{E}', \tilde{D}'_{ext}) \quad E_s$$

y viene de reducir $(t = s)/v$ con una instancia de una cláusula $(l' = r' \Leftarrow \tilde{E}', \tilde{D}'.)$ en $\mathcal{E}_j^{\sim}$ y donde $\tilde{D}'_{ext}$ es la forma extendida de $\tilde{D}'$. Para todos los subobjetivos básicos no negados en $\tilde{E}'$ y para todos los subobjetivos básicos negados en $\tilde{D}'$ se cumple el lema por la hipótesis. Finalmente, para $(t = s)[r']_u$ o bien se cumple $a)$ o bien $b.1)$ o bien $b.2)$.

Por lo tanto, el lema se cumple para $n = 1$. Para $n > 1$, la única diferencia con $n = 1$ es que ahora tenemos que $\langle t, s \rangle \in T_{\mathcal{E}_k^{\sim}} \Uparrow n(M_{k-1})$. Nuevamente, el resultado se sigue usando la hipótesis de inducción.

Por lo tanto, por inducción sobre n hemos probado que el lema se cumple para k y, por inducción sobre k, el lema queda probado.

Nótese que en el lema anterior el caso $k = 1$ significa que el subobjetivo básico no negado tiene una derivación de éxito y para los casos $k > 1$, las derivaciones que cumplen las condiciones del lema también son de éxito. El siguiente teorema establece el resultado más importante de esta sección. Se trata de la completitud del $\mathcal{ICN}cn$-cálculo para objetivos básicos. Debemos hacer notar que cuando el objetivo inicial es básico entonces si el objetivo es resoluble, el único sistema respuesta computado por el cálculo es (o puede simplificarse a) $\emptyset$.

Teorema 5.2.8 Completitud Básica
Sea $\mathcal{E}^{\sim}$ una teoría ecuacional estratificada, completamente definida y con disciplina de constructores, y sea $t = s$ un subobjetivo básico no negado.

a) Si $comp(\mathcal{E}^{\sim}) \models t = s$, entonces

$a.1$) el $\mathcal{ICN}cn$-árbol de derivaciones para $(t = s \;\; \emptyset)$ es de éxito total.

$a.2$) el $\mathcal{ICN}cn$-árbol de derivaciones para $\sim (t = s \;\; \emptyset) \;\; \emptyset$ es fallado finitamente.

b) Si $comp(\mathcal{E}^{\sim}) \models t \neq s$, entonces

$b.1$) el $\mathcal{ICN}cn$-árbol de derivaciones para $(t = s \;\; \emptyset)$ es fallado finitamente.

$b.2$) el $\mathcal{ICN}cn$-árbol de derivaciones para $\sim (t = s \;\; \emptyset) \;\; \emptyset$ es de éxito total.

Demostración.

* Probamos $a.1$) por inducción sobre el nivel de los símbolos de función en $\mathcal{E}^{\sim}$ (en la prueba haremos uso del apartado b) aunque lo demostraremos después).

  Caso base: Supongamos que el nivel máximo de cualquier símbolo de función en $\mathcal{E}^{\sim}$ es 1. Entonces $\mathcal{E}^{\sim}$ es una teoría ecuacional de Horn, completamente definida y con disciplina de constructores. Por la completitud del "innermost narrowing" para estas teorías ecuacionales [53], todas las derivaciones con éxito acabarán en nodos cuyos sistemas respuesta son $\emptyset$, por lo que, claramente, el $\mathcal{ICN}cn$-árbol de derivaciones para $(t = s \;\; \emptyset)$ es de éxito total.

  Paso de inducción: Supongamos que el teorema se cumple cuando el nivel máximo de los símbolos de función en $\mathcal{E}^{\sim}$ es $(n-1)$ y probamos que también se cumple cuando el nivel es n. Supongamos que $t = s$ contiene un símbolo de función de nivel n. Como $comp(\mathcal{E}^{\sim}) \models t = s$ y $M_{\mathcal{E}^{\sim}}$ es un E-modelo de $comp(\mathcal{E}^{\sim})$ tenemos que $\langle t, s\rangle \in M_{\mathcal{E}^{\sim}}$. Por la construcción de $M_{\mathcal{E}^{\sim}}$, $\langle t, s\rangle \in M_{\mathcal{E}_n^{\sim}} = T_{\mathcal{E}_n^{\sim}} \Uparrow \omega(M_{n-1})$. Por el Lema 5.2.7, existe una derivación que conduce a un nodo $(g \;\; E)$ tal que, por la hipótesis de inducción y por b), cada subobjetivo básico negado y no negado tiene un $\mathcal{ICN}cn$-árbol de derivaciones de éxito total. Luego $(t = s \;\; \emptyset)$ también tiene un $\mathcal{ICN}cn$-árbol de derivaciones de éxito total.

* Probamos $b.1$) Supongamos que $comp(\mathcal{E}^{\sim}) \models t \neq s$ y supongamos que $\mathcal{ICN}cn$-árbol de derivaciones para $(t = s \;\; \emptyset)$ no es fallado

finitamente. Entonces, existirá al menos una derivación de éxito que terminará en un nodo $(\emptyset \quad E)$ tal que $C_{ET} \models E\theta$. Ahora bien, por la corrección del $\mathcal{ICN}cn$-cálculo, $comp(\mathcal{E}^{\sim}) \models \forall(t = s)\theta$ y, como la ecuación $(t = s)$ es básica, $comp(\mathcal{E}^{\sim}) \models t = s$, lo que contradice la hipótesis.

* a.2) se sigue directamente de a.1) y del lema 5.2.4.

* b.2) se sigue directamente de b.1) y del lema 5.2.5.

Capítulo 6

Conclusiones y trabajos futuros

En esta tesis se propone la idea de resolver constructivamente la negación del predicado igualdad con respecto a teorías ecuacionales que incluyen, a su vez, negación en las condiciones de las cláusulas.

Las principales aportaciones del trabajo son las definiciones de las semánticas operacional y declarativa y los resultados de corrección y completitud que las relacionan y que establecen la equivalencia entre ambas.

En nuestra aproximación, se define la semántica operacional mediante un procedimiento basado en "narrowing" y convenientemente extendido para tratar sistemas de ecuaciones y disecuaciones. Una de las características más significativa de este procedimiento es que las ecuaciones y disecuaciones se manejan de forma similar siguiendo la idea de que una forma simplificada de una fórmula negada puede obtenerse a partir de su versión positiva. De esta forma, construimos una porción del árbol de derivaciones para un subobjetivo negado (concretamente el primer nivel del árbol) a partir del primer nivel del árbol de derivaciones generado por "narrowing" para la versión positiva del subobjetivo. También hemos mostrado cómo la respuesta a una conjunción de ecuaciones y disecuaciones en una teoría ecuacional normal, completamente definida y con disciplina de constructores es un sistema complejo que incluye negación y cuantificación y cuya satisfacibilidad debe comprobarse para determinar su corrección. Así, si usamos un algoritmo de "narrowing" con estrategia "innermost" entonces para ese tipo de teorías normales los símbolos de igualdad y desigualdad en los sistemas respuesta pueden interpretarse sintácticamente. Asimismo, proponemos algunas sencillas reglas de optimización del mecanismo operacional para simplificar los objetivos.

La principal contribución del trabajo en la semántica declarativa es la definición de los conceptos de compleción y estratificación de una teoría ecuacional normal. Hemos mostrado que existe un modelo minimal de la compleción de una teoría ecuacional estratificada y presentamos una caracterización por punto fijo de tal modelo, adaptando al caso ecuacional la teoría de operadores no monótonos desarrollada para la programación lógica.

Desde el punto de vista de las aplicaciones, una de las más claras e inmediatas consiste en incorporar nuestro sistema como núcleo de un sistema más amplio que integre la programación lógica y ecuacional dentro del esquema CLP. En este caso, las restricciones del nuevo lenguaje serán sistemas de ecuaciones y disecuaciones cuya satisfacibilidad deberá determinarse con respecto a una teoría ecuacional normal que formará parte del propio programa. Para esta aplicación sería interesante investigar la posibilidad de usar distintas técnicas incrementales para construir el árbol de derivaciones ya que, dentro del esquema CLP, es suficiente con encontrar una respuesta del objetivo (nótese que de esta forma queda ya probada su resolubilidad).

Las líneas de trabajo futuras son varias. Una de las más interesantes consiste en estudiar la aplicabilidad de técnicas de composicionalidad en la generación del árbol de derivaciones. Como hemos visto en el Capítulo 3, el árbol de derivaciones generado al aplicar la regla de negación constructiva ecuacional cuando se resuelve un subobjetivo negado puede contener el mismo subobjetivo repetido en distintos nodos (la Figura 3.4 ilustra este hecho). Claramente, podríamos optimizar el método si cada uno de estos subobjetivos fuera resuelto una sóla vez en lugar de resolverlo en cada uno de los nodos en los que aparece. Es por esto que la composicionalidad "and" para objetivos nos parece una forma atractiva de resolver este problema. Estas técnicas ya se han aplicado en el caso puramente positivo (sin negación) en [4] demostrando que el conjunto de éxitos del "narrowing" básico es composicional con respecto a la conjunción de literales en los objetivos y en el cuerpo de las cláusulas. Nuestra idea es investigar las propiedades de composicionalidad del "innermost narrowing" y extender este resultado para el caso de incluir disecuaciones.

Otros posibles trabajos que vendrían a completar lo presentado en esta tesis podrían ser demostrar la independencia de la estratificación elegida del modelo construido para una teoría ecuacional estratificada, y demostrar la completitud no básica del mecanismo operacional que hemos definido. Para el primero de ellos, una posibilidad es construir el modelo perfecto de una teoría ecuacional estratificada y comprobar que éste coincide con el modelo construido siguiendo la aproximación por punto fijo descrita en el Capítulo 4. Por otra parte, la completitud no básica podría abordarse, por ejemplo, con respecto a una compleción a tres valores, al estilo de como se ha definido en [19] para la negación constructiva en CLP.

Bibliografía

[1] M. Adi and C. Kirchner. AC-Unification Race: The System Solving Approach, Implementation and Benchmarks. *Journal of Symbolic Computation*, 14:51–70, 1992.

[2] H. Aït-Kaci and R. Nasr. A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215, 1986.

[3] M. Alpuente, M. Falaschi, and G. Levi. Incremental Constraint Satisfaction for Equational Logic Programming. Technical Report TR-20/91, Dipartimento di Informatica, Università di Pisa, 1991. To appear in *Theoretical Computer Science*.

[4] M. Alpuente, M. Falaschi, and G. Vidal. Semantics-Based Compositional Analysis for Equational Horn Programs. Technical Report DSIC-II/5/93, UPV, 1993.

[5] M. Alpuente and M.J. Ramírez. Rapid Prototyping of Database applications in the Logic + Equational EUROPA environment. In *Proc. 7th Int'l Conf. on Expert Systems, Theory and Applications*, pages 62 – 66. Los Angeles, USA, 1990.

[6] K. R. Apt. Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 493–574. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

[7] K. R. Apt, H. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive*

Databases and Logic Programming, pages 89–148. Morgan Kaufmann, Los Altos, Ca., 1988.

[8] K. R. Apt and M.H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3):841–862, 1982.

[9] L. Bachmair, N. Dershowitz, and J. Hsiang. Orderings for equational proofs. In *Proc. First Symp. on Logic In Computer Science LICS'86*, pages 346 – 357. IEEE Computer Society Press, Cambridge, Mass., 1986.

[10] L. Bachmair and H. Ganzinger. On restrictions of ordered paramodulation with simplification. In E. Y. Shapiro, editor, *Proc. 10th Int'l Conf. on Automated Deduction CADE'90*, volume 449 of *Lecture Notes in Computer Science*, pages 427–441. Springer-Verlag, Berlin, 1990.

[11] L. Bachmair and H. Ganzinger. Perfect Model Semantics for Logic Programs with Equality. In K. Furukawa, editor, *Proc. Eighth Int'l Conf. on Logic Programming ICLP'91*, pages 645–659. The MIT Press, Cambridge, Mass., 1991.

[12] R. Barbutti, M. Bellia, G. Levi, and M. Martelli. On the integration of logic programming and functional programming. In *Proc. First IEEE Int'l Symp. on Logic Programming SLP'84*, pages 160 – 166. IEEE Computer Society Press, N.W., Washington, 1984.

[13] R. Barbutti, M. Bellia, G. Levi, and M. Martelli. LEAF: A language which integrates logic, equations and functions. In D. DeGroot and G. Lindstrom, editors, *Logic Programming: Functions, Relations and Equations*, pages 201–238. Prentice-Hall, 1986.

[14] M. Bellia and G. Levi. The relation between logic and functional languages: a survey. *Journal of Logic Programming*, 3:217–236, 1986.

[15] J.A. Bergstra and J.W. Klop. Conditional Rewrite Rules: confluence and termination. *Journal of Computer and System Sciences*, 32:323–362, 1986.

[16] H. Beringer and F. Porcher. A Relevant Scheme for Prolog Extensions: CLP(Conceptual Theory). In G. Levi and M. Martelli, editors, *Proc. Sixth Int'l Conf. on Logic Programming ICLP'89*, pages 131–148. The MIT Press, Cambridge, Mass., 1989.

[17] P. Bosco, E. Giovannetti, and C. Moiso. Narrowing vs. SLD-resolution. *Theoretical Computer Science*, 59:3–23, 1988.

[18] P. G. Bosco, E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. A complete semantic characterization of K-LEAF, a logic language with partial functions. In *Proc. of the Fourth IEEE Symp. on Logic Programming SLP'87*, pages 318 – 327, San Francisco, 1987. IEEE Computer Society Press, N.W., Washington.

[19] P. Bruscoli, F. Levi, G. Levi, and M.C. Meo. Intensional Negation in Constraint Logic Programs. Technical Report TR-11/93, Dipartimento di Informatica, Università di Pisa, 1993.

[20] H.J. Burckert. Solving Disequations in Equational Theories. In E. Lusk and R. Overbeek, editors, *9th Int'l Conf. on Automated Deduction CADE'88*, volume 310 of *Lecture Notes in Computer Science*, pages 517–526. Springer-Verlag, Berlin, 1988.

[21] D. Chan. Constructive Negation Based on the Completed Database. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming ICLP'88*, pages 111–125. The MIT Press, Cambridge, Mass., 1988.

[22] D. Chan. An Extension of Constructive Negation and its Application in Coroutining. In E. Lusk and R. Overbeck, editors, *Proc. North American Conf. on Logic Programming NACLP'89*, pages 477–493. The MIT Press, Cambridge, Mass., 1989.

[23] K.L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*. Plenum Press, New York, 1978.

[24] A. Colmerauer. Equations and Inequations on Finite and Infinite Trees. In *Proc. Int'l Conf. on Fifth Generation Computer Systems FGCS'84*, pages 85–99. ICOT, Tokio, 1984.

[25] A. Colmerauer. Opening the Prolog-III universe. BYTE magazine, vol. 12, n. 9, 1987.

[26] A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7):69–90, 1990.

[27] H. Comon. Sufficient Completeness, Term Rewriting Systems and Antiunification. In *Proc. of the 8th Int'l Conf. on Automated Deduction CADE'86*, volume 230 of *Lecture Notes in Computer Science*, pages 128–140. Springer-Verlag, Berlin, 1986.

[28] H. Comon and P. Lescanne. Equational Problems and Disunification. *Journal of Symbolic Computation*, 7:371–425, 1989.

[29] J. Darlington, Y.K. Guo, and H. Pull. Constraints unify functional and logic programming. Technical report, Department of Computing, Imperial College, London, 1991.

[30] N. Dershowitz. Orderings for Term-Rewriting Systems. *Theoretical Computer Science*, 17:279–301, 1982.

[31] N. Dershowitz. Computing with rewrite systems. *Information and Control*, 65:122–157, 1985.

[32] N. Dershowitz. Synthesis by completion. In *Proc. of the Ninth Int'l Joint Conf. on Artificial Intelligence IJCAI'85*, pages 208–214, Los Angeles, Ca., 1985.

[33] N. Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[34] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 243–320. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

[35] N. Dershowitz, J.-P. Jouannaud, and J.W. Klop. Open Problems in Rewriting. In *Proc. of the Fourth Int'l Conf. on Rewriting Techniques and Applications RTA'91*, volume 488 of *Lecture Notes in Computer Science*, pages 445–456. Springer-Verlag, Berlin, 1991.

[36] N. Dershowitz, S. Kaplan, and D.A. Plaisted. Infinite normal forms. In *Proc. of the 16th EATCS Int'l Colloquium on Automata, Languages and Programming ICALP'89*, volume 372 of *Lecture Notes in Computer Science*, pages 249–262. Springer-Verlag, Berlin, 1989.

[37] N. Dershowitz, S. Kaplan, and D.A. Plaisted. Rewrite, rewrite, rewrite, rewrite, rewrite,.... *Theoretical Computer Science*, 83:71–96, 1991.

[38] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22:465–476, 1979.

[39] N. Dershowitz and M. Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990.

[40] N. Dershowitz, M. Okada, and G. Sivakumar. Confluence of Conditional Rewrite Systems. In *Proc. of the 1st Int'l Workshop on Conditional Term Rewriting Systems CTRS'87*, volume 308 of *Lecture Notes in Computer Science*, pages 31–44. Springer-Verlag, Berlin, 1987.

[41] N. Dershowitz, M. Okada, and G. Sivakumar. Canonical Conditional Rewrite Systems. In *Proc. of the 9th Int'l Conf. on Automated Deduction CADE'88*, volume 310 of *Lecture Notes in Computer Science*, pages 538–549. Springer-Verlag, Berlin, 1988.

[42] N. Dershowitz and A. Plaisted. Logic Programming cum Applicative Programming. In *Proc. First IEEE Int'l Symp. on Logic Programming SLP'84*, pages 54–66. IEEE, 1984.

[43] N. Dershowitz and A. Plaisted. Equational Programming. In J.E. Hayes, D. Michie, and J. Richards, editors, *Machine Intelligence*, volume 11, pages 21–56. Oxford University Press, 1987.

[44] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, and T. Graf. Applications of CHIP to Industrial and Engineering

problems. *Proc. of the First Int'l Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 693–702, 1988.

[45] R. Echahed. On completeness of narrowing strategies. In *Proc. of the 13th Colloquium on Trees in Algebra and Programing CAAP'88*, volume 299 of *Lecture Notes in Computer Science*, pages 89–101. Springer-Verlag, Berlin, 1988.

[46] F. Fages. Associative-Commutative Unification. In *Proc. of the 7th EATCS Int'l Conf. on Automated Deduction CADE'84*, volume 170 of *Lecture Notes in Computer Science*, pages 194–208. Springer-Verlag, Berlin, 1984. Also in Journal of Symbolic Computation, 3(3):257-275,1987.

[47] F. Fages and G. Huet. Unification and Matching in Equational Theories. In *Proc. of 8th Colloquium on Trees in Algebra and Programming CAAP'83*, volume 159 of *Lecture Notes in Computer Science*, pages 205–220. Springer-Verlag, Berlin, 1983.

[48] M. Fay. First Order Unification in an Equational Theory. In *4th Int'l Conf. on Automated Deduction CADE'79*, pages 161–167, 1979.

[49] M. Fernández. Narrowing Based Procedures for Equational Disunification. In *Applicable Algebras in Engineering Communications and Computing*, volume 3, pages 1–26. Springer-Verlag, Berlin, 1992.

[50] M. Fitting. A Kripke-Kleene semantics for logic programs. In *Journal of Logic Programming*, volume 2, pages 295–312, 1985.

[51] L. Fribourg. A narrowing procedure for theories with constructors. In *7th Int'l Conf. on Automated Deduction CADE'84*, pages 259–278, 1984.

[52] L. Fribourg. Handling Functions Definitions Through Innermost Superposition and Rewriting. In *Proc. First Int'l Conf. on Rewriting Techniques and Applications RTA'85*, volume 202

of *Lecture Notes in Computer Science*, pages 325–344. Springer-Verlag, Berlin, 1985.

[53] L. Fribourg. Slog: a logic programming language interpreter based on clausal superposition and rewriting. In *Proc. Second IEEE Int'l Symp. on Logic Programming SLP'85*, pages 172–185. IEEE, 1985.

[54] U. Furbach, S. Hölldobler, and J. Schreiber. Horn equality theories and paramodulation. *Journal of Automated Reasoning*, 5:309–337, 1989.

[55] J.H. Gallier and S. Raatz. SLD-resolution methods for Horn Clauses with Equality based on E-unification. In *Proc. Third IEEE Int'l Symp. on Logic Programming SLP'86*, pages 168–179. IEEE Computer Society Press, 1986.

[56] J.H. Gallier and S. Raatz. Extending SLD-resolution to equational Horn clauses using E-unification. *Journal of Logic Programming*, 6:3–43, 1989.

[57] E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel Leaf: A Logic plus Functional Language. *Journal of Computer and System Sciences*, 42, 1991.

[58] E. Giovannetti and C. Moiso. A completeness result for E-unification algorithms based on Conditional Narrowing. In M. Boscarol, L. Carlucci, and G. Levi, editors, *Foundations of Logic and Functional Programming*, volume 306 of *Lecture Notes in Computer Science*, pages 157–167. Springer-Verlag, Berlin, 1986.

[59] J. Goguen and J. Meseguer. Eqlog: Equality, Types and Generic Modules for Logic Programming. In D. de Groot and G. Lindstrom, editors, *Logic Programming, Functions, Relations and Equations*, pages 295–363. Prentice-Hall, 1986.

[60] J.A. Goguen, J.W. Thatcher, and E.G. Wagner. An initial algebra approach to the specification, correctness and implementation

of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology*, volume 4, pages 80–149. Prentice-Hall, 1978.

[61] J.C. Gonzalez, M.T. Hortalá, and M. Rodríguez-Artalejo. A Functional Logic Language with Higher Order Logic Variables. Technical Report DIA-90/6, Facultad de Matemáticas, Universidad Complutense de Madrid, 1990.

[62] M. Höhfeld and G. Smolka. Definite relations over constraint languages. Technical report, IBM Deutschland GmbH, Stuttgart, 1988.

[63] S. Hölldobler. Equational Logic Programming. In *Proc. Second IEEE Symp. on Logic In Computer Science LICS'87*, pages 335–346. IEEE Computer Society Press, 1987.

[64] S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1989.

[65] S. Hölldobler. Conditional equational theories and complete sets of transformations. *Theoretical Computer Science*, 75:85–110, 1990.

[66] J. Hsiang and M. Rusinowitch. On Word Problems in Equational Theories. In *Proc. 14th Int'l Colloquium on Automata, Languages and Programming ICALP'87*, pages 362–369, 1987.

