Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Resolución con orden y selección para la lógica $\mathcal{H}(@)$

Daniel Gorín
dgorin@dc.uba.ar

Director
Dr. Carlos Areces
INRIA Lorraine

Tesis para acceder al grado de
Licenciado en Ciencias de la Computación

2 de marzo de 2004

Resumen

La lógica $\mathcal{H}(@)$ es la que se obtiene al agregar nominales y el operador de satisfacción @ a la lógica modal básica. Ambas lógicas son decidibles para el problema de la validez de una fórmula, y su complejidad es PSPACE-complete. Tradicionalmente, los demostradores de teoremas para la lógica modal básica estuvieron basados en algoritmos de tableau, con los cuales se obtuvieron muy buenos resultados. Sin embargo, la lógica $\mathcal{H}(@)$ introduce una noción de igualdad que degrada la performance de este algoritmo; por esta razón resulta interesante investigar qué sucede con otras familias de algoritmos. En [Areces *et al.*, 2001] se propone un calculo basado en resolución para esta lógica. Este cálculo, si bien consistente y completo, carece de las estrategias de orden y selección que son estándar en resolución para lógica de primer orden. Es necesario contar con un mecanismo como este, que regule la generación de nuevas cláusulas, si se desea implementar en forma realista un demostrador basado en este cálculo.

En este trabajo incorporamos orden y selección al cálculo propuesto en [Areces *et al.*, 2001], y demostramos que este agregado preserva completitud refutacional. Además, proponemos un manejo más cuidadoso en la generación de nuevos nominales y en el manejo de la paramodulación; con esto podemos garantizar que el cálculo genera un número finito de cláusulas y, por lo tanto, que constituye un método de decisión para el problema de la validez de una fórmula de $\mathcal{H}(@)$. Finalmente, incorporamos estos resultados a HyLoRes, una implementación experimental del cálculo de [Areces *et al.*, 2001], y hacemos una breve comparación de la performance de ambas versiones.

Agradecimientos

Quiero agradecer especialmente a Carlos por toda la paciencia y dedicación que necesitó para guiarme hasta acá salvando, literalmente, una gran distancia.

También al jurado, por tomarse el tiempo de leer y corregir este trabajo.

A mi familia porque siempre me dio ánimo pero sin presionar nunca.

A mis amigos, que estuvieron cerca de una forma u otra todos estos años; a cada uno tengo que agradecerle por haberme acompañado, divertido, ayudado, distraído y, también, cuando lo creyeron necesario, presionado y hasta amenazado!

Finalmente, un agradecimiento especial para Ale que vio a esta tesis nacer y crecer; y otro para Laura, que se bancó estoicamente el final, donde uno siempre corre.

Índice General

1 Las lógicas modales 1
- 1.1 Los comienzos . . . 1
- 1.2 La revolución semántica . . . 3
- 1.3 Las lógicas modales, hoy . . . 4
- 1.4 Las lógicas híbridas . . . 6
- 1.5 Estructura de esta tesis . . . 9

2 Métodos de prueba 10
- 2.1 Decidibilidad del problema de la validez . . . 10
- 2.2 Tableaux . . . 11
  - 2.2.1 Tableau modal tradicional o *implícito* . . . 12
  - 2.2.2 Tableau etiquetado . . . 12
  - 2.2.3 Tableaux y lógica híbrida . . . 14
  - 2.2.4 Implementaciones del método de tableau . . . 14
- 2.3 Traducción a otros formalismos . . . 15
  - 2.3.1 Lógica de primer orden . . . 15
  - 2.3.2 Autómatas . . . 15
- 2.4 Resolución . . . 16
  - 2.4.1 Resolución para lógica de primer orden . . . 16
  - 2.4.2 Resolución para la lógica modal . . . 17
  - 2.4.3 El cálculo de resolución híbrido . . . 17
  - 2.4.4 Resolución con orden y selección . . . 19

3 Resolución híbrida con orden y selección 21
- 3.1 Relaciones de orden entre fórmulas . . . 21
- 3.2 Un orden *adecuado* para $\mathcal{H}(@)$ . . . 23
- 3.3 Un Teorema de Herbrand $\mathcal{H}(@)$ . . . 24
- 3.4 Resolución para $\mathcal{H}^{NNF}(@)$ con orden y selección . . . 26
  - 3.4.1 Definición del cálculo . . . 26
  - 3.4.2 Completitud refutacional . . . 28

4 Terminación del cálculo 35
- 4.1 Cómo generar infinitas fórmulas . . . 36
- 4.2 La regla ($\langle R \rangle$) . . . 37
- 4.3 La regla (PARAM) . . . 39
  - 4.3.1 Consistencia . . . 40
  - 4.3.2 Completitud refutacional . . . 42
  - 4.3.3 Un distribuidor de nominales concreto . . . 43
- 4.4 Terminación . . . 44

5 HyLoRes 2.0: Poniendo a prueba las ideas 48
5.1 HyLoRes 1.0 . . . 48
5.1.1 Algoritmo de la cláusula dada . . . 48
5.1.2 Eliminación de cláusulas redundantes . . . 50
5.1.3 La regla ($\langle R \rangle$) . . . 51
5.1.4 Paramodulación . . . 51
5.2 HyLoRes 2.0 . . . 51
5.2.1 Procesamiento de la cláusula dada . . . 52
5.2.2 Orden entre fórmulas . . . 53
5.2.3 Función de selección . . . 54
5.2.4 Criterios adicionales de subsunción . . . 54
5.2.5 Representación interna del conjunto de cláusulas . . . 54
5.2.6 La regla ($\langle R \rangle'$) . . . 55
5.2.7 La regla (RES-BOX) . . . 55
5.3 Evaluación empírica . . . 55
5.3.1 Conjuntos de problemas modales e híbridos . . . 56
5.3.2 Tests realizados y sus resultados . . . 58

A Descripción técnica de HyLoRes 2.0 67
A.1 Fórmulas . . . 67
A.2 Cláusulas y funciones de selección . . . 68
A.3 Conjuntos de cláusulas . . . 69
A.4 Motor monádico y ciclo principal . . . 71
A.5 Interfaz con el usuario . . . 71
A.6 Misceláneas . . . 72

Capítulo 1

Las lógicas modales

1.1 Los comienzos

Generalmente se ubica el origen de las lógicas modales como disciplina matemática en 1918, cuando C. I. Lewis publica su *Survey of Symbolic Logic* [Lewis, 1918]. Si bien podemos rastrear trabajos relacionados con lógicas que podríamos considerar *modales* desde Aristóteles hasta bien entrado el siglo XIX, no es fácil relacionarlos con la tradición lógica moderna.

Lewis analiza la diferencia que hay entre implicaciones de la forma $A \rightarrow B$ donde el hecho de que B no pueda ser falso cuando A es verdadero es *contingente* y aquellas donde este hecho es *necesario*. Alguien puede decir "si llueve, entonces Juan no va a trabajar" y esta afirmación puede ser *verdadera* en el contexto de enunciación (Juan puede estar engripado, o vivir en una zona inundable o vender bronceador en la playa), pero claramente no se trata de una *verdad universal* (dejará de ser verdadero si Juan se cura, se muda o cambia de trabajo). La relación que hay entre "que llueva" y "que Juan no vaya a trabajar" es contingente. Por otro lado, en la afirmación "si llueve, Juan no va a trabajar; pero como Juan sí va a trabajar, entonces no llueve" hay una relación lógica muy fuerte (*logical entailment*) entre el antecedente (que Juan no va a trabajar si llueve, pero que Juan sí va a trabajar) y el consecuente (que no llueve); de forma que en cualquier contexto en que el antecedente se cumpla, el consecuente necesariamente también debe valer.

En sus trabajos, Lewis extiende la lógica proposicional agregando el *operador modal* $\Diamond$, que se interpreta como "es posible que" y con él define $\mapsto$, la *implicación estricta*: $\varphi \mapsto \psi \equiv \neg\Diamond(\varphi \wedge \neg\psi)$; esta última fórmula se debe leer como "no es posible (en ningún contexto imaginable) que simultáneamente φ sea verdadero y $\neg\psi$ no lo sea". Usando este lenguaje podemos decir "si llueve, Juan no va a trabajar; pero como Juan sí va a trabajar, entonces no llueve" utilizando $\mapsto$, $((\mathsf{llueve} \rightarrow \neg\mathsf{trabaja}_{\mathsf{juan}}) \wedge \mathsf{trabaja}_{\mathsf{juan}}) \mapsto \neg\mathsf{llueve}$, y estaremos afirmando que éste es un hecho que no depende de circunstancias particulares.

$\Diamond$ es un operador modal ya que asigna un *modo* a un valor de verdad. Si $\Diamond$ toma el modo de *posibilidad*, lo usamos para diferenciar lo que *es verdadero* (o falso) de lo que *podría ser verdadero* (o falso). Tomemos dos proposiciones: nublado y llueve, y veamos ejemplos de lo que podemos decir al asociar este modo con dicho operador:

Fórmula	Significado
nublado	Está nublado en este momento
llueve	Está lloviendo en este momento
$\Diamond$nublado	Podría estar nublado en este momento
$\neg\Diamond$llueve	No podría llover en este momento
$\neg$nublado $\wedge$ llueve	En este momento llueve aunque no está nublado
$\neg\Diamond(\neg$nublado $\wedge$ llueve)	No podría suceder que ahora esté lloviendo aunque no esté nublado

En este ejemplo vemos dos casos de la forma "no es posible que suceda X". Intuitivamente, esto

es lo mismo que decir "es necesario que no suceda X". La *necesidad* es una modalidad que se suele notar con el operador $\Box$. Como vimos, existe una relación muy fuerte entre *posibilidad* y *necesidad*: $\neg\Diamond\neg\varphi \equiv \Box\varphi$. Cuando dos operadores modales cumplen con esta propiedad, se dice que uno es el *dual* del otro. Tradicionalmente, se asume que si $\Diamond$ es una modalidad cualquiera, $\Box$ es su dual (y viceversa).