[67] G. Huet. Deduction and Computation. In *Proc.18th IEEE Symp. on Foundations of Computer Science*, pages 30–45. IEEE Computer Society Press, 1977.

[68] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[69] G. Huet and J.M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2):239–266, 1982.

[70] G. Huet and D.C. Oppen. Equations and Rewrite Rules: a Survey. In *Formal Languages: perspectives and open problems*, pages 349–405. Academic Press, 1980.

[71] J.M. Hullot. Canonical Forms and Unification. In *5th Int'l Conf. on Automated Deduction CADE'80*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer-Verlag, Berlin, 1980.

[72] H. Hussmann. Unification in conditional-equational theories. Technical Report MIP-8502, Fakultät für Mathematik und Informatik, Universität Passau, 1986. Also in *Proc. European Conf. On Computer Algebra EUROCAL'85*, volume 204 of Lecture Notes in Computer Science, pages 543-553. Springer-Verlag, 1985.

[73] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. Technical report, Department of Computer Science, Monash University, 1986.

[74] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proc. Fourteenth Annual ACM Symp. on Principles of Programming Languages POPL'87*, pages 111–119. ACM, 1987.

[75] J. Jaffar, J.-L. Lassez, and J. W. Lloyd. Completeness of the Negation as Failure Rule. In *Proc. Seventh Int'l Joint Conf. on Artificial Intelligence IJCAI'83*, pages 500–506, Karlsruhe, 1983.

[76] J. Jaffar, J.-L. Lassez, and M. J. Maher. PROLOG-II an instance of the logic programming language scheme. In M. Wirsing, editor, *Formal Descriptions of Programming Concepts*. North-Holland, 1986.

[77] J. Jaffar, J.-L. Lassez, and M.J. Maher. A theory of complete logic programs with equality. *Journal of Logic Programming*, 3:211–223, 1984.

[78] J. Jaffar, J.-L. Lassez, and M.J. Maher. A logic programming language scheme. In D. de Groot and G. Lindstrom, editors, *Logic Programming, Functions, Relations and Equations*, pages 441–468. Prentice Hall, Englewood Cliffs, NJ, 1986.

[79] J. Jaffar, J.-L. Lassez, and M.J. Maher. Some issues and trends in the semantics of logic programming. In E. Y. Shapiro, editor, *Proc. Third Int'l Conf. on Logic Programming ICLP'86*, volume 225 of *Lecture Notes in Computer Science*, pages 223–241. Springer-Verlag, Berlin, 1986.

[80] J. Jaffar and S. Michaylov. Methodology and Implementation of a CLP System. In J.-L. Lassez, editor, *Proc. Fourth Int'l Conf. on Logic Programming ICLP'87*, pages 196–218. The MIT Press, Cambridge, Mass., 1987.

[81] J.P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. In *Proc. 11th ACM Conf. on Principles of Programming Languages POPL'84*, pages 83–92, 1984.

[82] J.P. Jouannaud and B. Waldmann. Reductive conditional Term Rewriting Systems. In *Proc. of the 3rd IFIP Working Conf. on Formal Description of Programming Concepts*, pages 223–244, 1986.

[83] S. Kaplan. Conditional Rewrite Rules. *Theoretical Computer Science*, 33:175–193, 1984.

[84] S. Kaplan. Fair conditional term rewriting systems: unification, termination and confluence. In H.-J. Kreowski, editor, *Recent Trends in Data Type Specification*, volume 116 of *Informatik-Fachberichte*, pages 136–155. Springer-Verlag, Berlin, 1985.

[85] S. Kaplan. Positive/negative conditional rewriting. In S.Kaplan and J.-P. Jouannaud, editors, *Proc. of the 1st Int'l Workshop on Conditional Term Rewriting Systems CTRS'87*, volume 308 of *Lecture Notes in Computer Science*, pages 129–143. Springer-Verlag, Berlin, 1987.

[86] S. Kaplan. Simplifying Conditional Term Rewriting Systems: Unification, Termination and Confluence. *Journal of Symbolic Computation*, 4:295–334, 1987.

[87] J.R. Kennaway, J.W. Klop, M.R. Sleep, and F.J. De Vries. Transfinite reductions in orthogonal term rewriting systems. In *Proc.*

of the Fourth Int'l Conf. on Rewriting Techniques and Applications RTA'91, volume 488 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1991.

[88] C. Kirchner. Computing Unification Algorithms. In *Proc. First Symp. on Logic In Computer Science LICS'86*, pages 206 – 216. IEEE Computer Society Press, Cambridge, Mass., 1986.

[89] C. Kirchner, H. Kirchner, and M. Rusinowitz. Deduction with Symbolic Constraints. Technical report, Centre de Recherche en Informatique de Nancy CRIN, France, 1991.

[90] C. Kirchner and P. Lescanne. Solving Disequations. In *Proc. Second IEEE Symp. on Logic In Computer Science LICS'87*, pages 347–352. IEEE Computer Society Press, 1987.

[91] S.C. Kleene. *Introduction to Metamathematics*. Van Nostrand, New York, 1952.

[92] J.W. Klop. Term Rewriting Systems: from Church-Rosser to Knuth-Bendix and beyond. In *Proc. of the 17th Int'l Colloquium on Automata, Languages and Programming ICALP'90*, volume 443 of *Lecture Notes in Computer Science*, pages 350–369. Springer-Verlag, Berlin, 1990.

[93] J.W. Klop. Term Rewriting Systems. Technical Report CS-R9073, Centre for Mathematics and Computer Science, Amsterdam, 1991. Also in "Handbook of Logic in Computer Science", vol. 1, pp.1-112, 1992.

[94] D.E. Knuth and P.B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[95] R. Kowalski. *Logic for Problem Solving*. North-Holland, Amsterdam, 1979.

[96] H. Kuchen, F.J. López-Fraguas, J.J. Moreno-Navarro, and M. Rodríguez-Artalejo. Implementing Disequality in a Lazy Functional Logic Language. Technical Report 92-20, Aachener Informatik-Berichte, 1992.

[97] K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4:289–308, 1987.

[98] K. Kunen. Signed Data Dependencies in Logic Programs. *Journal of Logic Programming*, 7(3):231–245, 1989.

[99] J.-L. Lassez and M. J. Maher. Closures and Fairness in the Semantics of Programming Logic. *Theoretical Computer Science*, 29:167–184, 1984.

[100] J.-L. Lassez, M. J. Maher, and K. Marriott. Unification Revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, Los Altos, Ca., 1988.

[101] J.-L. Lassez and K.G. Marriot. Explicit Representation of Terms Defined by Counter Examples. *Journal of Automated Reasoning*, 3(3), 1987.

[102] P. Lincoln and J. Christian. Adventures in Associative-Commutative Unification (A Summary). In *Proc. of the 9th Conf. on Automated Deduction CADE'88*, volume 310 of *Lecture Notes in Computer Science*, pages 358–367. Springer-Verlag, Berlin, 1988.

[103] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987. Second edition.

[104] F.J. López-Fraguas. A general scheme for constraint functional logic programming. In H. Kirchner and G. Levi, editors, *Proc. of the Third Int'l Conf. on Algebraic and Logic Programming ALP'92*, volume 632 of *Lecture Notes in Computer Science*, pages 213–227. Springer-Verlag, Berlin, 1992.

[105] M. J. Maher. Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees. In *Proc. Third IEEE Symp. on Logic In Computer Science LICS'88*, pages 348–357. Computer Science Press, New York, 1988.

[106] A. Martelli, C. Moiso, and G.F. Rossi. An algorithm for unification in equational theories. In *Proc. Third IEEE Int'l Symp. on Logic Programming SLP'86*, pages 180–186. IEEE, 1986.

[107] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. Technical Report SRI-CSL-91-05, SRI Menlo Park, Ca., 1991.

[108] A. Middeldorp. Modular Aspects of Properties of Term Rewriting Systems Related to Normal Forms. In *Proc. of the Third Int'l Conf. on Rewriting Techniques and Applications RTA'89*, volume 355 of *Lecture Notes in Computer Science*, pages 263–277. Springer-Verlag, Berlin, 1989.

[109] A. Middeldorp and E. Hamoen. Counterexamples to Completeness Results for Basic Narrowing. Technical report, Centre for Mathematics and Computer Science, Amsterdam, 1991.

[110] A. Middeldorp and E. Hamoen. Counterexamples to Completeness Results for Basic Narrowing (Extended Abstract). In H. Kirchner and G. Levi, editors, *Proc. of the Third Int'l Conf. on Algebraic and Logic Programming ALP'92*, volume 632 of *Lecture Notes in Computer Science*, pages 244–258. Springer-Verlag, Berlin, 1992.

[111] C.K. Mohan. Fitting Semantics for Conditional Term Rewriting. In *Proc. of the Fifteenth Int'l Joint Conf. on Artificial Intelligence IJCAI'91*, pages 857–862, 1985.

[112] C.K. Mohan and M.K. Srivas. Conditional Specifications with Inequational Assumptions. In *First Int'l Workshop on Conditional Term Rewriting Systems CTRS'87*, volume 308 of *Lecture Notes in Computer Science*, pages 161–178. Springer-Verlag, Berlin, 1987.

[113] C.K. Mohan and M.K. Srivas. Negation with Logical Variables in Conditional Rewriting. In *Proc. of the Third Int'l Conf. on Rewriting Techniques and Applications RTA'89*, volume 355 of *Lecture Notes in Computer Science*, pages 292–310. Springer-Verlag, Berlin, 1989.

[114] C.K. Mohan, M.K. Srivas, and D. Kapur. Reasoning in Systems of Equations and Inequations. In *7th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 287 of *Lecture Notes in Computer Science*, pages 305–325. Springer-Verlag, Berlin, 1987.

[115] C.K. Mohan, M.K. Srivas, and D. Kapur. Inference Rules and Proof Procedures for Inequations. In *Journal of Logic Programming*, pages 75–104. Elsevier Science Publishing Co., 1990.

[116] A. Mora, J. Piris, M.J. Ramírez, and M. Falaschi. A Prototype System for Equational Constructive Negation. In *Int. Logic Programming Symp. ILPS'93*. The MIT Press, Cambridge, Mass., 1993.

[117] A. Mora and M.J. Ramírez. Un algoritmo basado en eliminación de cuantificadores para comprobar la satisfacibilidad de sistemas ecuacionales. Technical Report DSIC-II/9/93, Departamento de Sistemas Informáticos y Computación, Valencia, 1993.

[118] J.J. Moreno and M. Rodriguez-Artalejo. BABEL: A Functional and Logic Programming Language based on a constructor discipline and narrowing. In I. Grabowski, P. Lescanne, and W. Wechler, editors, *Proc. of the First Int'l Conf. on Algebraic and Logic Programming ALP'88*, volume 343 of *Lecture Notes in Computer Science*, pages 223–232. Springer-Verlag, Berlin, 1988.

[119] J.J. Moreno and M. Rodriguez-Artalejo. Logic Programming with Functions and Predicates: The Language BABEL. *Journal of Logic Programming*, 1989.

[120] D.L. Musser. On Proving Inductive Properties of Abstract Data Types. In *Proc. 7th ACM Symp. of Principles of Programming Languages*, pages 154 – 162, 1980.

[121] W. Nutt, P. Réty, and G. Smolka. Basic narrowing revisited. *Journal of Symbolic Computation*, 7:295–317, 1989.

[122] M. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1977.

[123] P. Padawitz. *Computing in Horn Clause Theories*, volume 16 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1988.

[124] G.E. Peterson and M.E. Stickel. Complete sets of Reductions for Some Equational Theories. *Journal of the ACM*, 28(2):233–264, 1981.

[125] J. Piris, A. Mora, and M.J. Ramírez. Optimizaciones para el método de negación constructiva ecuacional. In *Proc. II Congreso Nacional de Programación Declarativa PRODE'93*, 1993.

[126] J. Piris and M.J. Ramírez. An Implementation of Equational Constructive Negation. In *Proc. Eighth Italian Conference on Logic Programming GULP'93*, pages 143–156, 1993.

[127] J. Piris and M.J. Ramírez. A prototype system for equational constructive negation. Technical Report DSIC-II/7/1993, Departamento de Sistemas Informáticos y Computación, UPV, Valencia, 1993.

[128] D.A. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.

[129] G.D. Plotkin. Building-in equational theories. *Machine Intelligence*, 7:73–90, 1972.

[130] T. Przymusinski. On the Declarative Semantics of Deductive Databases and Logic Programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, Los Altos, Ca., 1988.

[131] T. Przymusinski. On Constructive Negation in Logic Programming. Draft paper at Proc. North American Conf. on Logic Programming NACLP'89, 1989.

[132] T. Przymusinski. On the Declarative and Procedural Semantics of Logic Programs. *Journal of Automated Reasoning*, 5(2):201–228, 1989.

[133] M.J. Ramírez and M. Falaschi. Conditional Narrowing with Constructive Negation for Equational Logic Programming. In *Proc. I Congreso Nacional de Programación Declarativa PRODE'92*, pages 329–344, 1992.

[134] M.J. Ramírez and M. Falaschi. Conditional Narrowing with Constructive Negation. In *Third Int'l Workshop on Extensions of Logic Programing ELP'92*, volume 660 of *Lecture Notes in Artificial Intelligence*, pages 59–79. Springer-Verlag, Berlin, 1993.

[135] M.J. Ramírez, M. Falaschi, and G. Levi. Conditional Narrowing with Constructive Negation. In *Proc. Seventh Italian Conference on Logic Programming GULP'92*, pages 377–392, 1992.

[136] U.S. Reddy. Narrowing as the Operational Semantics of Functional Languages. In *Proc. Second IEEE Int'l Symp. on Logic Programming SLP'85*, pages 138–151. IEEE, 1985.

[137] U.S. Reddy. Functional Logic Languages, Part i. In *Proc. of a Workshop on Graph Reduction*, volume 279 of *Lecture Notes in Computer Science*, pages 401–425. Springer-Verlag, Berlin, 1987.

[138] U.S. Reddy. Rewriting Techniques for Program Synthesis. In *Proc. of the Third Int'l Conf. on Rewriting Techniques and Applications RTA'89*, volume 355 of *Lecture Notes in Computer Science*, pages 388–403. Springer-Verlag, Berlin, 1989.

[139] R. Reiter. On Closed World Data Bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, New York, 1978.

[140] M. Rusinowitch. Path of subterm ordering and decomposition orderings revisited. *Journal of Symbolic Computation*, 3:117–131, 1987.

[141] M. Sato and T. Sakurai. QUTE: A Functional Language based on Unification. In D. DeGroot and G. Lindstrom, editors, *Logic Programming, Funtions, Relations and Equations*, pages 131–155. Prentice Hall, Englewood Cliffs, New York, 1986.

[142] J. C. Shepherdson. Negation as failure: a comparison of Clark's completed database and Reiter's closed world assumption. *Journal of Logic Programming*, 1(1):51–79, 1984.

[143] J. C. Shepherdson. Negation as Failure II. *Journal of Logic Programming*, 2(3):185–202, 1985.

[144] J. C. Shepherdson. Negation in logic programming. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 19–87. Morgan Kaufmann, Los Altos, Ca., 1988.

[145] J.H. Siekmann and P. Szabó. Universal unification and a classification of equational theories. In *6th Int'l Conf. on Automated Deduction CADE'82*, volume 138 of *Lecture Notes in Computer Science*, pages 369–389. Springer-Verlag, Berlin, 1982.

[146] M. Stickel. A Unification Algorithm for Associative Commutative Functions. *Journal of the ACM*, 28(3):423–434, 1981.

[147] P.J. Stuckey. Constructive Negation for Constraint Logic Programming. In *Proc. Sixth IEEE Symp. on Logic In Computer Science LICS'91*. IEEE Computer Society Press, 1991.

[148] P.A. Subrahmanyam and Y.H. You. FUNLOG: A Computational Model Integrating Logic and Functional Pprogramming. In D. DeGroot and G. Lindstrom, editors, *Logic Programming, Funtions, Relations and Equations*, pages 157–198. Prentice Hall, Englewood Cliffs, New York, 1986.

[149] H. Tamaki. Semantics of a logic programming language with a reducibility predicate. In *Proc. First IEEE Int'l Symp. on Logic Programming SLP'84*, pages 259 – 264. IEEE Computer Society Press, N.W., Washington, 1984.

[150] Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.

[151] Y. Toyama, J.W. Klop, and H.P. Barendregt. Termination for the direct sum of left-linear term rewriting systems. In *Proc. of the 3rd Int'l Conf. on Rewriting Techniques and Applications RTA'89*, volume 355 of *Lecture Notes in Computer Science*, pages 477–491. Springer-Verlag, Berlin, 1989.

[152] D.A. Turner. Miranda: a non-strict functional language with polymorfhic types. In J.P. Jouannaud, editor, *Proc. ACM Conf. on Functional Programming Languages and Computer Architecture*, volume 201 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Berlin, 1985.

[153] M.H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[154] M.H. van Emden and T. Maibaum. Equations compared with clauses for specification of abstract data types. In J. Minker H. Gallier and J. Nicolas, editors, *Advances in Data Base Theory*, volume 1, pages 159–193. Plenum Press, New York, 1981.

[155] C. Walinsky. CLP(Σ^*): Constraint Logic Programming with Regular Sets. In G. Levi and M. Martelli, editors, *Proc. Sixth Int'l Conf. on Logic Programming ICLP'89*, pages 181–196. The MIT Press, Cambridge, Mass., 1989.

[156] M. Wallace. Negation by Constraints: A Sound and Efficient Implementation of Negation in Deductive Databases. In *Proc. of the Fourth IEEE Symp. on Logic Programming SLP'87*, pages 253 – 263, San Francisco, 1987. IEEE Computer Society Press, N.W., Washington.

[157] L. Wos and W. McCune. Negative Paramodulation. In *Proc. of the 8th Int'l Conf. on Automated Deduction CADE'86*, volume 230 of *Lecture Notes in Computer Science*, pages 229–239. Springer-Verlag, Berlin, 1986.

Anexo A

Un Sistema Prototipo para la Negación Constructiva Ecuacional

En este anexo presentamos un sistema experimental que resuelve ecuaciones y disecuaciones con respecto a una teoría ecuacional normal. Este sistema implementa el método de negación constructiva ecuacional que hemos definido en el Capítulo 3. El prototipo ha sido realizado en BIMProlog y funciona sobre estaciones de trabajo SUN 3/80.

Resolver ecuaciones y disecuaciones usando el método de negación constructiva ecuacional consiste en computar respuestas cuya satisfacibilidad (sintáctica) tiene que ser comprobada posteriormente para poder determinar su corrección. Ya que ambas tareas, computar respuestas y comprobar su satisfacibilidad, juegan papeles independientes dentro del propio método, lo más adecuado es que en el prototipo estén separadas una de la otra. Por ello, proponemos una estructura modular para el sistema que queda organizado en dos módulos principales (Figura A.1): el módulo de *Resolución de Objetivos* y el módulo *Analizador de Respuestas*. El primero de ellos hace la función de máquina de inferencia mientras que el Analizador de Respuestas es, como su nombre indica, el encargado de analizar la satisfacibilidad de las respuestas computadas aplicando el método de Maher [105] para el caso de una signatura finita. El sistema se completa con una biblioteca de cláusulas de uso general.

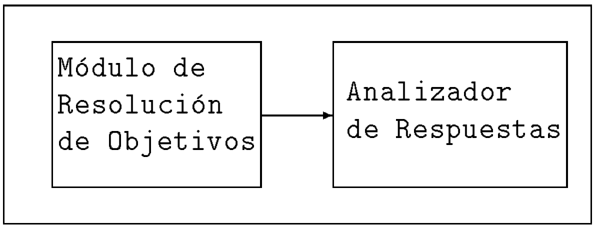

Figura A.1: *Estructura modular del sistema prototipo*

Asimismo, hemos estudiado la bondad de las optimizaciones propuestas en la Sección 3.4 y que han sido incorporadas en el prototipo. En este anexo, reflejaremos algunos de los resultados experimentales obtenidos con el sistema.

A continuación discutimos los detalles más destacados y significativos de la definición de los dos módulos mencionados. Al final de este anexo pueden encontrarse todos los aspectos referentes a la implementación de los mismos en el código completo que se adjunta. Debemos hacer notar en este punto que el sistema es un prototipo cuya principal finalidad es la de experimentar y evaluar la viabilidad de la negación constructiva para programación lógica-ecuacional, por lo que en esta versión no se han abordado las cuestiones relacionadas con el entorno (interfaz amigable con el usuario con facilidades adicionales tales como manejo de una biblioteca de programas, simplificación de las respuestas para mejorar su comprensibilidad, ...).

A.1 El Módulo de Resolución de Objetivos

El módulo de Resolución de Objetivos constituye el núcleo del sistema e implementa el mecanismo de deducción que evalúa los objetivos. Esto es, este módulo corresponde a la realización práctica del método de negación constructiva ecuacional de la Definición 3.2.8 y que hemos formalizado en la Sección 3.3 como el cálculo $\mathcal{ICN}cn$.

Para poder abordar la realización práctica del método debemos determinar primero qué estrategia de búsqueda y qué regla de computación se han de utilizar. En lo que se refiere a la estrategia de búsqueda es el propio método el que implícitamente la establece. Así, aunque una estrategia de búsqueda en profundidad es más eficiente que una búsqueda en amplitud e incluso es capaz de encontrar soluciones antes, no es adecuada para nuestros propósitos ya que para resolver subobjetivos negados, el módulo de Resolución de Objetivos tiene que expandir fronteras enteras que luego se negarán. Por consiguiente, los nodos del árbol de derivaciones para un objetivo dado se visitarán siguiendo una estrategia de búsqueda en amplitud. Una implementación sencilla de esta estrategia consiste en representar el árbol de derivaciones como una lista (cuyas componentes representan los distintos nodos) y luego tratar dicha lista como una cola. En lo que se refiere a la regla de computación, hemos adoptado una variante de la regla estándar de izquierda a derecha que recorre el objetivo en este sentido y selecciona el primer subobjetivo que encuentra no negado pero que contiene un redex de "narrowing" o bien uno negado.

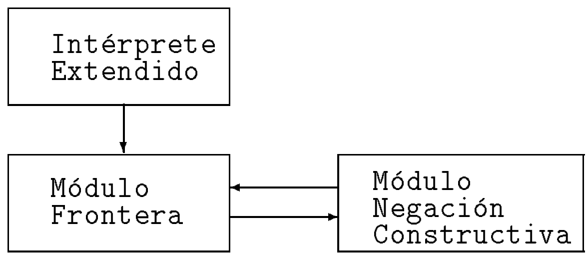

Figura A.2: *Estructura del módulo de Resolución de Objetivos*

El carácter jerárquico del cálculo $\mathcal{ICN}cn$ sugiere una implementación modular directa para el módulo de Resolución de Objetivos. Por este motivo, este módulo está a su vez organizado en tres módulos, cada uno de los cuales implementa uno de los niveles del cálculo $\mathcal{ICN}cn$: el módulo Intérprete Extendido (nivel $\rightarrow_{\mathcal{ICN}cn}$), el módulo Frontera (nivel $\rightarrow_f$) y el módulo de Negación Constructiva (nivel $\rightarrow_{\mathcal{ECN}}$). En la Figura A.2 se muestra esta estructura modular. El módulo Frontera y el módulo de Negación Constructiva interactúan haciendo uso cada uno de ellos de los predicados definidos en el otro.

Cada uno de los tres módulos anteriores define un procedimiento:

el módulo Intérprete Extendido define el predicado `--->ICNcn`/2, el módulo Frontera define el predicado `-f-->`/2 y, finalmente, el predicado `--->ECN`/2 es el núcleo del último módulo (Negación Constructiva). Con objeto de ofrecer un código fuente sencillo, el diseño de estos módulos reproduce las reglas del cálculo $\mathcal{ICN}cn$.

El módulo de Resolución de Objetivos se ocupa tanto de la interfaz con el ususario (acepta los programas y objetivos y devuelve las respuestas) como de computar las respuestas a los objetivos iniciales. Este módulo se define con el predicado `init`/0.

```
init:- read_program,read_signature,cn.
```

En la regla anterior, el predicado `read_program`/0 es el encargado de leer y almacenar en la base las reglas de que se compone el programa. Estas reglas siguen la sintaxis:

```
Left=Right<-Body.
```

y se almacenan como cláusulas Prolog de la forma:

```
rwr(Left,Right,Body)
```

donde `Left` y `Right` son, respectivamente, la parte izquierda y derecha de la cabeza de la ecuación condicional y `Body` es la condición. Tanto las reglas como el objetivo deben tener una sintaxis extendida. Para ello, las disecuaciones (y en general todos los subobjetivos negados) se representan como estructuras con símbolo de función ∼`exist`/2 y cuyo primer argumento es una lista que contiene las variables afectadas por el cuantificador existencial, mientras que el segundo argumento es de la forma (`G box E`) donde tanto `G` como `E` son conjunciones que denotan, respectivamente, la parte objetivo y la parte restricción del subobjetivo. Esta representación facilitará su posterior manejo por los distintos predicados que conforman el sistema.

Análogamente, el predicado `read_signature`/0 lee la signatura de referencia, almacenando en una lista pares con los símbolos de función del programa y su aridad y haciendo distinción entre los símbolos irreducibles y los reducibles. Finalmente, el predicado `cn`/0 lee el objetivo inicial y computa sus respuestas a través del predicado `--->ICNcn`/2.

```
cn:-write(' ==> ' ),nl,
    vread(Goal,LVbles),nl,take_vbles(LVbles,Vbles),
    (Vbles,([],[Goal])) '--->ICNcn' Solution,
    nl,print_(2,Solution),nl,nl,cn.
```

Por último, dado que el mecanismo operacional que se ocupa de computar las respuestas de un objetivo se basa en un procedimiento de resolución de ecuaciones, concretamente en un algoritmo de "narrowing", el módulo de Resolución de Objetivos se apoya en un algoritmo de "innermost narrowing" usado tanto por el módulo Frontera como por el módulo de Negación Constructiva. En consecuencia, el mecanismo computacional subyacente del sistema es una implementación del algoritmo de "innermost narrowing" a través del predicado `narred`/4 cuya definición presentaremos posteriormente.

A continuación, describimos brevemente cada uno de los módulos de que se compone el módulo de Resolución de Objetivos.

A.1.1 El Módulo Intérprete Extendido

El módulo Intérprete Extendido, que implementa el nivel $\rightarrow_{\mathcal{ICN}cn}$ del cálculo $\mathcal{ICN}cn$, se ocupa de controlar la expansión del árbol de derivaciones para un objetivo dado usando los procedimientos del módulo Frontera, así como de ir acumulando los distintos sistemas respuesta computados.

El predicado `--->ICNcn`/2, definido por las siguientes reglas, es el núcleo del Intérprete Extendido. Este predicado tiene dos argumentos: el primero de ellos es un par cuya primera componente es el conjunto de variables del objetivo inicial, y cuya segunda componente es un $\mathcal{ICN}cn$-estado, tal y como se ha especificado en la Definición 3.3.3, es decir, es un par de listas la primera de las cuales contiene los sistemas respuesta y la segunda los objetivos pendientes de evaluación. El segundo argumento del predicado `--->ICNcn`/2 es también un par que consta de las variables del objetivo inicial y del nuevo $\mathcal{ICN}cn$-estado.

```
(Vars_ini,(S,[])) '--->ICNcn' (Vars_ini,(S,[])):- !.  /* Stop rule */
(Vars_ini,(S,[(C box E)|L])) '--->ICNcn' (Vars_ini,(S',L')):-
        update(irr_neg(off)),\+ inner_term(C,_,_),
        syntac_unify(C,Mgu),t_l_to_c(Mgu,Cmgu),
```

```
          cappend(Cmgu,E,New_E),solvable(New_E),
          (Vars_ini,([New_E|S],L)) `--->ICNcn' (Vars_ini,(S',L')),!.
(Vars_ini,(S,[Goal|L])) `--->ICNcn' (Vars_ini,(S',L')):-
          (Vars_ini,Goal) `-f-->' (LL,_),
          simplify(LL,LL'),append(L,LL',LLL),
          (Vars_ini,(S,LLL)) `--->ICNcn' (Vars_ini,(S',L')),!.
(Vars_ini,(S,[Goal|L])) `--->ICNcn' (Vars_ini,(S',L')):-
          (Vars_ini,(S,L)) `--->ICNcn' (Vars_ini,(S',L')).
```

El sistema trabaja actualmente computando todas las soluciones del objetivo inicial (primera regla), es decir, la implementación considera como $\mathcal{ICN}cn$-estado terminal únicamente aquel estado $(S, [\,])$ en el que la lista de objetivos pendientes es vacía.

Cada una de las reglas restantes que definen `--->ICNcn`/2 corresponde a una regla del cálculo $\mathcal{ICN}cn$. Así, la primera de las mismas corresponde a la regla de éxito (regla 8 del cálculo) que sólo se aplica cuando el objetivo actual tiene como parte objetivo una conjunción de ecuaciones sin redexes que unifican sintácticamente y cuya parte restricción resultante es satisfacible. En esta regla, el predicado `inner_term`/3 comprueba si su primer argumento, la parte objetivo del objetivo actual, tiene redexes "innermost" y el predicado `syntac_unify`/2 trata de encontrar un unificador de esta parte objetivo (`inner_term`/3 se usa también en el módulo que implementa el algoritmo de "innermost narrowing"). Si todo esto ocurre entonces el unificador se añade al sistema en la parte restricción del objetivo actual si la nueva restricción formada es satisfacible (lo que comprueba el predicado `solvable`/1, núcleo del módulo que presentaremos en la Sección A.3) y el objetivo actual se elimina de la lista de objetivos pendientes. El predicado `syntac_unify`/2 calcula el mgu de una conjunción de ecuaciones, calculando uno a uno los mgus de cada ecuación (predicado `take_mgu`/4) y concatenándolos, eliminando aquellos que estan repetidos (predicado `append_new_mgu`/3).

```
syntac_unify(C,Mgu):- syntac_unify(C,[],Mgu),!.
syntac_unify((L=R,C2),Aux_mgu,Mgu):-
             syntac_unify(L=R,Aux_mgu,New_mgu),
             syntac_unify(C2,New_mgu,Mgu),!.
syntac_unify(L=R,Aux_mgu,New_mgu):-
```

```
update(aux_suw(L=R,Aux_mgu)),aux_suw(L_=R_,A_mgu),
apply_mgu(A_mgu),occur(L_,R_),
take_mgu(L=R,L_=R_,Mgu_theta,New_vars),
append_new_mgu(Aux_mgu,Mgu_theta,New_mgu).
```

La tercera regla en la definición de `--->ICNcn`/2 corresponde a la regla 9 del cálculo y realiza la expansión del árbol de derivaciones para el objetivo actual justo hasta su frontera, la cual se computa en el módulo Frontera (predicado `-f-->`/2). Una vez obtenida la frontera, el predicado `simplify`/2 es el encargado de simplificarla de acuerdo a las reglas de optimización de la Sección 3.4. `simplify`/2 se define mediante la regla:

```
simplify(L,L_):- simplify_(L,LL),(simpl(si),
                 simplify_s(LL,L_);LL=L_),!.
```

El predicado `simplify_`/2 elimina de la frontera los objetivos cuyas partes restricción son equivalentes al valor falso, bien por contener este valor o bien por ser insatisfacibles. La lista resultante es entonces nuevamente simplificada por el predicado `simplify_s`/2 que implementa las reglas de simplificación basadas en la distinción entre símbolos de función reducibles e irreducibles y que se aplican a la parte objetivo de cada nodo.

Finalmente, la última regla que define el predicado `--->ICNcn`/2 corresponde a la regla de fallo del método y, por lo tanto, lleva a cabo la eliminación en el árbol de derivaciones de los nodos fallados.

A.1.2 El Módulo Frontera

El módulo Frontera implementa el nivel $\rightarrow_f$ del cálculo $\mathcal{ICN}cn$. Este módulo coopera con el módulo de Negación Constructiva para llevar a cabo la expansión de la frontera del objetivo actual. La expansión del árbol se realiza de la manera siguiente: si la regla de computación selecciona una ecuación, entonces se ejecuta un paso del algoritmo de "innermost narrowing"; si selecciona un subobjetivo negado, entonces primero se genera la frontera del subobjetivo negado (usando el módulo de Negación Constructiva) y luego se construye la frontera del objetivo actual.

El módulo Frontera se define, básicamente, mediante el predicado `-f-->`/2. El primer argumento es un par cuyas componentes son, respectivamente, la lista de variables del objetivo inicial (que utilizará el algoritmo de "narrowing" para simplificar los mgus) y el objetivo actual, mientras que el segundo argumento de `-f-->`/2 es un par formado por dos listas, que representan la frontera del objetivo actual y las nuevas variables introducidas en la computación de la frontera.

```
(Ini_var,Goal) '-f-->' (Frontier,New_vars):-
               narred(Ini_var,Goal,Frontier,New_vars),!.
(Ini_var,Goal) '-f-->' (Frontier,New_vars):-
               dont_care(Goal,Neg_sub,Var_hole,S_hole),
               (Ini_var,Neg_sub) '--->ECN' Frontier_n,
               (Frontier_n=to_prune,Frontier=[],
                           New_vars=not_needed;
               generation_f(S_hole,Var_hole,Frontier_n,
                           Frontier,New_vars)),!.
```

La primera de las reglas anteriores implementa la regla 5 del cálculo y corresponde al caso en el que se selecciona un subobjetivo no negado del objetivo actual. En este caso, se debe reducir un redex "innermost" con todas las cláusulas con cuya parte izquierda de la cabeza unifica. Esto es lo que realiza el predicado `narred`/4 (Sección A.2).

La segunda de las reglas que definen el predicado `-f-->`/2 corresponde al caso en el que se selecciona un subobjetivo negado del objetivo actual (reglas 5 y 6 del cálculo), selección que efectúa el predicado `dont_care`/4. A continuación, el procedimiento `--->ECN`/2 construye la frontera del subobjetivo negado (módulo de Negación Constructiva). En el caso de que esta frontera tenga el valor '`to_prune`' esto significa que el objetivo actual tiene que eliminarse, con lo que el procedimiento `-f-->`/2 devuelve la lista vacía como respuesta. En caso contrario, `generation_f`/5 construye la frontera del subobjetivo actual. Cada nodo de esta frontera se obtiene tomando un elemento de la frontera computada por `--->ECN`/2, añadiendo su parte restricción a la restricción del objetivo actual (predicado `cappend`/3) y sustituyendo en el objetivo actual el subobjetivo negado por la parte objetivo del mismo (predicado `new_goal/4`).

```
generation_f(_,_,[],[],[]).
generation_f(S_hole,Var_hole,[A|R],[B|RC],[[]|Rn]):-
           update(aux_filling_holes(S_hole,Var_hole)),
           retract(aux_filling_holes(NS_hole,NVar_hole)),
           S_hole=NS_hole,A=(C box E1),
           NS_hole=(CC box E2),AA=C,cappend(E1,E2,E),
           NS_hole_=(CC box E),update(quit_trues(off)),
           new_goal(NS_hole_,Var_hole,AA,B),
           generation_f(S_hole,Var_hole,R,RC,Rn),!.
```

A.1.3 El Módulo de Negación Constructiva

El módulo de Negacion Constructiva implementa el nivel $\rightarrow_{ECN}$ del cálculo y tiene como objeto la evaluación de los subobjetivos negados. Como ya hemos mencionado, este módulo es utilizado por el módulo Frontera (cuando se selecciona un subobjetivo negado) y él, a su vez, hace uso de los procedimientos definidos en el módulo Frontera para generar la frontera del subobjetivo complementario del subobjetivo negado.

El predicado principal que define este módulo es `--->ECN/2`, cuyo primer argumento es un par formado por las variables del objetivo inicial y por el subobjetivo negado, mientras que su segundo argumento es una lista que representa la frontera de dicho subobjetivo. `--->ECN/2` se define mediante tres reglas. La primera de ellas implementa las dos primeras reglas de la relación $\rightarrow_{ECN}$, es decir, corresponde a las reglas de unificación y de fallo. Ambas deben aplicarse cuando el subobjetivo complementario del seleccionado no contiene redexes "innermost" (lo que comprueba `inner_term/3`) y sólo contiene ecuaciones que unifican sintácticamente (predicado `syntac_unify/2`). Ahora bien, si el predicado `solvable/1` tiene éxito, tomando como su argumento el sistema restricción construido con el mgu y la parte restricción del objetivo actual, entonces la frontera contiene un único nodo con parte objetivo vacía (lo que se expresa mediante la ecuación trivial `true=true`). En caso contrario, se devuelve como frontera el valor '`to_prune`', para indicar al módulo Frontera que el objetivo actual debe eliminarse.

```
(Ini_var,~(exist(Z,G box E))) '--->ECN' Return :-
          update(irr_neg(off)),\+inner_term(G,_,_),
          syntac_unify(G,Mgu),t_l_to_c(Mgu,CMgu),
          cappend(CMgu,E,E_theta_eqs),(solvable(~E_theta_eqs),
          take_vbles(E_theta_eqs,Vbles_),are_in(Z,Vbles_,_),
          (Vbles_=[],Fa=~E_theta_eqs;
          Fa=for_all(Vbles_,~E_theta_eqs)),
          Return=[true=true box Fa];Return=to_prune),!.
```

La siguiente regla corresponde a la regla de expansión de la frontera del subobjetivo negado (regla 3 del cálculo):

```
(Ini_var,~(exist(Z,C box E))) '--->ECN' L_one :-
          (Ini_var,C box E ) '-f-->' (List,New_var),
          append_m(Z,New_var,V_rule),
          e_c_n(List,V_rule,L_one),!.
```

En la regla anterior, una vez calculada la frontera del subobjetivo complementario, el predicado `e_c_n`/3 se ocupa de transformarla para construir a partir de ella la frontera del subobjetivo negado, aplicando la transformación de la Definición 3.2.7: `form_pairs`/4 descompone cada elemento de la frontera en dos y `combined`/3 combina entre sí estos nuevos elementos dos a dos.

```
e_c_n(List,New_vars,List_pairs_combined):-
     form_pairs(List,New_vars,List_pairsG,List_pairse),
     combined(List_pairsG,List_pairse,List_pairs_combined),!.
```

La tercera y última regla que define `--->ECN`/2 implementa la regla de satisfacción de subobjetivos negados (cuarta regla del cálculo).

```
(Ini_var,~(exist(Z,C box E)))'--->ECN'
                              [true=true box true=true]:- !.
```

A.2 El algoritmo de "Innermost Narrowing"

El algoritmo de "innermost narrowing" se implementa con el predicado `narred/3`. Este procedimiento realiza un paso de "narrowing" a una ocurrencia "innermost", reduciendo el redex seleccionado con todas las reglas con cuya parte izquierda de la cabeza unifica.

```
narred(Ini_var,C box E,New_goals,New_var):-
               update(irr_neg(off)),
               inner_term(C,(C_hole,Var_hole),Inner_term),
               look_rules(Inner_term,L_rules),
               one_step_reduce(Ini_var,(C_hole,Var_hole),E,
                       Inner_term,L_rules,New_goals,New_var),!.
```

El predicado `inner_term/3` es el que se ocupa de seleccionar un redex "innermost" de una de las ecuaciones de la conjunción que representa la parte objetivo del objetivo actual (primer argumento). Para luego poder reemplazar cómodamente el redex seleccionado por la parte derecha de las cabezas de las reglas, el segundo argumento de este predicado contiene una copia de la conjunción en la que el redex se ha sustituido por una variable nueva.

```
inner_term(T,(Term_hole,Var_hole),Term):-
             \+ var(T),functor(T,F,N_args),
             T=..[F|L_args],F \= (~),
             inner_term_args(L_args,(L_args_hole,Var_hole),Term),
             Term_hole=..[F|L_args_hole].
inner_term(T,(Var_hole,Var_hole),T):-
             \+ var(T),functor(T,F,N_args),
             irre(L_irr),
             \+ member((F,N_args),L_irr),!.
inner_term_args([Arg|R_arg],([Arg_hole|R_Arg_hole],Var_hole),Term):-
             (inner_term_args(R_arg,(R_Arg_hole,Var_hole),Term),
             Arg_hole=Arg;
             inner_term(Arg,(Arg_hole,Var_hole),Term),
                           R_args=R_Args_hole).
```

En las cláusulas anteriores que definen el predicado `inner_term/3`, el predicado `inner_term_args/3` se ocupa de buscar redexes de "narrowing" entre los subtérminos de un término dado. Realizando la búsqueda del redex de esta forma se garantiza que al final el término seleccionado es un término "innermost".

Siguiendo con la definición de `narred/3`, el predicado `look_rules/2` construye una lista con todas aquellas reglas de programa cuya parte izquierda de la cabeza unifica con el término "innermost" seleccionado.

```
look_rules(S,L):- update(aux_l(X,X)),
                  rclause(rwr(Left,_,_),_,D),
                  Left=S,aux_l(L,X),
                  X=[D|X_],update(aux_l(L,X_)),fail.
look_rules(_,L):- aux_l(L,[]).
```

Finalmente, el procedimiento `one_step_reduce/7` realiza todas las derivaciones por "narrowing" para el cuarto argumento sobre las reglas contenidas en el quinto argumento. En cada una de estas derivaciones (realizadas por `apply_rule/7`) se construye el nuevo objetivo (sexto argumento) a partir de las partes objetivo (segundo argumento) y restricción (tercer argumento) del objetivo actual.

```
one_step_reduce(Ini_var,_,_,_,[],[],[]).
one_step_reduce(Ini_var,Goal_part,Const_part,Term,[Rule|R_rules],
        [New_goal|R_New_goals],[New_var|R_New_var]):-
            apply_rule(Ini_var,Goal_part,Const_part,Term,
                       Rule,New_goal,New_var),
            one_step_reduce(Ini_var,Goal_part,Const_part,Term,
                       R_rules,R_New_goals,R_New_var).
```

A.3 El Módulo Analizador de Respuestas

El módulo Analizador de Respuestas es el encargado de estudiar la satisfacibilidad de las soluciones computadas, implementando el método definido por Maher [105]. Este método es aplicable a las álgebras de los árboles finitos e infinitos, tanto para el caso de signaturas finitas como infinitas, y se basa en un procedimiento de eliminación de los cuantificadores que aparecen en la fórmula lógica inicial, aunque una eliminación completa no siempre es posible. Para comprobar si un sistema respuesta es satisfacible, el módulo Analizador de Respuestas implementa únicamente este procedimiento para el caso de una signatura finita y no singular. Una de las características más destacadas del método implementado es que al trabajar con signaturas finitas tenemos también que usar el Axioma de Cierre de Dominio (DCA), ya que en una misma fórmula pueden aparecer todos los símbolos de función.

Informalmente, el método transforma en primer lugar cada ecuación de la fórmula inicial (que debe estar en forma prenexa normal disyuntiva) en una fórmula básica (una conjunción de ecuaciones en forma resuelta donde las variables iniciales aparecen todas en la parte izquierda de las ecuaciones, mientras que los parámetros, todos cuantificados existencialmente, son variables que no pertenecen al conjunto de variables iniciales). La forma resuelta de una conjunción de ecuaciones se obtiene aplicando las siguientes reglas:

a) $f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)$
reemplazar por las ecuaciones $t_1 = s_1, \ldots, t_n = s_n$.

b) $f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)$ con $f \not\equiv g$
parar con fallo.

c) $x = x$
eliminar la ecuación.

d) $t = x$ donde t no es una variable
reemplazar por la ecuación $x = t$.

e) $x = t$ donde $t \not\equiv x$ y x aparece en alguna otra ecuación
Si t *incluye a* x *parar con fallo ("occur-check")*
sino reemplazar x *por* t *en el resto de ecuaciones.*

El proceso termina cuando no se puede aplicar ninguna regla o cuando se devuelve fallo.

Una vez calculada la forma resuelta, se selecciona el cuantificador prenexo más interno. Si éste es existencial, entonces se distribuye sobre la disyunción. Si es necesario se aplica el DCA a cada conjunción hasta que la profundidad de cada ocurrencia de los parámetros de la fórmula básica positiva es más grande que la profundidad de cualquier parámetro en las fórmulas básicas negativas, o dicho de otra forma, hasta que la fórmula básica positiva no unifique con las negativas o bien hasta que cada ocurrencia variable en las fórmulas básicas negativas sea una ocurrencia no variable en la fórmula básica positiva. Finalmente, cada conjunción se transforma en una combinación booleana de fórmulas básicas equivalente, donde se ha eliminado el cuantificador existencial considerado. Si el cuantificador prenexo más interno es un cuantificador universal, esto es, si tenemos la disyunción de la forma $Q\forall y\ (D_1\ \ \vee \ldots \vee\ \ D_n)$, donde Q denota el resto de cuantificadores, entonces la fórmula anterior es reemplazada por $Q\ \ \sim \exists y \sim (D_1\ \ \vee \ldots \vee\ \ D_n)$. A continuación se distribuye la negación dentro de la disyunción y se calcula la forma normal disyuntiva (tratando las fórmulas básicas como átomos). Llegado este punto, el método procede como antes. Finalmente, si todas las variables iniciales están cuantificadas, cuando se elimina el último cuantificador la fórmula se evalúa a cierto o falso.

El Analizador de Respuestas se define con el predicado `solvable/1` cuyo único argumento es el sistema respuesta calculado por el módulo de Resolución de Objetivos.

```
solvable(X):- update(solv(X)),retract(solv(Y)),
              dnf(Y,C),signa(SS),
              (solver(C,SS,L),write('is solvable'),nl;
               write('not is solvable'),nl,fail),nl,write(L),!.
```

El procedimiento `solver/3` es el que realmente implementa el método de eliminación de cuantificadores y constituye, por consiguiente, el núcleo de este módulo. Como ya hemos comentado, antes de aplicar el método es necesario poner el sistema respuesta en forma prenexa normal disyuntiva, transformación que realiza el predicado `dnf/2`.

```
solver([Cuan,List_sc],Signat,List_fin):-
       do_fb(List_sc,List_fb1),
       out_false(List_fb1,List_fb,Bool),
       (Bool=true;(!,solv_fin([Cuan,List_fb],Signat,List_fin),!,
       write(List_fin),eval(List_fin))).
```

En la regla anterior, que define el predicado `solver`/3, el predicado `do_fb`/2 transforma cada ecuación en una conjunción de fórmulas básicas y el predicado `out_false`/3 elimina aquellas conjunciones equivalentes al valor falso. A continuación, el procedimiento `solv_fin`/3 es el que implementa el método para signaturas finitas y, finalmente, `eval`/1 analiza si la fórmula resultante se reduce a un valor booleano.

A.4 Implementación y Resultados Experimentales

En esta sección quisieramos resaltar los últimos detalles sobre la implementación y las pruebas realizadas.

Dado que las variables del objetivo inicial (y, en general, cualquiera de las variables que aparecen como resultado de los pasos de computación) se tratan como variables Prolog y que incluso las ligaduras de las variables se manejan por éste, para que la salida ofrecida por el prototipo sea legible ha sido necesario desarrollar algunos procedimientos específicos capaces de mantener y manipular estructuras de la forma (`name=_Number`) que asocian el nombre con que se conoce externamente una variable y su representación interna. Asimismo, ya que el lenguaje de implementación es Prolog, hemos hecho uso de algunos de sus mecanismos, en particular del de "backtracking". Por ejemplo, el fallo de todos los algoritmos se ha implementado como fallo Prolog.

Nuestras primeras experiencias han ido encaminadas a observar el comportamiento del método y a evaluar su aplicabilidad práctica. En la Tabla A.1 resumimos algunos experimentos realizados con el sistema tanto en su versión optimizada como no optimizada. La teoría ecuacional normal considerada es la del Ejemplo 2 del Capítulo 2. La tabla refleja los tiempos de computación de la primera solución del objetivo. En la primera columna se incluyen los tiempos para el método de negación constructiva ecuacional mientras que en la segunda columna aparecen reflejados los tiempos para el método incluyendo las optimizaciones.

Como puede observarse en la tabla, la mejora en eficiencia del método es, como ya anunciabamos en la Sección 3.4, sobre todo apreciable para aquellos objetivos que incluyen negación (para algunos de estos

Objetivo	Tiempo s.o.	Tiempo c.o.
`ins(a,elim(x,ins(b,ins(b,empty))))= ins(a,ins(b,empty))`	7.20	3.25
`~ elim(a,ins(a,ins(b,empty)))= ins(a,ins(b,empty))`	15.2	8.49
`elim(y,ins(a,empty))=empty`	0.13	0.1
`~ elim(a,ins(a,empty))= empty`	0.4	0.24
`elim(a,ins(a,empty))= elim(a,ins(a,empty))`	0.45	0.3
`elim(a,empty)=y`	0.2	0.28
`~ elim(a,empty)=y`	1.26	0.31
`~ elim(a,empty)=empty`	0.75	0.5

Tabla A.1: *Tiempos en segundos sobre SUN-3/80*

objetivos se llega incluso a reducir a la mitad el tiempo de ejecución).

En un principio, se podría pensar que el test de satisfacibilidad de la parte restricción de cada nodo incrementaría el tiempo de computación de las soluciones de un objetivo dado, al tener que aplicarse en cada nodo del árbol de derivaciones. Los experimentos realizados demuestran lo contrario. Por ejemplo, para el objetivo `elim(a,empty) = y`, cuyo árbol de derivaciones no contiene nodos con subobjetivos negados, los tiempos obtenidos son del mismo orden de magnitud con o sin la aplicación de las optimizaciones.

A continuación incluimos el código completo del sistema.

```
/******************************************************
        ***************************************
                          CNS
             CONSTRUCTIVE NEGATION SYSTEM
           FOR EQUATIONAL LOGIC PROGRAMMING

         Maria Jose Ramirez          May 1993
        ***************************************
 *****************************************************/

   ?- op(740,fx,~).      /* azucar negacion/*
   ?- op(730,fx,exist).  /* cuantificador existencial/*
   ?- op(710,xfx,box).   /* conector objetivo-restriccion/*
   ?- op(800,xfx,'--->ICNcn').  /* relacion → ICNcn/*
   ?- op(800,xfx,'-f-->').      /* relacion → f/*
   ?- op(800,xfx,'--->ECN').    /* relacion → ECN/*
   :- alldynamic.    /* Todos los predicados son dinamicos/*
   ?- please(w,on).  /* Mensajes activos /*
   ?- please(c,on).  /* compatibilidad DEC-10 Prolog syntax/*

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

              Carga de los modulos restantes:
   icncn.pro:     interprete extendido
   frontier.pro:  modulo frontera
   cnm.pro:       negacion constructiva
   narrer.pro:    innermost narrowing
   solver.pro:    analizador de respuestas
   library.pro:   biblioteca de clausulas generales
   -a alldynamic
   -c DEC-10 Prolog syntax compatibility
   -d debug code
   -w warning on

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

?-[-'-a -c -d -w icncn.pro'].
?-[-'-a -c -d -w frontier.pro'].
?-[-'-a -c -d -w cnm.pro'].
?-[-'-a -c -d -w narrer.pro'].
?-[-'-a -c -d -w solver.pro'].
?-[-'-a -c -d -w library.pro'].
```

```
/**************************************************************
 Definicion de los simbolos de funcion irreducibles estandar con su
                              aridad.
 **************************************************************/


irre_standard([(=,2),(exist,2),(for_all,2),(',',2),
              (nil,0),('.',2),(box,2),(true,0)]).
irr_neg(off).
irre(L):-(irr_neg(off),pos_irre(L1),L=[(~,1)|L1]; neg_irre(L)),!.

/**************************************************************
   init/0 lee los programas y signatura y lanza la ejecucion del
                              sistema
 **************************************************************/

init:- read_program,read_signature,cn.

read_program:-please(w,off),
             write('Introducir una regla:' ),nl,
             vread(Clause),nl,Clause=(Left=Right<-Body),
             assert(rwr(Left,Right,Body),read_program,!.
read_signature:- irre_standard(I_s),read(I_p),update(irre_program(I_p))
             append(I_s,I_p,A_i),update(pos_irre(A_i)),
             read(R),append(A_i,R,Neg),update(neg_irre(Neg)),
             append(I_p,R,SS),update(signa(SS)),!.

/**************************************************************
 cn/0 lee el objetivo inicial y lo resuelve.  A partir del objetivo,
   take_vbles/2 genera una lista con los nombres de las variables
   iniciales que sera usada a lo largo de toda la computacion.  A
 continuacion, --->ICNcn/2 lanza el objetivo y print_/2 imprime las
                            soluciones.
 **************************************************************/

cn:- write(' ==> ' ),nl,
     vread(Goal,LVbles),nl,take_vbles(LVbles,Vbles),
     (Vbles,([],[Goal])) '--->ICNcn' Solution,
     nl,print_(2,Solution),nl,nl,cn.

print_(T,L):- tab(T), pprint(L),nl.
pprint([]):- !.
pprint([H|T]):- write(H), pprint(T).
```

```
/*****************************************************
        **************************************
                      ICNCN.PRO
                (Interprete Extendido)
        **************************************
*****************************************************/


/***************************************************************
 --->ICNcn/2 implementa la relacion → ICNcn del calculo ICNcn.  El
 primer argumento es un par que contiene la lista de variables del
 objetivo inicial y un ICNcn-estado, y su segundo argumento contiene
      la lista de variables iniciales y el nuevo ICNcn-estado.
 ***************************************************************/


(Vars_ini,(S,[])) `--->ICNcn' (Vars_ini,(S,[])):-!.  /* Stop rule */
(Vars_ini,(S,[(C box E)|L])) `--->ICNcn' (Vars_ini,(S',L')):-
     update(irr_neg(off)),\+ inner_term(C,_,_),
     syntac_unify(C,Mgu),t_l_to_c(Mgu,Cmgu),
     cappend(Cmgu,E,New_E),solvable(New_E),
     (Vars_ini,([New_E|S],L)) `--->ICNcn' (Vars_ini,(S',L')),!.
(Vars_ini,(S,[Goal|L])) `--->ICNcn' (Vars_ini,(S',L')):-
     (Vars_ini,Goal) `-f-->' (LL,_),
     simplify(LL,LL'),append(L,LL',LLL),
     (Vars_ini,(S,LLL)) `--->ICNcn' (Vars_ini,(S',L')),!.
(Vars_ini,(S,[Goal|L])) `--->ICNcn' (Vars_ini,(S',L')):-
     (Vars_ini,(S,L)) `--->ICNcn' (Vars_ini,(S',L')).


    /*******************************************************
 syntac_unify/2 calcula el mgu (segundo argumento) de una conjuncion
                de ecuaciones (primer argumento).
    *******************************************************/


   syntac_unify(C,Mgu):- syntac_unify(C,[],Mgu),!.


    /*******************************************************
syntac_unify/3 realiza la unificacion sintactica de una conjuncion de
 ecuaciones calculando uno a uno los unificadores de cada ecuacion.
  Sus argumentos son:  la conjuncion de ecuaciones, el mgu parcial
    calculado hasta el momento y el unificador de la conjuncion.
    *******************************************************/
```

```
syntac_unify((L=R,C2),Aux_mgu,Mgu):-
            syntac_unify(L=R,Aux_mgu,New_mgu),
            syntac_unify(C2,New_mgu,Mgu),!.
syntac_unify(L=R,Aux_mgu,New_mgu):-
            update(aux_suw(L=R,Aux_mgu)),aux_suw(L_=R_,A_mgu),
            apply_mgu(A_mgu),occur(L_,R_),
            take_mgu(L=R,L_=R_,Mgu_theta,New_vars),
            append_new_mgu(Aux_mgu,Mgu_theta,New_mgu).

     /*******************************************************
apply_mgu/1 instancia las variables de las ecuaciones de la lista que
 es su argumento usando el propio algoritmo de unificacion de Prolog.
     *******************************************************/

apply_mgu([]).
apply_mgu([A|R]):- call(A),apply_mgu(R).

     /*******************************************************
             take_mgu/4 calcula el mgu de una ecuacion.
    Este mgu es directo cuando ambos terminos de la ecuacion son
  identicos (el mgu es la identidad) o bien variables (el mgu es la
                         propia ecuacion).
     *******************************************************/

take_mgu(E,NE,[],[]):- E=(L=R),L==R,!.
take_mgu(E,NE,[E],[]):-X=(L=R),var(L),var(R),!.
take_mgu(E,NE,R,New_var):- take_mgu(E,NE,[],R,[],New_var),!.

take_mgu(X,T,Or,R,Onv,New_vars):-
        var(X),take_vbles(T,New_vars_),
        not_are_in(New_vars_,Onv,New_vars),
        (\+(is_in(X=T,Or)),R=[X=T|Or];R=Or),!.
take_mgu(A,A_,Or,R,Onv,Nv):-
        functor(a,_,0),A=A_,(var(A_),R=[A_=A|Or];R=Or),!.
take_mgu(T,T_,Or,L,Onv,New_vars):-
        T=..[F|Args],T_=..[F|Args_],
        functor(T,F,N),functor(T_,F,N),
        take_mgu_args(Args,Args_,Or,L,Onv,New_vars),!.

take_mgu_args([],[],Or,Or,Onv,Onv).
take_mgu_args([Ar|R_ar],[Ar_|R_ar_],Or,L,Onv1,New_var):-
        take_mgu(Ar,Ar_,Or,M_ar,Onv1,Onv2),
        take_mgu_args(R_ar,R_ar_,M_ar,L,Onv2,New_var),!.
```

```
/*********************************************************
append_new_mgu/3 concatena dos listas de mgu's de forma que no
aparezcan dos elementos iguales en la lista resultante.
*******************************************************/


append_new_mgu([],Mgu_theta,Mgu_theta).

append_new_mgu([Mgu|R_mgu],Old_L_mgu,New_L_mgu):-
              (\+(is_in(Mgu,Old_L_mgu)),
              append_new_mgu(R_mgu,[Mgu|Old_L_mgu],New_L_mgu);
              append_new_mgu(R_mgu,Old_L_mgu,New_L_mgu)),!.

/***************************************************************
simplify/2 toma la lista que representa la frontera del objetivo
actual (primer argumento) y la simplifica (segundo argumento)
eliminando los nodos cuya parte restriccion es insatisfacible y
aplicando las optimizaciones para el calculo que hemos propuesto.
***************************************************************/


simpl(no).

simplify(L,L_):- simplify_(L,LL),(simpl(si),simplify_s(LL,L_);LL=L_),!.

/***************************************************************
simplify_/2 elimina aquellos elementos de la lista primer argumento
cuya restriccion es falsa (por contener el valor falso o por ser
insatisfacible).
***************************************************************/


simplify_([],[]).
simplify_([G box S|R],RR):- (is_false(S);simpl(si),\+ solvable(S)),
                            simplify_(R,RR),!.
simplify_([G|R],[G|RR]):- simplify_(R,RR),!.

is_false(~(A)):- is_really_false(A,i),!.
is_false((~(A),B)):- is_really_false(A,i),!.
is_false((_,B)):- is_false(B),!.

is_really_false(C,i):- C==(true=true),!.
is_really_false(~(A),P):- (P=i,PP=p;P=p,PP=i),is_really_false(A,PP),!.
```

```
/*************************************************************
 simplify_s/2 aplica las reglas de simplificacion a cada uno de los
subobjetivos que aparecen en la lista pasada como primer argumento.
 ************************************************************/


simplify_s([],[]).
simplify_s([Subgoal|R_subgoals],List_subgoals):-
            simplify_sg(Subgoal,Simp_Subgoal),
            simplify_s(R_subgoals,R_list_subgoals),
            (Simp_Subgoal==to_prune,List_subgoals=R_list_subgoals;
             Simp_Subgoal==pos_sust;
             List_subgoals=[Simp_Subgoal|R_list_subgoals]),!.


 /*************************************************************
 simplify_sg/2 simplifica una a una las ecuaciones con simplify_pos/3
   y los subobjetivos negados con simplify_sgneg/2.  Las reglas de
     simplificacion de los subobjetivos negados son aplicadas por
                           simplify_neg/3.
  ************************************************************/


simplify_sg(G,Simp_Subgoal):-
              (has_pos_formula(G,Formula,G_hole),
              simplify_pos(G_hole,Formula,New_G),New_G=(Goal,Hole),
              (Hole==to_prune,Simp_Subgoal=Hole;
               simplify_sg(Goal,Simp_Subgoal_);G=Simp_Subgoal_),
              simplify_sgneg(Simp_Subgoal_,Simp_Subgoal),!.

simplify_sgneg(to_prune,to_prune):-!.
simplify_sgneg(G,Simp_Subgoal):-
              (has_neg_formula(G,Formula,G_hole),
              simplify_neg(G_hole,Formula,New_G),New_G=(Goal,Hole),
              (Hole==to_prune,Simp_Subgoal=Hole;
               simplify_sgneg(Goal,Simp_Subgoal));G=Simp_Subgoal),!.


 /*************************************************************
     has_pos_formula/3 selecciona una ecuacion de una conjuncion.
has_neg_formula/3 selecciona un subobjetivo negado de una conjuncion.
    En ambos casos se hace una copia de la conjuncion en la que el
subobjetivo se ha sustituido por una variable nueva.  Sus argumentos
   son:  la conjuncion, la ecuacion (o el subobjetivo negado) y la
         copia de la conjuncion junto con la nueva variable.
  ************************************************************/
```

```
has_pos_formula(Goal,Formula,(G_ box E_,V)):-
       Goal=(G box E),has_pos_formula_(G,Formula,(G_,V)),
       mark(C),(V==(true=true),E_=(_,E);E_=E),cut(C).

has_pos_formula_(T=S,T=S,R):-
       ((var(T);var(S)),R=(true=true,true=true);R=(V,V)),!.
has_pos_formula_((Eq,R_eq),Formula,(New_G,V)):-
       has_pos_formula_(Eq,Formula,(G,V)),cappend_g(G,R_eq,New_G);
       has_pos_formula_(R_eq,Formula,(G,V)),New_G=(Eq,G).

has_neg_formula(Goal,Formula,(G_hole box E,Hole)):-
       Goal=(G box E),
       has_neg_formula_(G,Formula_,(G_hole,Hole_)),
       Formula_=~(exist(L,G_ )),
       has_pos_formula(G_,Formula,(G_hole_,Hole)),
       Hole_=~(exist(L,G_hole_)).

has_neg_formula_(~(T),~(T),(V,V)).
has_neg_formula_((Eq,R_eq),Formula,(New_G,V)):-
       has_neg_formula_(Eq,Formula,(G,V)),New_G=(G,R_eq);
       has_neg_formula_(R_eq,Formula,(G,V)),New_G=(Eq,G).


  /*****************************************************************
             REGLAS DE SIMPLIFICACION PARA ECUACIONES.
  ****************************************************************/


            /******************************************
                         Regla de rechazo.
             ****************************************/


simplify_pos((_,_),T=R,(_,to_prune)):-
           nonvar(T),nonvar(R),mark(C),functor(T,F,A1),
           functor(R,G,A2),(F\=G;Af\=Ag),cut(C),
           are_irreducible((F,Af),(G,Ag)),!.


            /******************************************
                     Regla de descomposicion.
             ****************************************/


simplify_pos((Subg_hole,Var_hole),Formula,(Subg_hole_,Var_hole)):-
           pos_decom_condition(Formula,Subformulae),
           Subg_hole=(G box E),
```

```
            (nonvar(G),(\+ Subformulae==(true=true),
                        Var_hole=Subformulae,Subg_hole_=Subg_hole;
                        drop_conj(Var_hole,G,G_),Subg_hole_=(G_ box E));
             Var_hole=Subformulae,Subg_hole_=Subg_hole),!.


 /*****************************************************************
  pos_decom_condition/2 aplica la regla de descomposicion al primer
        argumento, si se cumplen las condiciones para ello.
  *****************************************************************/


pos_decom_condition(T=R,Subformulae):-
         nonvar(T),nonvar(R),T=..[F|Af],R=..[F|Ag],
         functor(T,_,A),functor(R,_,A),
         are_irreducible((F,A),(F,A)),
         (A=0,Subformulae=(true=true);
          form_subformulae(Af,Ag,Subformulae)),!.

form_subformulae([Af],[Ag],Af=Ag).
form_subformulae([Af|R_Af],[Ag|R_Ag],(Af=Ag,Rc)):-
         form_subformulae(R_Af,R_Ag,Rc),!.


             /******************************************
                        Regla de sustitucion.
              ******************************************/


simplify_pos(Subgoal,T=S,Subgoal_):-
            (var(T),Var=T,Term=S;var(S),Var=S,Term=T),
            functor(Term,F,A),are_irreducible((Term,A),(Term,A)),
            pos_sustitution_rule(Subgoal,Var,Term,Subgoal_),!.


 /*****************************************************************
 pos_sustitution_rule/4 aplica la regla de sustitucion eliminando del
 primer argumento una ecuacion de la forma X=t (X y t son el segundo
 y tercer argumentos) y devuelve como cuarto argumento el subobjetivo
   que resulta de reemplazar todas las ocurrencias de X por t en el
 primer argumento y de incorporar la ecuacion a la parte restriccion
                              del mismo.
  *****************************************************************/


pos_sustitution_rule((Subgoal,V),Var,Term,(N_subgoal,V)):-
                   V=(true=true),sust(Var,Term,Subgoal,N_subgoal),
```

```
                  N_subgoal=(_ box (V_,_)),V_=(Var=Term),!.

sust(Var,Term,T,Term):- Var==T,!.
sust(V1,Term,T,T):- (var(T);functor(T,F,0)),!.
sust(V1,Term,T,N_T):- T=..[F|Args],
                    sust_args(V1,Term,Args,N_Args),N_T=..[F|N_Args],!.

sust_args(_,_,[],[]).
sust_args(V1,Term,[A|R],[N_A|N_R]):-
                    sust(V1,Term,A,N_A),sust_args(V1,Term,R,N_R),!.


 /*****************************************************************
       REGLAS DE SIMPLIFICACION PARA SUBOBJETIVOS NEGADOS.
 *****************************************************************/


            /*****************************************
                     Regla de satisfaccion.
            *****************************************/


simplify_neg((_,_),T=R,(_,to_prune)):-
           nonvar(T),nonvar(R),mark(C),
           functor(T,F,A1),functor(R,G,A2),
           F\=G,cut(C),are_irreducible((F,Af),(G,Ag)),!.


            /*****************************************
                    Regla de descomposicion.
            *****************************************/


simplify_neg((Subgoal_hole,Var_hole),Formula,(Subgoal_hole_,Var_hole)):-
           neg_decom_condition(Formula,Subformulae),
           Subgoal_hole=(G box E),
           (nonvar(G),(\+ Subformulae==(true=true),
                       Var_hole=Subformulae,Subgoal_hole_=Subgoal_hole;
                       drop_conj(Var_hole,G,G_),Subgoal_hole_=(G_ box E));
            Var_hole=Subformulae,Subgoal_hole_=Subgoal_hole),!.


 /*****************************************************************
  neg_decom_condition/2 aplica la regla de descomposicion al primer
        argumento, si se cumplen las condiciones para ello.
 *****************************************************************/
```

```
neg_descom_condition(T=R,Subformulae):-
                    nonvar(T),nonvar(R),T=..[F|Af],R=..[F2|Ag],
                    F\=F2,functor(T,_,A),functor(R,_,A),
                    are_irreducible((F,A),(F2,A)),
                    (A=0,Subformulae=(true=true);
                     form_subformulae(Af,Ag,Subformulae)),!.


             /******************************************
                        Regla de sustitucion.
              *****************************************/


simplify_neg(Subgoal,T=S,Subgoal_):-
            (var(T),Var=T,Term=S;var(S),Var=S,Term=T),
            pos_sustitution_rule(Subgoal,Var,Term,Subgoal_),!.
```

```
/*****************************************************
          ***************************************
                     FRONTIER.PRO
       (Calculo de la Frontera del Objetivo Actual)
          ***************************************
*****************************************************/

 /*************************************************************
-f-->/2 implementa la relacion → f del calculo ICNcn, calculando la
  frontera del objetivo actual.  El primer argumento es un par que
  contiene la lista de variables del objetivo inicial y el objetivo
 actual, y su segundo argumento es un par que contiene la frontera y
  una lista con las nuevas variables introducidas en el calculo de
                         dicha frontera.
 *************************************************************/


        /***********************************************
             Se selecciona un redex de una ecuacion.
         ***********************************************/


(Ini_var,Goal) `-f-->' (Frontier,New_vars):-
        narred(Ini_var,Goal,Frontier,New_vars),!.


        /***********************************************
               Se selecciona un subobjetivo negado
         ***********************************************/
 /*************************************************************
 dont_care/4 selecciona un subobjetivo negado, --->ECN/2 calcula su
  frontera y si el objetivo actual no tiene que eliminarse entonces
       generation_f/5 crea la frontera del objetivo actual.
 *************************************************************/


(Ini_var,Goal) `-f-->' (Frontier,New_vars):-
        dont_care(Goal,Neg_sub,Var_hole,S_hole),
        (Ini_var,Neg_sub) '--->ECN' Frontier_n,
        (Frontier_n=to_prune,Frontier=[],New_vars=not_needed;
        generation_f(S_hole,Var_hole,Frontier_n,Frontier,New_vars)),!.


 /*************************************************************
```

```
   dont_care/4 selecciona el primer subobjetivo negado (buscando de
  izquierda a derecha) que aparece en la parte objetivo del objetivo
   actual.  Sus argumentos son:  el objetivo actual, el subobjetivo
  seleccionado, una nueva variable que representa este subobjetivo y
    el objetivo actual en el que se ha reemplazado el subobjetivo
                 seleccionado por la nueva variable.
  *************************************************************/

dont_care(C box E,Neg_eq,Var_hole,Conj_hole box E):-
                update(irr_neg(on)),
                inner_term(C,(Conj_hole,Var_hole),Neg_eq).

 /****************************************************************
  generation_f/5 reemplaza en la parte objetivo del primer argumento
 todas las ocurrencias del segundo argumento por la parte objetivo de
   cada elemento del tercer argumento (new_goal/4) y concatena las
 partes restriccion de ambos subobjetivos creando con ello una lista
  (cuarto argumento) que representa la nueva frontera y la lista de
                variables nuevas (ultimo argumento).
  *************************************************************/

generation_f(_,_,[],[],[]).
generation_f(S_hole,Var_hole,[A|R],[B|RC],[[]|Rn]):-
           update(aux_filling_holes(S_hole,Var_hole)),
           retract(aux_filling_holes(NS_hole,NVar_hole)),
           S_hole=NS_hole,A=(C box E1),
           NS_hole=(CC box E2),AA=C,cappend(E1,E2,E),
           NS_hole_=(CC box E),update(quit_trues(off)),
           new_goal(NS_hole_,Var_hole,AA,B),
           generation_f(S_hole,Var_hole,R,RC,Rn),!.

new_goal(N,V,A,B):- N==V,(A=true,B=(true=true);A=B),!.
new_goal(N,V,A,N):- var(N),!.
new_goal(N,V,A,B):- N=..[',',Arg1,Arg2],
        new_goal(Arg1,V,A,N_Arg1),new_goal(Arg2,V,A,N_Arg2),
        ((N_Arg1==(true=true),B=N_Arg2);
        (N_Arg2==(true=true),B=N_Arg1);B=(N_Arg1,N_Arg2)),!.
new_goal(N,V,A,B):- N=..[F|Args],new_goal_args(Args,V,A,N_args),
        B=..[F|N_args],!.

new_goal_args([],V,A,[]).
new_goal_args([Arg|R_arg],V,A,[B|R_B]):- new_goal(Arg,V,A,B),
                                        new_goal_args(R_arg,V,A,R_B),!.
```

```
/*******************************************************
         ***************************************
                         CNM.PRO
            (Metodo de Negacion Constructiva)
         ***************************************
 ******************************************************/

 /****************************************************************
--->ECN/2 implementa la relacion → ECN del calculo ICNcn calculando
 la frontera del objetivo actual.  El primer argumento es un par que
contiene la lista de variables del objetivo inicial y el subobjetivo
   negado, y su segundo argumento es una lista que representa su
                            frontera.
 ****************************************************************/
              /**************************************
                     reglas 1 y 2 del calculo
              *************************************/

(Ini_var,~(exist(Z,G box E))) `--->ECN' Return :-
          update(irr_neg(off)),\+inner_term(G,_,_),
          syntac_unify(G,Mgu),t_l_to_c(Mgu,CMgu),
          cappend(CMgu,E,E_theta_eqs),(solvable(~E_theta_eqs),
          take_vbles(E_theta_eqs,Vbles_),are_in(Z,Vbles_,_),
          (Vbles_=[],Fa=~E_theta_eqs;
          Fa=for_all(Vbles_,~E_theta_eqs)),
          Return=[true=true box Fa];Return=to_prune),!.

              /**************************************
                       regla 3 del calculo
              *************************************/

(Ini_var,~(exist(Z,C box E)))`--->ECN' L_one :-
          (Ini_var,C box E )`-f-->'(List,New_var),
          append_m(Z,New_var,V_rule),e_c_n(List,V_rule,L_one),!.

 /****************************************************************
    e_c_n/3 aplica las reglas de obtencion de la frontera de un
    subobjetivo negado a partir de la frontera de su subobjetivo
 complementario.  Sus argumentos son:  una lista que representa la
 frontera de su subobjetivo complementario, una lista que contiene
  las nuevas variables en esta lista y la lista que representa la
               frontera del subobjetivo negado.
 ****************************************************************/
```

```
e_c_n(List,New_vars,List_pairs_combined):-
    form_pairs(List,New_vars,List_pairsG,List_pairse),
    combined(List_pairsG,List_pairse,List_pairs_combined),!.

 /***************************************************************
 form_pairs/4 aplica la regla de descomposicion a cada elemento de la
    frontera.  Sus argumentos son cuatro listas que representan,
  respectivamente, la frontera, las variables nuevas y el primero y
  segundo de los elementos en que se descompone cada elemento de la
                          primera lista.
  **************************************************************/

form_pairs([],_,[],[]).
form_pairs([Gi box Ei|T],[N_V|R_N_V],
          [~exist(N_V,Gi box Ei)|T_],[Fa|TT]):-
                 (Ei=for_all(L,_),append(L,N_V,A),
                 (A=[],Fa=~Ei;Fa=for_all(A,~Ei));
                 (N_V=[],Fa=~Ei;Fa=for_all(N_V,~Ei))),
                 form_pairs(T,R_N_V,T_,TT),!.

 /***************************************************************
   combined/3 combina los elementos de los dos primeros argumentos
 construyendo con ellos el tercero.  Cada argumento es una lista que
  contiene, respectivamente, el primer y segundo elemento en que se
 descompone cada objetivo al aplicar form_pairs/4 y la combinacion de
                              estos.
  **************************************************************/

combined(L_g,L_e,L_c):- subsets_of(S1,L_g),subsets_of(S2,L_e),
                        inverse(S2,S2_i),put_pairs(S1,S2_i,L_c).
put_pairs([],[],[]):-!.
put_pairs([S1|R_S1],[S2_i|R_S2_i],[S1_ box S2_i_|R]):-
                        t_l_to_c(S1,S11_),t_l_to_c(S2_i,S22_i_),
                        S1_=S11_,S2_i_=S22_i_,put_pairs(R_S1,R_S2_i,R).

              /**************************************
                        regla 4 del calculo
              ************************************/

(Ini_var,~(exist(Z,C box E)))'--->ECN'
                             [true=true box true=true]:- !.
```

```
/*****************************************************
          ***************************************
                        NARRER.PRO
           (Algoritmo de Innermost Narrowing)
          ***************************************
 *****************************************************/


    /*******************************************************
 narred/3 realiza un paso de innermost narrowing reduciendo un redex
       innermost con todas las posibles reglas del programa.
     *******************************************************/


narred(Ini_var,C box E,New_goals,New_var):-
             update(irr_neg(off)),
             inner_term(C,(C_hole,Var_hole),Inner_term),
             look_rules(Inner_term,L_rules),
             one_step_reduce(Ini_var,(C_hole,Var_hole),E,
                    Inner_term,L_rules,New_goals,New_var),!.


    /*******************************************************
  inner_term/3 selecciona un termino innermost (tercer argumento) de
  una conjuncion de ecuaciones (primer argumento) sustituendo este
   termino en la conjuncion por una variable (segundo argumento).
     *******************************************************/


inner_term(T,(Term_hole,Var_hole),Term):-
            \+ var(T),functor(T,F,N_args),
            T=..[F|L_args],F \= (~),
            inner_term_args(L_args,(L_args_hole,Var_hole),Term),
            Term_hole=..[F|L_args_hole].
inner_term(T,(Var_hole,Var_hole),T):-
            \+ var(T),functor(T,F,N_args),
            irre(L_irr),\+ member((F,N_args),L_irr),!.


inner_term_args([Arg|R_arg],([Arg_hole|R_Arg_hole],Var_hole),Term):-
              (inner_term_args(R_arg,(R_Arg_hole,Var_hole),Term),
              Arg_hole=Arg;
              inner_term(Arg,(Arg_hole,Var_hole),Term),
                        R_args=R_Args_hole).
```

```
     /*******************************************************
look_rules/2 construye la lista de todas las reglas del programa cuya
 parte izquierda de la cabeza unifica con el termino que aparece como
                         su primer argumento.
     *******************************************************/

look_rules(S,L):- update(aux_l(X,X)),rclause(rwr(Left,_,_),_,D),
          Left=S,aux_l(L,X),X= [D|X_],update(aux_l(L,X_)),fail.

look_rules(_,L):- aux_l(L,[]).

     /*******************************************************
   one_step_reduce/7 realiza un paso de narrowing condicional.  Sus
                           argumentos son:
     1) Lista de las variables del objetivo inicial.
     2) Parte objetivo del objetivo actual.
     3) Parte restriccion del objetivo actual.
     4) Termino a reducir.
     5) Lista de reglas con las que se puede reducir el termino.
     6) Lista de los nuevos objetivos.
     7) Lista de las variables nuevas introducidas en el paso de
     reduccion.
     *******************************************************/

one_step_reduce(Ini_var,_,_,_,[],[],[]).

one_step_reduce(Ini_var,Goal_part,Const_part,Term,[Rule|R_rules],
        [New_goal|R_New_goals],[New_var|R_New_var]):-
            apply_rule(Ini_var,Goal_part,Const_part,Term,
                       Rule,New_goal,New_var),
            one_step_reduce(Ini_var,Goal_part,Const_part,Term,
                       R_rules,R_New_goals,R_New_var).

     /*******************************************************
apply_rule/7 reduce un termino aplicando una regla del programa.  Sus
                           argumentos son:
    1) Lista de las variables del objetivo inicial.
    2) Parte objetivo del objetivo actual.
    3) Parte restriccion del objetivo actual.
    4) Termino a reducir.
    5) Regla del programa con la que se va a reducir el termino.
    6) Nuevo objetivo.
    7) Lista de las variables nuevas introducidas en el paso de
    reduccion.
```

```
    ******************************************************/

apply_rule(Ini_var,Goal_hole,Const_part,Term,Rule,
                 New_goal,New_var):-
          rclause(rwr(N_term,Rhs,Body),_,Rule),
          take_mgu(Term,N_term,Mgu,New_var),
          apply_Mgu_to(New_var,Mgu),
          update(aux_apply_rule(Goal_hole)),
          retract(aux_apply_rule(New_Goal_hole)),
          Goal_hole = New_Goal_hole,
          use_mgu(Mgu,(Rhs,Goal_hole,Body),NGB,NMgu,N_v,New_var),
          (Rhs,DO,RE)=NGB,(DO,RE)=NGB_,
          sust_Rhs(NGB_,Const_part,NMgu,Rhs,N_goal),
          apply_sust(NMgu,N_goal,New_goal),!.

    /******************************************************
 apply_Mgu_to/2 aplica el mgu (segundo argumento) a los elementos del
                          primer argumento.
    ******************************************************/

apply_Mgu_to([],_).
apply_Mgu_to([V|R_vars],Mgu):- (is_eq_in(V=T,Mgu),call(V=T);true),
                            apply_Mgu_to(R_vars),!.

    /******************************************************
 use_mgu/6 elimina las ecuaciones de la forma V1=V2 (componentes del
   primer argumento) del segundo argumento, reemplazando todas las
 ocurrencias de V1 por V2 (apply_use/4) y devolviendo el resultado en
    el tercer argumento y el mgu en el cuarto.  Los dos ultimos
  argumentos mantienen actualizadas las variables del rango del mgu.
    ******************************************************/

use_mgu([],G,G,[],NV,NV).
use_mgu([V1=V2|R],G,NGB,RR,NV,NV_):-
          var(V1),var(V2),apply_use(V1,V2,G,GB),
          drop_list(V2,NV,NNV),use_mgu(R,GB,NGB,RR,NNV,NV_),!.
use_mgu([E|R],(Goal_hole,Body),NGB,[E|RR],NV,NV_):-
          use_mgu(R,(Goal_hole,Body),NGB,RR,NV,NV_),!.

apply_use(V1,V2,T,V1):-T==V2,!.
apply_use(V1,V2,T,T):- (var(T);functor(T,F,0)),!.
```

```
apply_use(V1,V2,T,TT):- T=..[F|Args],
          apply_use_arg(V1,V2,Args,NArgs),TT=..[F|NArgs],!.

apply_use_arg(V1,V2,[],[]).
apply_use_arg(V1,V2,[A|R],[NA|RR]):- apply_use(V1,V2,A,NA),
                                     apply_use_arg(V1,V2,R,RR),!.


     /*******************************************************
sust_Rhs/5 aplica una regla del programa al objetivo actual (primer y
 segundo argumentos) y devuelve el resultado en el ultimo argumento.
     *******************************************************/


sust_Rhs((Goal_hole,Body),Const_part,Mgu,Rhs,New_goal):-
          Goal_hole=(Conj,Hole),
          (Body=true,N_goal=Conj;add_body(Conj,Body,N_goal)),
          apply_sust_rhs(N_goal,Mgu,Hole,Rhs,New_goal_),
          t_l_to_c(Mgu,CMgu),
          cappend(CMgu,Const_part,New_Const_part),
          New_goal=(New_goal_ box New_Const_part),!.


     /*******************************************************
    add_body/3 añade el segundo argumento (cuerpo de una regla de
  programa) al primero (un objetivo) devolviendo el resultado como
               tercer argumento (nuevo objetivo).
     *******************************************************/


add_body((C,R_C),Body,CC):- C==(true=true),add_body(R_C,Body,CC),!.
add_body((C,R_C),Body,C_):- add_body(R_C,Body,CC),
                            (CC==(true=true),C_=C;C_=(C,CC)),!.
add_body(C,Body,Body):- C==(true=true),!.
add_body(C,Body,(C,Body)):-!.


     /*******************************************************
apply_sust_rhs/5 sustituye la variable que representa al redex por la
 parte derecha de la cabeza de una regla y lo instancia todo con el
                                mgu.
     *******************************************************/


apply_sust_rhs(Term,Mgu,Hole,Rhs,New_Term):-
                 var(Term),Term==Hole,New_Term=Rhs,!.
apply_sust_rhs(Term,Mgu,Hole,Rhs,New_Term):-
```

```
                  var(Term),(is_eq_in(Term=T,Mgu),
                             New_Term=T;New_Term=Term),!.
apply_sust_rhs(Term,Mgu,Hole,Rhs,Term):- functor(Term,_,0),!.
apply_sust_rhs(Term,Mgu,Hole,Rhs,New_Term):-
                  Term=..[F|T_arg],
                  sust_rhs_arg(T_arg,Mgu,Hole,Rhs,N_arg),
                  New_Term=..[F|N_arg],!.

sust_rhs_arg([],_,_,_,[]).
sust_rhs_arg([Arg|Rarg],Mgu,Hole,Rhs,[N_arg|R_arg]):-
                  apply_sust_rhs(Arg,Mgu,Hole,Rhs,N_arg),
                  sust_rhs_arg(Rarg,Mgu,Hole,Rhs,R_arg),!.


     /**********************************************************
   apply_sust/3 instancia el segundo argumento con el primero.  El
            resultado se devuelve como tercer argumento.
     *********************************************************/


apply_sust([],Ng,Ng).
apply_sust([V=T|Rs],Ng,Ng_):- apply_sust(V,T,Ng,NNg),
                             apply_sust(Rs,NNg,Ng_),!.
apply_sust(Var,Term,T,Term):- Var==T,!.
apply_sust(V1,Term,T,T):- (var(T);functor(T,F,0)),!.
apply_sust(V1,Term,T,N_T):-
     T=..[F|Arg],apply_sust_arg(V1,Term,Arg,N_Arg),N_T=..[F|N_Arg],!.

apply_sust_arg(_,_,[],[]).
apply_sust_arg(V1,Term,[A|R],[N_A|N_R]):-
     apply_sust(V1,Term,A,N_A),
     apply_sust_arg(V1,Term,R,N_R),!.
```

```
/*****************************************************
        ***************************************
                       SOLVER.PRO
                (Analizador de Respuestas)
        ***************************************
 *****************************************************/

 ?- op(100,fy,no).       /* azucar negacion */
 ?- op(150,yfx,#).       /* azucar or */
 ?- op(200,yfx,&).       /* azucar and */
 ?- op(730,fx,for_all).  /* azucar ∀ */

    /*****************************************************
       solvable/1 aplica el algoritmo de Maher para resolver
  sintacticamente sistemas con negacion y cuantificacion cuando la
                    signatura es finita.
    *****************************************************/

solvable(X):- update(solv(X)),retract(solv(Y)),dnf(Y,C),
              signa(SS),(solver(C,SS,L),write('is solvable'),nl;
                         write('not is solvable'),nl,fail),nl,write(L),!.

    /*****************************************************
   dnf/2 calcula la forma normal disyuntiva del primer argumento.
    *****************************************************/

dnf(X,Y):- t_n(Y1,Z1),r_q(Z1,Z),distr_y_o(Z,Y_),tr(Y_,Y),!.

    /*****************************************************
       t_n/2 Introduce el operador 'no' dentro de la formula.
    *****************************************************/

t_n(~(X,Y),V # W):- !,t_n(~(X),V),t_n(~(Y),W).
t_n(~(X # Y),(V,W)):- !,t_n(~(X),V),t_n(~(Y),W).
t_n~exist(X,Y),for_all(X,Z)):- !,t_n(~(Y),Z).
t_n(~for_all(X,Y),exist(X,Z)):- !,t_n(~(Y),Z).
t_n(~~X,Z):- !,t_n(X,Z).
t_n(exist(X,Y),exist(X,Z)):- !,t_n(Y,Z).
t_n(for_all(X,Y),for_all(X,Z)):- !,t_n(Y,Z).
t_n((X,Y),(V,W)):- !,t_n(X,V),t_n(Y,W).
t_n(X # Y,V # W):- !,t_n(X,V),t_n(Y,W).
t_n(X,X).
```

```
/*********************************************************
r_q/2 a partir de su primer argumento, cuantificador y formula
cuantificada, elimina los cuantificadores de esta ultima.
*********************************************************/

r_q(for_all(X,Y),for_all(X,V)):- !,r_q(Y,V).
r_q(exist(X,Y),exist(X,V)):- !,r_q(Y,V).
r_q(X & Y,Z):- !,r_q(X,V),r_q(Y,W),r_q_o(V & W,Z).
r_q(X # Y,Z):- !,r_q(X,V),r_q(Y,W),r_q_o(V # W,Z).
r_q(X,X).

/*********************************************************
r_q_o/2 pone los cuantificadores fuera del alcance de los operadores.
*********************************************************/

r_q_o(for_all(X,Y) & Z,for_all(X,V)):- !,r_q_o(Y & Z,V).
r_q_o(for_all(X,Y) # Z,for_all(X,V)):- !,r_q_o(Y # Z,V).
r_q_o(exist(X,Y) & Z,exist(X,V)):- !,r_q_o(Y & Z,V).
r_q_o(exist(X,Y) # Z,exist(X,V)):- !,r_q_o(Y # Z,V).
r_q_o(X & for_all(Y,Z),for_all(Y,V)):- !,r_q_o(X & Z,V).
r_q_o(X # for_all(Y,Z),for_all(Y,V)):- !,r_q_o(X # Z,V).
r_q_o(X & exist(Y,Z),exist(Y,V)):- !,r_q_o(X & Z,V).
r_q_o(X # exist(Y,Z),exist(Y,V)):- !,r_q_o(X # Z,V).
r_q_o(X,X).

/*********************************************************
distr_y_o/2 aplica la propiedad distributiva del operador and
respecto del or.
*********************************************************/

distr_y_o(for_all(X,Y),for_all(X,Z)):- !,distr_y_o(Y,Z).
distr_y_o(exist(X,Y),exist(X,Z)):- !,distr_y_o(Y,Z).
distr_y_o((X,(Y # Z)),Z1 # Z2):- !,distr_y_o((X,Y),Z1),
                        distr_y_o((X,Z),Z2).
distr_y_o(((X # Y),Z),Z1 # Z2):- !,distr_y_o((X,Z),Z1),
                        distr_y_o((Y,Z),Z2).
distr_y_o(X # Y,V # W):- !,distr_y_o(X,V),distr_y_o(Y,W).
distr_y_o(X,X):- conjun(X),!.
distr_y_o((X,Y),M):- !,distr_y_o(X,Z),distr_y_o(Y,W),
                        distr_y_o((Z,W),M).

conjun(Y # X):- !,fail.
conjun((Y,X)):- !,conjun(X),conjun(Y).
conjun(X).
```

```
     /*******************************************************
                                tr/2 .
      *******************************************************/


tr(Exp,[L|L_]):- tr_(Exp,LL,R_),tr__(R_,R),ap_ex(LL,R_,L),
                 d_out_rbf(R,L1),d_out_conj(L1,L_),!.
tr_(exist(LV,R),L,Re):- tr_c_e(LV,L1),tr_(R,L_,Re),append_q(L1,L_,L),!.
tr_(for_all(LV,R),L,Re):- tr_c_a(LV,L1),tr_(R,_,Re),append_q(L1,L_,L),!.
tr_(Exp,[],Exp).
tr_c_e([],[]).
tr_c_e([V|R],R_):- (\+ is_in(V,R),R_=[e(V)|RR];R_=RR),tr_c_e(R,RR).

tr_c_a([],[]).
tr_c_a([V|R],R_):- (var(V),(\+ is_in(V,R),
                 R_=[a(V)|RR];R_=RR);R_=RR),tr_c_a(R,RR).

tr__(A # B,L):- tr__(A,La),tr__(B,Lb),append(La,Lb,L),!.
tr__(A,[La]):- c_to_l_m(A,La),!.


     /*******************************************************
                              ap_ex/3 .
      *******************************************************/


ap_ex(L,R,LR):- take_vbles(L,VL),take_vbles(R,VR),
                append_q(VR,VL,LR_),append_ex(LR_,L,LRR),
                inverse(LRR,LR).

append_ex([],L,L).
append_ex([V|R],L,[e(V)|RR]:- append_ex(R,L,RR).


     /*******************************************************
   solver/3 aplica el algoritmo de Maher para signaturas finitas al
     sistema computado en el modulo ICNCN.pro.  Si el sistema es
  resoluble entonces calcula su forma resuelta.  Sus argumentos son:
 el sistema de entrada, la signatura y la forma resuelta del sistema.
      *******************************************************/


solver([Quan,List_sc],Signat,List_fin):-
       do_bf(List_sc,List_bf1),out_false(List_bf1,List_bf,Bool),
       (Bool=true;(!,solv_fin([Quan,List_bf],Signat,List_fin),!,
       write(List_fin),eval(List_fin))).
```

```
/********************************************************
do_bf/2 transforma el sistema de ecuaciones en formulas basicas.
*******************************************************/

do_bf(List_eq,Formu):- update(f(no)),do_c_bf(List_eq,Form_nf),
                       put_eq_param(Form_nf,Form_nf,Formu1),
                       (f(no);!,append(Formu1,[[f]],Formu2)),
                       ((f(no),append(Formu1,[[end]],Formu));!,
                       append(Formu2,[[end]],Formu)).

/********************************************************
do_c_bf/2 transforma cada conjuncion de ecuaciones en una conjuncion
                de formulas basicas.
*******************************************************/

do_c_bf([],[]).
do_c_bf([First_c|Rest_c],Formb):-
       do_basicf(First_c,For_bas),out_nil(For_bas,For_c_bas),
       do_c_bf(Rest_c,Formb_r),append([For_c_bas],Formb_r,Formb).
do_c_bf([First_c|Rest_c],Formb):-
       update(f(si)),!,do_c_bf(Rest_c,Formb).

/********************************************************
do_basicf/2 transforma cada conjuncion de ecuaciones en forma
resuelta (solved/2), sustituye cada parametro por una nueva variable
      (put_param/3) añadiendo la ecuacion parametro=variable.
*******************************************************/

do_basicf(List_eq,Formu):-
         div(List_eq,List_noneg,List_neg),
         solved(List_noneg,Solv_eq),!,
         put_param(Solv_eq,L_quantif,L_eq),
         do_basicf1(List_neg,Rest_for),
         append([L_quantif,L_eq],Rest_for,Formu).

do_basicf1([],[]).
do_basicf1([Eq_neg|Rest],Formu):-
         Eq_neg=..[~|Eq],solved(Eq,Solv_eq),!,
         put_param(Solv_eq,L_quantif,L_eq),
         update(flag1(no)),neg(L_quantif,L_quan_neg),
         do_basicf1(Rest,Rest_for),
         append([L_quan_neg,L_eq],Rest_for,Formu).
```

```
do_basicf1([Eq_neg|Rest],Formu):-
         Eq_neg=..[~|Eq],\+(solved(Eq,Solv_eq)),!,
         do_basicf1(Rest,Formu).

     /*******************************************************
    div/3 divide la lista de ecuaciones en dos listas, una con las
      ecuaciones positivas y la otra con las ecuaciones negadas.
      *******************************************************/

div([],[],[]).
div([(A=B)|Rest],[(A=B)|List_noneg],List_neg):-
    !,div(Rest,List_noneg,List_neg).
div([~(A=B)|Rest],List_noneg,[~(A=B)|List_neg]):-
    !,div(Rest,List_noneg,List_neg).

     /*******************************************************
    solved/2 aplica el algoritmo de FORMA RESUELTA a un sistema de
                          ecuaciones.
      *******************************************************/

solved(Temp_Constr,F_Constr):-
       update(flag(no)),reduc1(Temp_Constr,R_Constr),
       reduc2(R_Constr,P_Constr),vars_to_left(P_Constr,T1_Constr),
       rem_rep(T1_Constr,T_Constr),
       reduc3(T_Constr,T_Constr,S_Constr),
       ((flag(si),solved(S_Constr,F_Constr)) ,!;(flag(no),
       append([],S_Constr,F_Constr))).

     /*******************************************************
      reduc1/2 implementa la regla de reemplazo:  f(t1,...,tn) =
                 f(s1,...,sn)⇒ t1 = s1,...,tn = sn.
      *******************************************************/

reduc1([],[]):- !.
reduc1([(Left=Right)|Rest_Constr],R1_Constr):-
       nonvar(Left),nonvar(Right),
       functor(Left,F1,A1),functor(Right,F1,A1),A1>0,!,
       Left=..[F1|Args1],Right=..[F1|Args2],
       join_args(Args1,Args2,L_eq),
       append(Rest_Constr,L_eq,New_Constr),
       update(flag(si)),reduc1(New_Constr,R1_Constr).
reduc1([(Left=Right)|Rest_Constr],[(Left=Right)|R1_Constr]):-
       ((nonvar(Left),nonvar(Right),Left==Right),!;
       (var(Left),!;var(Right))),>reduc1(Rest_Constr,R1_Constr).
```

```
/*********************************************************
join_args/3 forma una lista de ecuaciones con las variables que
aparecen en dos listas.
*********************************************************/

join_args([A1|R_args1],[A2|R_args2],R):-
        append([(A1=A2)],L,R),join_args(R_args1,R_args2,L).
join_args([],[],[]).

/*********************************************************
reduc2/2 elimina las ecuaciones triviales, X=X.
*********************************************************/

reduc2([],[]):-!.
reduc2([(Left=Right)|R_const],T_Const):-
       Left==Right,!,update(flag(si)),reduc2(R_const,T_Const).
reduc2([(Left=Right)|R_const],[(Left=Right)|T_Const]):-
       reduc2(R_const,T_Const).

/*********************************************************
vars_to_left/2  cambia las ecuaciones t=X por X=t, siendo X una
variable y t un termino.
*********************************************************/

vars_to_left([],[]):-!.
vars_to_left([(Left=Right)|Rest_Constr],[(Right=Left)|Rest]):-
            var(Right), nonvar(Left),!,update(flag(si)),
            vars_to_left(Rest_Constr,Rest).
vars_to_left([(Left=Right)|Rest_Constr],[(Left=Right)|Rest]):-
            vars_to_left(Rest_Constr,Rest).

/*********************************************************
reduc3/3 para todas las ecuaciones de la forma X=t, aplica la
sustitucion {X/t} al resto.
*********************************************************/

reduc3([],L,L):-!.
reduc3([(Left=Right)|R_Const],L_Const,N_Const):-
       var(Left),take_vbles(Right,L_var),\+(is_in(Left,L_var)),!,
       apply_sust([[Left,Right]],L_Const,N_L_Cons),
```

```
       rem_rep(N_L_Cons,N_L_Cons1),
       find_eq((Left=Right),N_L_Cons1,Nc_L_Cons),
       reduc3(Nc_L_Cons,N_L_Cons1,N_Const).
reduc3([(Left=Right)|R_Const],L_Const,N_Const):-
       var(Left),take_vbles(Right,L_var),\+(is_in(Left,L_var)),!,
       reduc3(R_Const,L_Const,N_Const).
reduc3([(Left=Right)|R_Const],L_Const,N_Const):-
       var(Right),take_vbles(Left,L_var),\+(is_in(Right,L_var)),!,
       reduc3(R_Const,L_Const,N_Const).
reduc3([(Left=Right)|R_Const],L_Const,N_Const):-
       \+(var(Right)),\+(var(left)),
       functor(Right,F1,A1),functor(Left,F1,A1),A1>0,
       !,reduc3(R_Const,L_Const,N_Const).
reduc3([(Left=Right)|R_Const],L_Const,N_Const):-
       Left==Right,reduc3(R_Const,L_Const,N_Const).


     /*********************************************************
 find_eq/3 busca una ecuacion en una lista y devuelve el resto de la
                  lista que queda por examinar.
      *********************************************************/


find_eq((Left=Right),[(Left1=Right1)|R],R):-
    (Left=Right)==(Left1=Right1).
find_eq((Left=Right),[(Left1=Right1)|R],Rl):-
    (Left=Right)\==(Left1=Right1),find_eq((Left=Right),R,Rl).


     /*********************************************************
         put_param/4 sustituye los parametros por variables.
      *********************************************************/


put_param(Solv_eq,L_quantif,L_eq):-
          put_eq_qu(Solv_eq,Solv_eq,[],L_quantif,L_eq,[]).

put_eq_qu([],L,K,K,L,_).
put_eq_qu([(Left=Right)|Rest],L_eq_en,L_quan,
                 L_quan1,Ln_eq_sc,L_param):-
          take_vbles(Right,L_vars),L_vars\=nil,!,
          do_substi(L_vars,L_eq_en,L_quan,L_quantif,
                 L_eq,L_param,Nl_param),
          put_eq_qu(Rest,L_eq,L_quantif,L_quan1,Ln_eq_sc,Nl_param).
put_eq_qu([(Left=Right)|Rest],L_eq_en,L_quan,
                 L_quantif,L_eq,L_param):-
```

```
        !,put_eq_qu(Rest,L_eq_en,L_quan,L_quantif,L_eq,L_param).

do_substi([],K,J,J,K,L,L).
do_substi([X|Rest],Rest_nt,L_quan,[e(X1)|L_q],
                [(X=X1)|Rest_dst1],L_param,Nl1_param):-
        \+(is_in(X,L_param)),!,apply_sust([[X,X1]],
        Rest_nt,Rest_st),append(L_param,[X],Nl_param),
        do_substi(Rest,Rest_st,L_quan,L_q,
                Rest_dst1,Nl_param,Nl1_param).
do_substi([X|Rest],Rest_nt,L_quan,L_quantif,
                Rest_st,L_param,Nl1_param):-
        !,do_substi(Rest,Rest_nt,L_quan,L_quantif,
                Rest_st,L_param,Nl1_param).


     /*********************************************************
     out_nil/2 elimina de la lista la pareja de nil que la pueda
                          encabezar.
     ********************************************************/

out_nil([nil,nil|Rest],List):- !,out_nil(Rest,List).
out_nil(List,List).
out_nil([],[]).


     /*********************************************************
     neg/2 niega los cuantificadores existenciales de una lista.
     ********************************************************/

neg([],[]):- flag1(si),!.
neg([],[~]).
neg([e(X)|L_quan],[~(e(X))|L_q_neg]):-
    update(flag1(si)),!,neg(L_quan,L_q_neg).


     /*********************************************************
 put_eq_param/3 propaga las ecuaciones de la forma parametro=variable
                 al resto de las conjunciones.
     ********************************************************/

put_eq_param([],Form_sf,Form_sf).
put_eq_param([[]|Rest_tot],Form_nf,Form_sf):-
   !,put_eq_param(Rest_tot,Form_nf,Form_sf).
   put_eq_param([[_,[(X=Y)|Rest]|Rest_tot_c]|
```

```
               Rest_tot],Form_nf,Formu):-
   var(X),Rest=nil,!,make_put((X=Y),Form_nf,Form_Sf),
   put_eq_param([Rest_tot_c|Rest_tot],Form_Sf,Formu).
put_eq_param([[_,[(X=Y)|Rest]|Rest_tot_c]|Rest_tot],Form_nf,Formu):-
   var(X),!,make_put((X=Y),Form_nf,Form_Sf),
   put_eq_param([[_,Rest|Rest_tot_c]|Rest_tot],
               Form_Sf,Formu).
put_eq_param([[_,[(X=Y)|Rest]|Rest_tot_c]|Rest_tot],Form_nf,Formu):-
   Rest=nil,!,put_eq_param([Rest_tot_c|Rest_tot],Form_nf,Formu).
put_eq_param([[_,[(X=Y)|Rest]|Rest_tot_c]|Rest_tot],Form_nf,Formu):-
   !,put_eq_param([[_,Rest|Rest_tot_c]|Rest_tot],Form_nf,Formu).


     /********************************************************
 make_put/3 incluye una ecuacion con su correspondiente cuantificador
                   en el resto de formulas basicas.
      ********************************************************/


make_put(_,[],[]):-!.
make_put((X=Y),[[]|Rest],[[]|Rest1]):-
         !,make_put((X=Y),Rest,Rest1).
make_put((X=Y),[[L_qua,L_eq|Rest_c]|Rest],
              [[List_quan,List_eq|Form_sf]|Rest1]):-
         \+(is_var_in(X,L_eq)),((L_qua=[e(Z)|_],!);L_qua=nil),!,
         append(L_qua,[e(Y1)],List_quan),
         append(L_eq,[(X=Y1)],List_eq),
         make_put((X=Y),[Rest_c|Rest],[Form_sf|Rest1]).
make_put((X=Y),[[L_qua,L_eq|Rest_c]|Rest],
              [[List_quan,List_eq|Form_sf]|Rest1]):-
         \+(is_var_in(X,L_eq)),L_qua=[~(e(Z))|_],!,
         append(L_qua,[~(e(Y1))],List_quan),
         append(L_eq,[(X=Y1)],List_eq),
         make_put((X=Y),[Rest_c|Rest],[Form_sf|Rest1]).
make_put((X=Y),[[L_qua,L_eq|Rest_c]|Rest],
              [[List_quan,List_eq|Form_sf]|Rest1]):-
         \+(is_var_in(X,L_eq)),L_qua=[~],!,
         append([],[~(e(Y1))],List_quan),
         append(L_eq,[(X=Y1)],List_eq),
         make_put((X=Y),[Rest_c|Rest],[Form_sf|Rest1]).
make_put((X=Y),[[L_quan,L_eq|Rest_c]|Rest],
              [[L_quan,L_eq|Form_sf]|Rest1]):-
         !,make_put((X=Y),[Rest_c|Rest],[Form_sf|Rest1]).


     /********************************************************
```

```
 solv_fin/2 sustituye iterativamente cada pareja de formulas basicas
    por una combinacion booleana equivalente de formulas basicas,
          eliminando en cada paso un cuantificador.
     ****************************************************/


/************************
CUANTIFICADOR EXISTENCIAL
************************/
solv_fin([[],L_eq],_,L_eq):-!.
solv_fin([Quan,L_eq],Signat,Dis_c_bf):-
        take_last(Quan,Last_Quan),Last_Quan=e(W),!,
        solved_f_fin(Quan,L_eq,e(W),Signat,Dis_c_bf1),
        rem_var(Last_Quan,Quan,Quan_out),
        solv_fin([Quan_out,Dis_c_bf1],Signat,Dis_c_bf).
/**********************
CUANTIFICADOR UNIVERSAL
**********************/
solv_fin([Quan,L_eq],Signat,Dis_c_bf):-
        take_last(Quan,Last_Quan),
        Last_Quan=a(W),!,distrib(L_eq,L_eq_n),
        solved_f_fin(Quan,L_eq_n,e(W),Signat,Dis_c_bf1),
        distrib(Dis_c_bf1,Dis_c_bfn),
        rem_var(Last_Quan,Quan,Quan_out),
        solv_fin([Quan_out,Dis_c_bfn],Signat,Dis_c_bf).

solved_f_fin(_,[],_,_,[]):-!.
solved_f_fin(Quan,[L_eq|Rest],e(W),Signat,List_lem):-
        (Quan=[e(Y)|Rest_quan];Quan=[a(Y)|Rest_Quan]),
        (W\==Y;\+(only_onevar(Y,[L_eq|Rest]))),
        L_eq=[A,B|C],(A=nil;A=[e(T)|RR]),C==nil,!,
        solved_c(nf,[0,L_eq],e(W),List_aplem),
        solved_f_fin(Quan,Rest,e(W),Signat,List_lem1),
        append([List_aplem],List_lem1,List_lem).
solved_f_fin(Quan,[L_eq|Rest],e(W),Signat,List_lem):-
        (Quan=[e(Y)|Rest_quan];Quan=[a(Y)|Rest_Quan]),W==Y,
        L_eq=[A,B|C],C==nil,(A=nil;A=[e(T)|RR]),!,
        solved_c(f,[0,L_eq],e(W),List_aplem),
        solved_f_fin(Quan,Rest,e(W),Signat,List_lem1),
        append([List_aplem],List_lem1,List_lem).
solved_f_fin(Quan,[L_eq|Rest],e(W),Signat,List_lem):-
        (Quan=[e(Y)|Rest_quan];Quan=[a(Y)|Rest_quan]),
        (W \==Y;\+(only_onevar(Y,[L_eq|Rest]))),!,
        enter_E(L_eq,Signat,List_dca),List_dca=[Conj_dca|Rest_dca],
        solved_c(nf,[∧,Conj_dca],e(W),List_aplem),
```

```
        List_aplem=[Quan_noneg,Eq_noneg|Rest_neg],
        out_equal_bf(Rest_neg,List_aplem_neg),
        append([Quan_noneg,Eq_noneg],List_aplem_neg,List_Faplem),
        append(Rest_dca,Rest,Rest_new),
        solved_f_fin(Quan,Rest_new,e(W),Signat,List_lem1),
        append([List_Faplem],List_lem1,List_lem).
solved_f_fin(Quan,[L_eq|Rest],e(W),Signat,List_lem):-
        (Quan=[e(Y)|Rest_quan];Quan=[a(Y)|Rest_Quan]),W==Y,!,
        enter_E(L_eq,Signat,List_dca),List_dca=[Conj_dca|Rest_dca],
        solved_c(f,[∧,Conj_dca],e(W),List_aplem),
        append(Rest_dca,Rest,Rest_new),
        solved_f_fin(Quan,Rest_new,e(W),Signat,List_lem1),
        append([List_aplem],List_lem1,List_lem).

     /*********************************************************
 solved_c/5 introduce el cuantificador existencial a eliminar en una
          conjuncion de formulas basicas y simplifica esta.
      *********************************************************/

solved_c(_,[_,[]],_,[]):- !.
solved_c(_,[∧,[Quan1,Bf]],_,[]):- !.
solved_c(F,[0,[Quan1,Bf]],Last_Quan,List_aplem):-
        (Last_Quan=e(W),!;Last_Quan=a(W)),!,
        solved_pair(F,nil,[Quan1,Bf],W,List_aplem).
solved_c(F,[∧,[Quan1,Bf|List_eq]],Last_Quan,List_aplem):-
        (Quan1=[~(e(X))|R];Quan1=[~|R]),Last_Quan=e(W),!,
        solved_c(F,[∨,[Quan1,Bf|List_eq]],Last_Quan,List_aplem).
solved_c(F,[∧,[Quan1,Bf,Quan2,Bf_neg|List_eq]],
               Last_Quan,List_aplem):-
        Last_Quan=e(W),!,
        solved_pair(F,[Quan1,Bf],[Quan2,Bf_neg],W,Bf_out),
        solved_c(F,[∧,[Quan1,Bf|List_eq]],Last_Quan,List_ap),
        append(Bf_out,List_ap,List_aplem).
solved_c(F,[∨,[[~|R],Bf_neg|Rest]],Last_Quan,List_aplem):-
        Last_Quan=e(W),!,
        solved_pair(F,nil,[[~(e(C))|R],Bf_neg],W,Bf_out),
        solved_c(F,[∨,Rest],Last_Quan,List_ap),
        append(Bf_out,List_ap,List_aplem).
solved_c(F,[∨,[Quan,Bf|Rest]],Last_Quan,List_aplem):-
        Last_Quan=e(W),!,solved_pair(F,nil,[Quan,Bf],W,Bf_out),
        solved_c(F,[∨,Rest],Last_Quan,List_ap),
        append(Bf_out,List_ap,List_aplem).

     /*********************************************************
```

```
solved_pair/5 introduce el cuantificador existencial a la pareja
   escogida de formulas basicas, simplifica esta y elimina el
                       cuantificador.
   ******************************************************/


solved_pair(f,[Quan,Bf],[Quan_neg,Bf_neg],_,[[e(true)],true]):-
            \+(is_solv(Bf,Bf_neg,_)),!.
solved_pair(f,[Quan,Bf],[Quan_neg,Bf_neg],_,[[e(false)],false]):-
            Bf=[(X=Y)],Bf_neg=[(U=V)],solved([(Y=V)],L),L=nil,!.
solved_pair(f,[Quan,Bf],[Quan_neg,Bf_neg],_,[[e(true)],true]):-
            Bf=[(X=Y)],Bf_neg=[(U=V)],
            solvedfin([(Y=V)],L),L=[(K=P)],var(K),\+(var(P)),!.
solved_pair(f,[Quan,Bf],[Quan_neg,Bf_neg],_,[[e(false)],false]):-
            Bf=[(X=Y)],Bf_neg=[(U=V)],
            solvedfin([(Y=V)],L),equ_vars(L),!.
solved_pair(f,nil,[[~(e(C))|Rest],Bf_neg],_,[[e(true)],true]):-
            Bf_neg=[(X=Y)],\+(var(Y)),!.
solved_pair(f,nil,[[~(e(C))|Rest],Bf_neg],_,[[e(false)],false]):-
            Bf_neg=[(X=Y)],var(Y),!.
solved_pair(f,nil,[nil,Bf],_,[[e(true)],true]):-!.
solved_pair(f,nil,[[~|Rest],Bf_neg],_,[[e(true)],true]):- !.
solved_pair(f,[Quan,Bf],[Quan_neg,Bf_neg],_,[[e(true)],true]).
solved_pair(nf,nil,[Quan,Bf],W,Bf_out):-
            (Quan=[~(e(X))|R];Quan=[~|R]),!,
            make_taut(Bf,Bf_t),put_quant([Bf_t],Bf_qu),
            solved_pair(nf,Bf_qu,[Quan,Bf],W,Bf_out).
solved_pair(nf,nil,[Quan,Bf],W,Bf_out):-
            !,out_W([Quan,Bf],W,Bf_out).
solved_pair(nf,[Quan,Bf],[Quan_neg,Bf_neg],W,Bf_out):-
            is_solv(Bf,Bf_neg,List_s),is_inconst(Bf,Bf_neg,W),!,
            out_W([Quan,Bf],W,Bf_out1),
            out_W([Quan_neg,Bf_neg],W,Bf_out2),
            renaming(List_s,List_new_s),Bf_out2=[C,Bf1],
            u0(Bf1,List_new_s,Bf1_out),put_quant([Bf1_out],Bfout3),
            Bfout3=[Qu_n,Bf_neg],update(flag1(no)),neg(Qu_n,Qu_neg),
            Bf_o3=[Qu_neg,Bf_neg],append(Bf_out1,Bf_o3,Bf_out).
solved_pair(nf,[Quan,Bf],[Quan_neg,Bf_neg],W,Bf_out):-
            !,out_W([Quan,Bf],W,Bf_out).


   /*********************************************************
equ_vars/1 comprueba que las partes derechas de las ecuaciones de una
              lista de formulas basicas son variables.
   ******************************************************/
```

```
equ_vars([]).
equ_vars([(X=Y)|R]):- var(Y),equ_vars(R).

     /*******************************************************
 distrib/2 introduce la negacion y aplica la propiedad distributiva a
         la disyuncion de conjunciones de formulas basicas.
      ******************************************************/

distrib(L_conj,Lc_out):- neg_quan(L_conj,L_neg_q),
                         distr_yo(L_neg_q,Lc_out1),
                         unif_pos(Lc_out1,Lc_out).

     /*******************************************************
   only_onevar/2 comprueba si solo queda una variable inicial en la
                         lista de ecuaciones.
      ******************************************************/

only_onevar(Y,[]):-!.
only_onevar(Y,[[Qu,[]]|Rest]):- !,only_onevar(Y,Rest).
only_onevar(Y,[[Qu,[]|Rest]|Rest_p]):-
            !,only_onevar(Y,[Rest|Rest_p]).
only_onevar(Y,[[Qu,[(X=Z)|Rest_eq]|Rest]|Rest_p]):-
            take_vbles(X,L),only_Y(Y,L),!,
            only_onevar(Y,[[Qu,Rest_eq|Rest]|Rest_p]).

only_Y(Y,[]):-!.
only_Y(Y,[X|R]):- Y==X,!,only_Y(Y,R).

     /*******************************************************
    neg_quan/2 niega las listas de cuantificadores existenciales.
      ******************************************************/

neg_quan([],[]).
neg_quan([[]|Rest],[[]|Rest_n]):- neg_quan(Rest,Rest_n).
neg_quan([[Quan,Bf|Restc]|Rest],[[C_neg,Bf|Restc_n]|Rest_n]):-
   !,neg_q(Quan,C_neg),neg_quan([Restc|Rest],[Restc_n|Rest_n]).

neg_q([e(X)],[~(e(X))]):-!.
neg_q([~(e(X))],[e(X)]):-!.
neg_q([],[~]):-!.
neg_q([~],[]):-!.
neg_q([e(X)|Rest],[~(e(X))|Restq]):- !,neg_q(Rest,Restq).
neg_q([~(e(X))|Rest],[e(X)|Restq]):- !,neg_q(Rest,Restq).
```

```
/*********************************************************
distr_yo/2 aplica la propiedad distributiva a la que es ahora
conjuncion de disyunciones de formulas basicas.
*********************************************************/


distr_yo([],[]):-!.
distr_yo([Conj|Rest],List):- Rest == nil,!,
              make_sollist(Conj,List).
distr_yo([Conj|Rest],Lt_out):- make_sollist(Conj,List),
              distr_yo(Rest,List_out1),
              mix(List,List_out1,List_out1,List_out2),
              d_out_rbf(List_out2,List_out3),
              d_out_conj(List_out3,List_out4),
              simpli_c(List_out4,Lt_out).

make_sollist([],[]):-!.
make_sollist([Quan,Bf|Rest],[[Quan,Bf]|Rest_out]):-
              !,make_sollist(Rest,Rest_out).

mix([],_,_,[]):-!.
mix([A|Restc],[],List,C):- !,mix(Restc,List,List,C).
mix([A|Restc],[B|Rest],List,[C|D]):-
              append(A,B,C),mix([A|Restc],Rest,List,D).


/*********************************************************
d_out_rbf/2 elimina las formulas basicas repetidas dentro de una
conjuncion.
*********************************************************/


d_out_rbf([],[]):-!.
d_out_rbf([[]|Rest],[[]|Rest_o]):-
         !,d_out_rbf(Rest,Rest_o).
d_out_rbf([[Quan,Bf|Restq]|Rest],[Restq_o|Rest_o]):-
         is_inbf(Bf,Restq),is_inqu(Quan,Restq),!,
         d_out_rbf([Restq|Rest],[Restq_o|Rest_o]).
d_out_rbf([[Quan,Bf|Restq]|Rest],[[Quan,Bf|Restq_o]|Rest_o]):-
         d_out_rbf([Restq|Rest],[Restq_o|Rest_o]).


/*********************************************************
d_out_conj/2 elimina las conjunciones de formulas basicas repetidas.
*********************************************************/
```

```
d_out_conj([],[]):-!.
d_out_conj([Conj|Rest],Rest_o):-
     is_conj_in(Conj,Rest),!,d_out_conj(Rest,Rest_o).
d_out_conj([Conj|Rest],[Conj|Rest_o]):-
     d_out_conj(Rest,Rest_o).

     /*******************************************************
  is_conj_in/2 analiza si una conjuncion se encuentra en una lista de
                          conjunciones.
      *******************************************************/

is_conj_in(_,[]):-!,fail.
is_conj_in(Conj,[Con_ju|Rest]):- eq_length(Conj,Con_ju),
                                 equal_co(Conj,Con_ju),!.
is_conj_in(Conj,[Con_ju|Rest]):- is_conj_in(Conj,Rest).

equal_co([],_):-!.
equal_co([Quan,Bf|Rest],Con_ju):- is_inqubf([Quan,Bf],Con_ju),
                                 equal_co(Rest,Con_ju).

is_inqubf(_,[]):-!,fail.
is_inqubf([Quan,Bf],[C,F|R]):- is_inqu(Quan,[C,F]),
                                 is_inbf(Bf,[C,F]),!.
is_inqubf([Quan,Bf],[C,F|R]):- !,is_inqubf([Quan,Bf],R).

is_inqu(Var,[]):- !,fail.
is_inqu(Var,[Vble,Bf|Rest]):- Var==Vble,!.
is_inqu(Var,[Vble,Bf|Rest]):- is_in(Var,Rest).

is_inbf(Var,[]):- !,fail.
is_inbf(Var,[Qu,Vble|Rest]):- Var==Vble,!.
is_inbf(Var,[Qu,Vble|Rest]):- is_in(Var,Rest).

     /*******************************************************
   simpli_c/2 simplifica logicamente la disyuncion de conjunciones.
      *******************************************************/

simpli_c(L_conj,L_c_sim):-
        !,mk_list_re(L_conj,L_conj,L_conj,List_rem),
        out_co(L_conj,List_rem,L_c_sim).

     /*******************************************************
   mk_list_re/4 construye una lista de las conjunciones a eliminar.
      *******************************************************/
```

```
mk_list_re([],_,_,[]):-!.
mk_list_re([Conj|Rest],[],L_conj,Rest_o):-
          !,mk_list_re(Rest,L_conj,L_conj,Rest_o).
mk_list_re([Conj|Rest],[Conj1|Rest1],L_conj,[Conj1|Rest_o]):-
          \+(eq_length(Conj,Conj1)),equal_co(Conj,Conj1),!,
          mk_list_re([Conj|Rest],Rest1,L_conj,Rest_o).
mk_list_re([Conj|Rest],[Conj1|Rest1],L_conj,Rest_o):-
          mk_list_re([Conj|Rest],Rest1,L_conj,Rest_o).

     /*********************************************************
  out_co/3 elimina las conjunciones de una lista que aparecen en la
                           otra lista.
     *******************************************************/

out_co([],_,[]):-!.
out_co([Conj|Rest],List_rem,[Conj|Rest_o]):-
   \+(is_in(Conj,List_rem)),!,out_co(Rest,List_rem,Rest_o).
out_co([Conj|Rest],List_rem,Rest_o):-
   out_co(Rest,List_rem,Rest_o).

     /*********************************************************
  unif_pos/2 unifica las posibles formulas basicas positivas en una
               sola en la diyuncion de conjunciones.
     *******************************************************/

unif_pos([],[]).
unif_pos([Conj|Rest],[Conj_unif|Rest_out]):-
    unif_conj(Conj,Conj_unif),Conj_unif \=nil,!,
    unif_pos(Rest,Rest_out).
unif_pos([Conj|Rest],Rest_out):- unif_pos(Rest,Rest_out).

unif_conj(Conj,New_conj):-
    agrup_pos(Conj,All_pos,All_neg,Qu_pos),
    solved(All_pos,One_pos),
    out_eq_aux(One_pos,Qu_pos,One_pos1),
    put_quant([One_pos1],One_pos_qu),
    ((One_pos_qu \= [nil,nil],
    append(One_pos_qu,All_neg,New_conj));!,New_conj=All_neg).
unif_conj(Conj,nil).

     /*********************************************************
 agrup_pos/2 separa las formulas basicas positivas de las negativas,
             devolviendo las primeras sin cuantificar.
     *******************************************************/
```

```
agrup_pos([],[],[],[]).
agrup_pos([Qu,Bf|Rest],Only_pos,Rest_neg,Qu_pos):-
          (Qu=[e(_)|R] ; Qu=[]),!,
          agrup_pos(Rest,Rest_pos,Rest_neg,C_p),
          append(Bf,Rest_pos,Only_pos),append(Qu,C_p,Qu_pos).
agrup_pos([Qu,Bf|Rest],Rest_pos,[Qu,Bf|Rest_neg],Qu_pos):-
          agrup_pos(Rest,Rest_pos,Rest_neg,Qu_pos).


     /*******************************************************
  out_eq_aux/2 elimina las ecuaciones en las que aparezcan solamente
                       variables auxiliares.
     *******************************************************/


out_eq_aux(One_pos,Qu_pos,One_pos_out):-
          make_l_v(Qu_pos,L_var),
          out_equations(One_pos,L_var,One_pos_out).

make_l_v([],[]).
make_l_v([e(X)|Rest],[X|R_out]):- make_l_v(Rest,R_out).
make_l_v([nil|Rest],R_out):- Rest\=nil,!,make_l_v(Rest,R_out).

out_equations([],_,[]).
out_equations([(X=Y)|Rest],L_vars,[(X=Y)|R_out]):-
          \+(is_in(X,L_vars)),!,out_equations(Rest,L_vars,R_out).
out_equations([(X=Y)|Rest],L_vars,R_out):-
          out_equations(Rest,L_vars,R_out).


     /*******************************************************
    enter_E/3 introduce el cuantificador existencial dentro de la
            conjuncion, aplicando el DCA si es necesario.
     *******************************************************/


enter_E([Quan,Bf|Rest],Signat,List_out):-
     (Quan=[~(e(X))|R];Quan=[~|Re]),!,make_taut(Bf,Bf_t),
     overlap1([Bf_t],[Quan,Bf|Rest],Signat,List_bf),
     put_quant(List_bf,List_quantif),
     mix_bf(List_quantif,[Quan,Bf|Rest],List_out).
enter_E([Quan,Bf|Rest],Signat,[Quan,Bf]):- Rest==nil,!.
enter_E([Quan,Bf|Rest],Signat,List_out):-
     overlap1([Bf],Rest,Signat,List_bf),
     put_quant(List_bf,List_quantif),
     mix_bf(List_quantif,Rest,List_out).
```

```
/*****************************************************
make_taut/2 crea una formula basica tautologica.
*****************************************************/

make_taut([],[]).
make_taut([(X=Y)|Rest],[(X=X1)|Rest_bf]):-
         make_taut(Rest,Rest_bf).

/*****************************************************
overlap1/4 se encarga de aplicar el DCA a las formulas basicas de la
disyuncion que unifiquen y no solapen con las formulas basicas
negativas.
*****************************************************/

overlap1(Disyu,List_neg,Signat,List_out):-
         overlap(Disyu,List_neg,Signat,List_neg,Var_ch),
         Var_ch \= nil,!,simp(Var_ch,[],Var_ch1),
         make_new_d(Disyu,Var_ch1,Var_ch1,New_disyu),
         overlap1(New_disyu,List_neg,Signat,List_out).
overlap1(Disyu,_,_,Disyu).

simp([],_,[]).
simp([[X|R]|R_v],Vars,[[X|R]|Var_ch]):-
         \+(is_in(X,Vars)),!,simp(R_v,[X|Vars],Var_ch).
simp([[X|R]|R_v],Vars,Var_ch):- simp(R_v,Vars,Var_ch).

/*****************************************************
overlap/5 analiza si se puede introducir el cuantificador
existencial dentro de la conjuncion y, en caso necesario, aplica el
DCA.
*****************************************************/

overlap([],_,_,_,[]).
overlap([[]|R_c],Eq,Sig,L_neg,R_d):- !,overlap(R_c,Eq,Sig,L_neg,R_d).
overlap([[(X=Y)|R]|R_c],[Qu,[]],Sig,L_neg,L):-
        !,overlap([R|R_c],L_neg,Sig,L_neg,L).
overlap([[(X=Y)|R]|R_c],[Qu,[]|R_neg],Sig,L_neg,LL_out):-
        !,overlap([[(X=Y)|R]|R_c],R_neg,Sig,L_neg,LL_out).
overlap([[(X=Y)|R]|R_c],[Qu,[(U=W)|R_n]|R_c_n],Sig,L_neg,LL_out):-
        X==U,\+(solved([(Y=W)],Out)),!,
        overlap([[(X=Y)|R]|R_c],[Qu,R_n|R_c_n],Sig,L_neg,LL_out).
overlap([[(X=Y)|R]|R_c],[Qu,[(U=W)|R_n]|R_c_n],Sig,L_neg,LL_out):-
```

```
        X==U,solvedfin([(Y=W)],Out),\+(eq_2_var(Out)),!,
        overlap([[(X=Y)|R]|R_c],[Qu,R_n|R_c_n],Sig,L_neg,LL_out).
overlap([[(X=Y)|R]|R_c],[Qu,[(U=W)|R_n]|R_c_n],Sig,L_neg,LL_out):-
        X==U,!,dca((X=Y),W,Sig,L_eq),
        overlap([[(X=Y)|R]|R_c],[Qu,R_n|R_c_n],Sig,L_neg,L_out),
        append(L_eq,L_out,LL_out).
overlap([[(X=Y)|R]|R_c],[Qu,[(Z=W)|R_n]|R_c_n],Sig,L_neg,LL_out):-
        !,overlap([[(X=Y)|R]|R_c],[Qu,R_n|R_c_n],Sig,L_neg,LL_out).

     /**********************************************************
                 dca/3 aplica el DCA a una ecuacion.
      *********************************************************/

dca((X=Y),W,Signat,[List_eq]):-
    var(Y),!,dca_terms(Signat,List_term),
    append([Y],List_term,List_eq).
dca((X=Y),W,Signat,List_eq):-
    functor(Y,F,N),N>0,!,solved([(Y=W)],Out),
    change((X=Y),Out,Signat,List_eq).

dca_terms([],[]).
dca_terms([X|Rest],[F|Rest_out]):-
    X=(F,0),!,dca_terms(Rest,Rest_out).
dca_terms([X|Rest],[Term|Rest_out]):-
    X=(F,N),!,make_l_arg(N,L_arg),
    Term=..[F|L_arg],dca_terms(Rest,Rest_out).

     /**********************************************************
make_l_arg/2 hace una lista de N (primer argumento) variables nuevas.
      *********************************************************/

make_l_arg(0,[]):-!.
make_l_arg(N,[N1|Rest]):- D is N-1,!,make_l_arg(D,Rest).

     /**********************************************************
   change/4 cambia una ecuacion por las ecuaciones que se obtienen
                   aplicando el algoritmo de DCA.
      *********************************************************/

change(X=Y),Solved_eq,Signat,List_out):-
       take_vbles(Y,List_var),
       var_only(List_var,Solved_eq,Solved_eq,List_ch),
       make_new_e((X=Y),List_ch,Signat,List_out).
```

```
/*********************************************************
 make_new_e/4 crea las nuevas ecuaciones surgidas al cambiar las
 variables de la parte derecha de una ecuacion por los terminos DCA
 correspondientes.
 ********************************************************/

make_new_e(_,[],_,[]).
make_new_e((X=Y),[Z|Rest],Signat,List):-
          dca_terms(Signat,L_terms),append([Z],L_terms,N_terms),
          make_new_e((X=Y),Rest,Signat,L_out),
          append([N_terms],L_out,List).

/*********************************************************
var_only/4 hace una lista con las variables que aparecen en una lista
 dada y que tambien aparecen en una lista de ecuaciones.
 ********************************************************/

var_only([],_,_,[]).
var_only([X|Rest],[],Solved_eq,Rest_out):-
         !,var_only(Rest,Solved_eq,Solved_eq,Rest_out).
var_only([X|Rest],[(Z=Y)|Rest_sol],Solved_eq,[X]):- (X==Z;X==Y),!.
var_only([X|Rest],[(Z=Y)|Rest_sol],Solved_eq,Rest_out):-
         !,var_only([X|Rest],Rest_sol,Solved_eq,Rest_out).

/*********************************************************
eq_2_var/1 comprueba que en una lista de ecuaciones al menos una esta
 formada por (variable = variable) o por (variable = termino).
 ********************************************************/

eq_2_var([]):- !,fail.
eq_2_var([(X=Y)|Rest]):- var(X),var(Y),!.
eq_2_var([(X=Y)|Rest]):- var(X),functor(Y,_,N),N>0,!.
eq_2_var([(X=Y)|Rest]):- !,eq_2_var(Rest).

/*********************************************************
 make_new_d/4 crea la nueva disyuncion de formulas basicas.
 ********************************************************/

make_new_d([],_,_,[]).
make_new_d([Bf|Rest_Bf],[],Var_ch,[Bf|List_out]):-
```

```
          !,make_new_d(Rest_Bf,Var_ch,Var_ch,List_out).
make_new_d([Bf|Rest_bf],[V_dca|Rest_dca],Var_ch,List_out):-
          do_new_e(Bf,V_dca,L_out_bf),[Bf]\=L_out_bf,!,
          append(L_out_bf,Rest_bf,New_l_bf),
          make_new_d(New_l_bf,Rest_dca,Var_ch,List_out).
make_new_d([Bf|Rest_Bf],[V_dca|Rest_dca],Var_ch,List_out):-
          make_new_d([Bf|Rest_Bf],Rest_dca,Var_ch,List_out).

do_new_e([],_,[]).
do_new_e([(X=Y)|Rest],[Z|Rest_dca],New_eq):-
          var(Y),Z==Y,!,new_eq(X,Rest_dca,New_eq1),
          do_new_e(Rest,[Z|Rest_dca],New_eq2),
          do_f_n(New_eq1,New_eq2,New_eq2,New_eq).
do_new_e([(X=Y)|Rest],[Z|Rest_dca],New_eq):-
          \+(var(Y)),functor(Y,F,N),N>0,take_vbles(Y,L),
          is_in(Z,L),!,make_ch((X=Y),[Z|Rest_dca],New_eq1),
          do_new_e(Rest,[Z|Rest_dca],New_eq2),
          do_f_n(New_eq1,New_eq2,New_eq2,New_eq).
do_new_e([(X=Y)|Rest],[Z|Rest_dca],New_eq):-
          do_new_e(Rest,[Z|Rest_dca],New_eq2),
          do_f_n([(X=Y)],New_eq2,New_eq2,New_eq).

do_f_n([],_,_,[]).
do_f_n(L,[],[],L_out):- !,make_struct([c,L],L_out).
do_f_n(L,[],_,L).
do_f_n([(X=Y)|Rest],[Bf|Rest_bf],T,[N_bf|L_out]):-
          Rest_bf == nil,Bf\=(Z=Q),!,
          append([(X=Y)],Bf,N_bf),do_f_n(Rest,[Bf],T,L_out).
do_f_n([(X=Y)|Rest],[Bf|Rest_bf],T,[N_bf|L_out]):-
          Rest_bf == nil,!,append([(X=Y)],[Bf],N_bf),
          do_f_n(Rest,[Bf],T,L_out).
do_f_n([(X=Y)|Rest],[Bf|Rest_bf],T,[N_bf|L_out]):-
          Rest=nil,Rest_bf\=nil,Bf\=(Z=Q),!,
          append([(X=Y)],Bf,N_bf),
          do_f_n([(X=Y)],Rest_bf,T,L_out).
do_f_n([(X=Y)|Rest],[Bf|Rest_bf],T,[N_bf|L_out]):-
          Rest=nil,Rest_bf\=nil,!,append([(X=Y)],[Bf],N_bf),
          do_f_n([(X=Y)],Rest_bf,T,L_out).
do_f_n([(X=Y)|Rest],[Bf|Rest_bf],T,[N_bf|L_out]):-
          Bf\=(Z=Q),!,append([(X=Y)],Bf,N_bf),
          do_f_n(Rest,Rest_bf,T,L_out).
do_f_n([(X=Y)|Rest],[Bf|Rest_bf],T,[N_bf|L_out]):-
          append([(X=Y)],[Bf],N_bf),
          do_f_n(Rest,Rest_bf,T,L_out).
```

```
make_ch((X=Y),[Z],[]).
make_ch((X=Y),[Z,Q|R_dca],[(X=T)|New_eq]):-
          change__(Y,Z,Q,T),make_ch((X=Y),[Z|R_dca],New_eq).

new_eq(X,[],[]).
new_eq(X,[Y|Rest_t],[(X=Y)|Rest_out]):-
          new_eq(X,Rest_t,Rest_out).

make_struct([C,[]],[]).
make_struct([C,[(X=Y)|Rest]],[[(X=Y)]|L_out]):-
          make_struct([C,Rest],L_out).


     /**********************************************************
put_quant/2 cuantifica existencialmente una lista de formulas basicas
                        sin cuantificar.
     *********************************************************/


put_quant([],[]):-!.
put_quant([[]|R_d],[[],[]|R_out]):- !,put_quant(R_d,R_out).
put_quant([[(X=Y)|R_q]|R_d],[[e(Y)|R_qu],[(X=Y)|R_q_out]|R_out]):-
          var(Y),put_quant([R_q|R_d],[R_qu,R_q_out|R_out]),
          take_vbles(R_qu,L_vars),\+(is_in(Y,L_vars)),!.
put_quant([[(X=Y)|R_q]|R_d],[R_qu,[(X=Y)|R_q_out]|R_out]):-
          var(Y),!,put_quant([R_q|R_d],[R_qu,R_q_out|R_out]).
put_quant([[(X=Y)|R_q]|R_d],[More_quant,[(X=Y)|R_q_out]|R_out]):-
          functor(Y,F,N),N>0,!,take_vbles(Y,List_var),
          put_quant([R_q|R_d],[R_qu,R_q_out|R_out]),
          take_vbles(R_qu,Qu_var),
          do_l_quant(List_var,Qu_var,List_quant),
          append(List_quant,R_qu,More_quant).
put_quant([[(X=Y)|R_q]|R_d],[R_qu,[(X=Y)|R_q_out]|R_out]):-
          !,put_quant([R_q|R_d],[R_qu,R_q_out|R_out]).
put_quant([(X=Y)|R],L_out):- !,put_quant([[(X=Y)|R]],L_out).

do_l_quant([],_,[]).
do_l_quant([X|Rest_var],Qu_var,[e(X)|R_out]):-
          \+(is_in(X,Rest_var)),\+(is_in(X,Qu_var)),!,
          do_l_quant(Rest_var,Qu_var,R_out).
do_l_quant([X|Rest_var],Qu_var,R_out):-
          do_l_quant(Rest_var,Qu_var,R_out).


     /**********************************************************
```

```
mix_bf/3 forma una lista combinando las formulas basicas positivas de
    una lista (primer argumento) con la lista de formulas basicas
                    negativas (segundo argumento).
      *******************************************************/


mix_bf([],_,[]).
mix_bf([Qu,Bf|Rest],Part_neg,[Conj|Rest_for]):-
       !,append([Qu,Bf],Part_neg,Conj),
       mix_bf(Rest,Part_neg,Rest_for).


     /*******************************************************
  out_equal_bf/2 elimina las listas de ecuaciones no negadas iguales
                   que se producen en el algoritmo.
      *******************************************************/


out_equal_bf([],[]).
out_equal_bf([[~],Eq|Rest_qu_eq],[[~],Eq|List_aplem]):-
            !,out_equal_bf(Rest_qu_eq,List_aplem).
out_equal_bf([[~(e(X))|Rest],Eq|Rest_qu_eq],
                  [[~(e(X))|Rest],Eq|List_aplem]):-
            !,out_equal_bf(Rest_qu_eq,List_aplem).
out_equal_bf([[(e(X))|Rest],Eq|Rest_qu_eq],List_aplem):-
            !,out_equal_bf(Rest_qu_eq,List_aplem).
out_equal_bf([nil,Eq|Rest_qu_eq],List_aplem):-
            !,out_equal_bf(Rest_qu_eq,List_aplem).


     /*******************************************************
   is_solv/3 analiza si la formula construida con los parametros es
                              resoluble.
      *******************************************************/


is_solv(List_eq,List_eq_neg,List_solv):-
        do_0(List_eq,List_eq_neg,List_eq_neg,List_eq_0),
        solved(List_eq_0,List_solv).

do_0([],_,_,[]):-!.
do_0([(A=B)|Rest],[(D=C)|Rest_neg],List_eq_neg,[(B=C)|List_eq_0]):-
     A==D,!,do_0(Rest,List_eq_neg,List_eq_neg,List_eq_0).
do_0(List_eq,[(D=C)|Rest_neg],List_eq_neg,List_eq_0):-
     do_0(List_eq,Rest_neg,List_eq_neg,List_eq_0).
```

```
/********************************************************
 solvedfin/2 aplica a un sistema de ecuaciones un algoritmo
      simplificado de transformacion a forma resuelta.
********************************************************/

solvedfin(Temp_Constr,F_Constr):- reduc1(Temp_Constr,R_Constr),
                                  reduc2(R_Constr,P_Constr),
                                  rem_rep(P_Constr,F_Constr).

/********************************************************
out_W/4 elimina una ecuacion y un cuantificador de una formula
                           basica.
*******************************************************/

out_W([Quan,Bf],W,[Quan_out,Bf_out]):-
      out_bf_W(Bf,W,Bf_out),out_quan_W(Quan,Bf_out,Quan_out1),
      out_equal_e(Quan_out1,Quan_out2),
      put_ifnil(Quan,Quan_out2,Quan_out).

out_bf_W([(A=B)|Rest],W,Rest):- A==W,!.
out_bf_W([(A=B)|Rest],W,[(A=B)|Bf_out]):- out_bf_W(Rest,W,Bf_out).

out_quan_W(_,[],[]).
out_quan_W(nil,_,nil).
out_quan_W([~],_,[~]).
out_quan_W([e(X)|Rest],[(A=B)|Rest_bf],[e(B)|Quan_out]):-
      var(B),!,out_quan_W([e(X)|Rest],Rest_bf,Quan_out).
out_quan_W([~(e(X))|Rest],[(A=B)|Rest_bf],[~(e(B))|Quan_out]):-
      var(B),!,out_quan_W([~(e(X))|Rest],Rest_bf,Quan_out).
out_quan_W([e(X)|Rest],[(A=B)|Rest_bf],Quan_out):-
      take_vbles(B,List_vars),!,do_list_e(List_vars,+,List_e),
      out_quan_W([e(X)|Rest],Rest_bf,Quan_out1),
      append(List_e,Quan_out1,Quan_out).
out_quan_W([~(e(X))|Rest],[(A=B)|Rest_bf],Quan_out):-
      take_vbles(B,List_vars),!,do_list_e(List_vars,~,List_e),
      out_quan_W([~(e(X))|Rest],Rest_bf,Quan_out1),
      append(List_e,Quan_out1,Quan_out).
out_quan_W(C,[(A=B)|Rest_bf],Quan_out):-
      atom(B),!,out_quan_W(C,Rest_bf,Quan_out).

/********************************************************
do_list_e/3 cuantifica existencialmente los elementos de una lista
                   formada por variables.
*******************************************************/
```

```
do_list_e([],_,[]).
do_list_e([X|Rest],+,[e(X)|List_e]):-
        \+(is_in(X,Rest)),!,do_list_e(Rest,+,List_e).
do_list_e([X|Rest],~,[~(e(X))|List_e]):-
        \+(is_in(X,Rest)),!,do_list_e(Rest,~,List_e).
do_list_e([X|Rest],S,List_e):- do_list_e(Rest,S,List_e).


     /*******************************************************
 out_equal_e/3 elimina los cuantificadores existenciales repetidos de
                            una lista.
     *******************************************************/


out_equal_e([],[]).
out_equal_e([~],[~]).
out_equal_e([e(X)|Rest],[e(X)|List_e]):-
          out_equal_e(Rest,List_e),\+(is_e_in(e(X),List_e)),!.
out_equal_e([e(X)|Rest],List_e):-
          out_equal_e(Rest,List_e).
out_equal_e([~(e(X))|Rest],[~(e(X))|List_e]):-
          out_equal_e(Rest,List_e),\+(is_e_in(~(e(X)),List_e)),!.
out_equal_e([~(e(X))|Rest],List_e):-
          out_equal_e(Rest,List_e).

is_e_in([],_):- fail.
is_e_in(e(X),[e(Y)|List_e]):- X==Y,!.
is_e_in(e(X),[e(Y)|List_e]):- is_e_in(e(X),List_e).
is_e_in(~(e(X)),[~(e(Y))|List_e]):- X==Y,!.
is_e_in(~(e(X)),[~(e(Y))|List_e]):- is_e_in(~(e(X)),List_e).


     /*******************************************************
 put_ifnil/3 incluye una negacion si los cuantificadores negados han
                       sido todos eliminados.
     *******************************************************/


put_ifnil([~(e(X))|Rest],nil,[~]):-!.
put_ifnil(_,E,E).


     /*******************************************************
 is_inconst/3 analiza si una variable v es restringida con respecto a
                        u U w en una formula.
     *******************************************************/
```

```
is_inconst(List_eq,List_eq_neg,Var_Y):-
          get_vars(List_eq,Var_Y,_,Eq_Y,[],U,V),
          get_vars(List_eq_neg,Var_Y,_,Eq_Y_neg,[],W,Z),
          see_inconst(Eq_Y,Eq_Y_neg,U,V,W,Z).

get_vars([(A=B)|Rest],Var_Y,Eq_Y,Eq_Y_out,
                 Vars_out_1,Vars_out,Vars_in):-
          atom(B),A\==Var_Y,!,
          get_vars(Rest,Var_Y,Eq_Y,Eq_Y_out,
                 Vars_out_2,Vars_out,Vars_in).
get_vars([(A=B)|Rest],Var_Y,Eq_Y,Eq_Y_out,
                 Vars_out_1,Vars_out,Vars_in):-
          A\==Var_Y,!,take_vbles(B,List_var),
          append(Vars_out_1,List_var,Vars_out_2),
          get_vars(Rest,Var_Y,Eq_Y,Eq_Y_out,
                 Vars_out_2,Vars_out,Vars_in).
get_vars([(A=B)|Rest],Var_Y,Eq_Y,Eq_Y_out,
                 Vars_out_1,Vars_out,Vars_in):-
          A==Var_Y,!,get_vars(Rest,Var_Y,(A=B),
                 Eq_Y_out,Vars_out_1,Vars_out,Vars_in).
get_vars([],Var_Y,(A=B),(A=B),Vars_out_1,Vars_out_1,Vars_in):-
          take_vbles(B,List_vars),\+(atom(B)),!,
          not_are_in(List_vars,Vars_out_1,Vars_in).
get_vars([],Var_Y,(A=B),(A=B),Vars_out_1,nil,Vars_in):-
          not_are_in(nil,Vars_out_1,Vars_in).

see_inconst((A=B),(A=C),U,V,W,Z):-
          solved([(B=C)],List_eq),
          inconst(List_eq,U,V,W,Z).
inconst([],_,_,_,_).
inconst([(A=B)|Rest],U,V,W,Z):-
          take_vbles(A,A_vars),take_vbles(B,B_vars),
          (V==nil,!;((is_v_in(B_vars,V,V),
          \+(is_v_in(A_vars,W,W)),\+(is_v_in(A_vars,U,U)),!);
          (is_v_in(A_vars,V,V),\+(is_v_in(B_vars,W,W)),
          \+(is_v_in(B_vars,U,U))))),inconst(Rest,U,V,W,Z).


    /**********************************************************
   renaming/2 renombra las variables de los terminos de las partes
          derecha de las ecuaciones de una formula basica.
     *********************************************************/


renaming(Eq,New_eq):- make_li_v(Eq,[],List_vars),
        do_rename(Eq,List_vars,New_eq).
```

```
make_li_v([],L,L).
make_li_v([(X=Y)|Rest],List_v,List_vars):-
        take_vbles(Y,L_v),not_are_in(L_v,List_v,L_p_v),
        append(List_v,L_p_v,New_l_v),
        make_li_v(Rest,New_l_v,List_vars).

do_rename(L,[],L).
do_rename(L_eq,[Z|R_vars],N_equa):-
        do_ch(L_eq,Z,Z1,N_l_eq),do_rename(N_l_eq,R_vars,N_equa).

do_ch([],_,_,[]).
do_ch([(X=Y)|R],Z,Q,[(X=Ny)|R_eq]):-
        change__(Y,Z,Q,Ny),do_ch(R,Z,Q,R_eq).


     /*********************************************************
    u0/3 construye la nueva formula basica despues de eliminar el
               cuantificador existencial introducido.
      ********************************************************/


u0(F,[],F).
u0(Bf_out2,[(X=Y)|Rest],Bf_out3):-
           \+(var(Y)),!,put_Y((X=Y),Bf_out2,Bf_out),
           u0(Bf_out,Rest,Bf_out3).
u0(Bf_out2,[(X=Y)|Rest],Bf_out3):-
           u0(Bf_out2,Rest,Bf_out3).

put_Y(_,[],[]).
put_Y((X=Y),[(Z=Q)|Rest],[(Z=New_Q)|Rest_new]):-
           take_vbles(Q,List_vars),is_in(X,List_vars),!,
           change__(Q,X,Y,New_Q),put_Y((X=Y),Rest,Rest_new).
put_Y((X=Y),[(Z=Q)|Rest],[(Z=Q)|Rest_new]):-
           put_Y((X=Y),Rest,Rest_new).


     /*********************************************************
   change__/4 cambia una variable por otra dada en un termino dado.
      ********************************************************/


change__(X1,X2,T,T):- var(X1),X1==X2,!.
change__(X1,X2,T,X1):- var(X1),X1 \== X2,!.
change__(X1,_,_,X1):- functor(X1,_,0),!.
change__(X1,X2,T,TT):-
```

```
        X1=..[F|Args],change_args(Args,X2,T,Nargs),
        TT=..[F|Nargs],!.
change_args([],_,_,[]).
change_args([Arg|Rest_args],X2,T,[New_arg|New_rest_args]):-
        change_(Arg,X2,T,New_arg),
        change_args(Rest_args,X2,T,New_rest_args).


    /*******************************************************
  eval/1 evalua una disyuncion de conjunciones de valores booleanos.
     *******************************************************/


eval([Conj|RestC]):- eval_co(Conj),!;eval(RestC).

eval_co([]).
eval_co([[~(e(false))],false|Rest]):- !,eval_co(Rest).
eval_co([[(e(false))],false|Rest]):- !,fail.
eval_co([[~(e(true))],true|Rest]):- !,fail.
eval_co([[(e(true))],true|Rest]):- !,eval_co(Rest).
eval_co([[e(X)|_]|_]):-!.
eval_co([[~(e(X))|_]|_]):-!.
eval_co([[]|_]):-!.
eval_co([[~],nil]):-!,fail.
eval_co([[~]|_]):-!.
eval_co([true|Rest]):- !,eval_co(Rest).
```

```
/*****************************************************
         ***************************************
                       LIBRARY.PRO
            (Biblioteca de Clausulas Generales)
         ***************************************
 *****************************************************/

  /**************************************************************
                        member/2 - append/3
   *************************************************************/

member(A,[A|R]).
member(A,[_|R]):- member(A,R).

append([],L,L).
append([M|L1],L2,[M|L3]):- append(L1,L2,L3).

  /**************************************************************
    is_in/2 comprueba si un cierto elemento pertenece a una lista.
   *************************************************************/

is_in(_,[]):- !,fail.
is_in(El,[El_|R]):- El==El_,!.
is_in(El,[_|R]):- is_in(El,R).

  /**************************************************************
   is_eq_in/2 comprueba si una cierta ecuacion esta en una lista de
                            ecuaciones.
   *************************************************************/

is_eq_in(A=B,[A_=B|R]):- A==A_,!.
is_eq_in(E,[_|R]):- is_eq_in(E,R),!.

     /*******************************************************
 is_some/2 determina si algun elemento de la primera lista ocurre en
                            la segunda.
     *******************************************************/

is_some([Var|Rest], List):- is_in(Var,List);is_some(Rest,List).

  /**************************************************************
 are_in/3 construye una lista (tercer argumento) con los elementos de
              la primera lista que estan en la segunda.
   *************************************************************/
```

```
are_in([],_,[]).
are_in([V|R],S,RR_):- (is_in(V,S),RR_=[V|RR];RR_=RR),are_in(R,S,RR),!.

  /**************************************************************
 not_are_in/3 construye una lista (tercer argumento) con los elementos
          de la primera lista que no estan en la segunda.
   *************************************************************/

not_are_in(S,[],S):-!.
not_are_in([],L,L).
not_are_in([X|Rx],Lvbles,R):- (is_in(X,Lvbles),R=RR;R=[X|RR]),
                              not_are_in(Rx,Lvbles,RR),!.

  /**************************************************************
     take_vbles/2 construye la lista de variables de un termino.
   *************************************************************/

take_vbles(T,L_v):- take_vbles(T,[],L_v).
take_vbles(T,L,R):- var(T),(\+(is_in(T,L)),R=[T|L];R=L),!.
take_vbles(T,L,L):- functor(T,F,0),!.
take_vbles(T,L,List):- T=..[F|Args_F],take_vbles_args(Args_F,L,List).
take_vbles_args([],L,L).
take_vbles_args([Arg_i|Args],L,List):-
    take_vbles(Arg_i,L,List_Arg_i),take_vbles_args(Args,List_Arg_i,List).

     /*********************************************************
      is_var_in/2 comprueba si una variable esta en una lista de
                            ecuaciones.
      ********************************************************/

is_var_in(X,[(Z=Y)|Rest]):- X==Z,!.
is_var_in(X,[(Z=Y)|Rest]):- !,is_var_in(X,Rest).

     /*********************************************************
 is_v_in/3 comprueba si alguna de las variables de una lista pertenece
                     tambien a otra lista dada.
      ********************************************************/

is_v_in([],_,_):- !,fail.
is_v_in([X|Rest],[],A):- !,is_v_in(Rest,A,A).
is_v_in([X|Rest],[Y|R],_):- X==Y,!.
is_v_in([X|Rest],[Y|R],A):- !,is_v_in([X|Rest],R,A).
```

```
    /*******************************************************
 eq_length/2 comprueba si dos conjunciones tienen la misma longitud.
    *******************************************************/


eq_length([],[]):-!.
eq_length([Quan,Bf|Rest],[Qu1,Bf1|Rest1]):- !,eq_length(Rest,Rest1).


  /************************************************************
        t_l_to_c/2 transforma una lista en una conjuncion.
  ************************************************************/


t_l_to_c([],true=true):-!.
t_l_to_c(S1,S1_):- t_l_to_c(S1,S1_),!.
t_l_to_c([A],A):-!.
t_l_to_c([A|R],(A,RR)):- t_l_to_c(R,RR),!.

    /*******************************************************
         c_to_l_m/2 transforma una conjuncion en una lista.
    *******************************************************/


c_to_l_m((A,B),L):- c_to_l_m(A,L1),c_to_l_m(B,L2),
  (L1=[X1=Y1],X1==Y1,L2=[X2=Y2],X2==Y2,L=[true=true];
   L1=[X1=Y1],X1==Y1,L=L2;L2=[X2=Y2],X2==Y2,L=L1;append(L1,L2,L)),!.
c_to_l_m(A,[A]).


  /************************************************************
                   inverse/2 invierte una lista.
  ************************************************************/


inverse([],[]).
inverse([A|R],Rr):- inverse(R,R_),append(R_,[A],Rr).


  /************************************************************
       sublist/2 comprueba si una lista es sublista de otra.
  ************************************************************/


sublist([],_).
sublist(S,[_|T]):- sublist(S,T).
sublist([X|S],[X|T]):- sublist(S,T).
```

```
/************************************************************
      number/2 cuenta el numero de elementos de una lista.
************************************************************/

number([],0).
number([_|R],C):- number(R,CC), C is CC+1,!.

/**************************************************************
put_number/3 numera los elementos de una lista de forma consecutiva.
    Sus argumentos son el numero, la lista y la lista numerada.
**************************************************************/

put_number(_,[],[]).
put_number(N,[A|R],[(N,A)|RR]):- M is N+1,put_number(M,R,RR),!.

/**************************************************************
rem_number/2 a partir de una lista de pares construye otra lista que
  contiene como elementos las segundas componentes de estos pares.
**************************************************************/

rem_number([(_,A)|R],[A|RR]):- rem_number(R,RR),!.
rem_number([[]|R],[[]|RR]):- rem_number(R,RR),!.
rem_number([],[]).
rem_number([A|R],[A_|RR]):- rem_number(A,A_),rem_number(R,RR),!.

/************************************************************
cappend/3 concatena dos conjunciones de ecuaciones eliminando la
                 ecuacion trivial true=true.
************************************************************/

cappend(E,C,C):- E==(true=true),!.
cappend(C,E,C):- E==(true=true),!.
cappend((C,RC),H,R):- (is_in_c(C,RC),R=RC_; R=(C,RC_)),
                      cappend(RC,H,RC_),!.
cappend(C,H,(C,H)).
is_in_c(C,(CC,RC)):- C==CC,!.
is_in_c(C,CC):- C==CC,!.

/************************************************************
cappend_g/3 concatena dos conjunciones de ecuaciones eliminando la
  ecuacion trivial true=true y añadiendo al final una variable.
************************************************************/
```

```
cappend_g(E,C,C):- E==(true=true),!.
cappend_g(C,E,C):- E==(true=true),!.
cappend_g(Var,H,(Var,H)):- var(Var),!.
cappend_g((C,RC),H,R):- (is_in_c(C,RC),R=RC_;R=(C,RC_)),
                        cappend_g(RC,H,RC_),!.
cappend_g(C,H,(C,H)).

  /************************************************************
                 append_m/3 concatena dos listas.
   ************************************************************/

append_m(L,[],[]).
append_m(L,[LL|RL],[LR|RR_]):- append(L,LL,LR),append_m(L,RL,RR_).

  /************************************************************
   append_q/3 concatena dos listas situando la primera detras de la
                              segunda.
   ************************************************************/

append_q([],L,L).
append_q([X|RX],L,LR):- (\+ is_in(X,L),LR=[X|R];LR=R),append_q(RX,L,R).


    /*********************************************************
  append_exist/3 concatena dos listas cuantificando existencialmente
                          sus elementos.
     *********************************************************/

append_exist([],L,L).
append_exist([V|R],L,[e(V)|RR]):- append_exist(R,L,RR).

  /************************************************************
      rem_rep/2 elimina los elementos repetidos de una lista.
   ************************************************************/

rem_rep([],[]).
rem_rep([A|R],[A|R_]):- rem_e(A,R,RR),rem_rep(RR,R_).
rem_e(_,[],[]):-!.
rem_e(E,[E_|R],R_):- E==E_,rem_e(E,R,R_),!.
rem_e(E,[E_|R],[E_|R_]):- v(E,R,R_).
```

```
/*************************************************************
drop_list/3 elimina una cierto elemento de una lista.
*************************************************************/

drop_list(_,[],[]).
drop_list(X,[V|R],R):- X==V,!.
drop_list(X,[V|R],[V|RR]):- drop_list(X,R,RR),!.

/*************************************************************
drop_conj/3 elimina una cierto elemento de una conjuncion.
*************************************************************/

drop_conj(V,(A,B),B):- V==A,!.
drop_conj(V,(A,B),(A,C)):- drop_conj(V,B,C),!.

/***************************************************************
subsets_of/2 calcula los subconjuntos de un conjunto.
***************************************************************/

subsets_of(Subsets,Set_):- put_number(1,Set_,Set),
       bagof(Subset,sublist(Subset,Set),Subbags),
       rem_rep(Subbags,Subsets_),rem_number(Subsets_,Subsets),!.

/***************************************************************
are_irreducible/2 tiene exito si ambos argumentos son irreducibles.
***************************************************************/

are_irreducible(F,G):- irre_program(I),are_irr(F,no,G,no,I),!.
are_irr(_,si,_,si,_):- !.
are_irr(F,V,G,W,[S|R]):- (F=S,V1=si;V1=V),(G=S,V2=si;V2=W),!,
                         are_irr(F,V1,G,V2,R).

/*******************************************************
take_last/2 selecciona el ultimo cuantificador de la lista de
cuantificadores.
*******************************************************/

take_last([A|Rest],A):- Rest=[].
take_last([A|Rest],Last_Quan):- take_last(Rest,Last_Quan).
```