Interpretar informalmente una fórmula modal como hicimos en el ejemplo anterior no es complicado. De hecho, en ese ejemplo podríamos haberle asignado a $\Diamond$ un modo cualquiera, por ejemplo *deseo*, e interpretado $\Diamond$nublado como "desearía que estuviera nublado". Hasta aquí, lo que tenemos no es más que una forma de escribir simbólicamente algunos enunciados. Para *capturar* efectivamente una noción imprecisa como "la posibilidad", Lewis analizó qué inferencias son válidas al utilizar este modo. Por ejemplo, inferir que $\Diamond$llueve dado que llueve es correcto si $\Diamond$ representa *posibilidad* (no es admisible aceptar como válido que "llueve pero no es posible que llueva"), pero claramente no lo es si $\Diamond$ representa *deseo*. En sus trabajos, Lewis presenta diversos sistemas axiomáticos que intentan generar como teoremas todas las fórmulas que son *verdaderas* en una lógica dada y sólo dichas fórmulas (la primera característica hace a un sistema axiomático *completo*, la segunda lo hace *consistente*).

A partir de los trabajos de Lewis, surgió el interés de buscar nuevas extensiones *modales* de la lógica proposicional. Se investigaron, de esta forma, las axiomatizaciones de conceptos tales como *obligación*, *creencia*, *conocimiento*, etc. Lo que, en general, estos primeros lógicos modales hicieron fue capturar de una manera puramente sintáctica conceptos que hasta ese momento pertenecían al dominio de la intuición.

Un trabajo que merece un comentario aparte es el que realizó Arthur Prior [Prior, 1957; Prior, 1967] a principios de la década de 1950 con una familia especial de lógicas modales: las llamadas *lógicas temporales* (*temporal logics* o *tense logics*). Prior intentaba representar en una lógica la forma en la cual tratamos las relaciones temporales en los lenguajes naturales (de ahí su nombre original de *tense logics*); para ello introduce la ubicación temporal como modo. En la lógica temporal básica, Prior utiliza la modalidad F para referirse a algún instante indeterminado en el futuro, y la modalidad P para hacerlo sobre algún instante del pasado. Los operadores duales de F y P son G y H respectivamente, y permiten predicar sobre todos los instantes del futuro o del pasado. En este lenguaje podemos, por ejemplo, escribir la frase "siempre que llovió, paró" como $H(\text{llueve} \rightarrow F(\neg\text{llueve}))$. La fórmula $\text{llueve} \rightarrow F(\neg\text{llueve})$ se interpreta como "si llueve (en este momento), entonces en algún momento del futuro no lloverá"; al precederla con el operador H estamos pidiendo que la fórmula sea verdadera en todo instante del pasado.

Prior también encontró una importante limitación en la lógica temporal básica: en ella no se pueden expresar ideas tales como "ayer fui al cine" o "si en cinco minutos no llega, me voy". Como veremos más adelante, detrás de esta idea se encuentra una importante limitación de las lógicas modales estándar. En sus últimos años, Prior investigó distintas maneras de enriquecer las lógicas temporales para resolver estos problemas. Los resultados a los que llegó, redescubiertos recientemente, anticipan de alguna manera ideas con las que se está empezando a trabajar hoy en día, casi cuarenta años después.

La lógica temporal es también un ejemplo de lógica multi-modal. Si bien F y P están íntimamente relacionados (e.g. $\varphi \rightarrow \neg F\neg P\varphi$ es verdadero, para todo φ), no es posible escribir uno en función del otro. Ambos deben incluirse como operadores primitivos de la lógica temporal; a partir de ellos es posible derivar G y H. De aquí en más utilizaremos un lenguaje uniforme para notar a todas las lógicas (multi-)modales.

Definición 1.1. Dados PROP un conjunto numerable de símbolos de proposición, y REL un conjunto numerable de símbolos de relación[1], el conjunto $\mathcal{M}$ de fórmulas bien formadas de la lógica (multi) modal básica definidas sobre PROP y REL se define inductivamente como:

$$\mathcal{M} ::= p \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \langle r\rangle\varphi$$

[1] Como veremos en la Sección 1.2, una característica de los operadores modales es que su semántica está definida en términos de relaciones de accesibilidad.

donde $p \in \mathsf{PROP}$, $r \in \mathsf{REL}$ y $\varphi, \varphi' \in \mathcal{M}$. El operador $[r]$ se define como $[r]\varphi \equiv \neg\langle r\rangle\neg\varphi$. El resto de los operadores lógicos habituales se definen de la forma usual. En los casos en que REL sea un conjunto unitario, usaremos $\Diamond$ y $\Box$ como operadores modales.

1.2 La revolución semántica

Treinta años después de la publicación del trabajo original de Lewis, el interés por las lógicas modales se estaba extinguiendo. Hasta ese momento, cada una de las lógicas modales se caracterizaba únicamente por el conjunto de sus teoremas, los cuáles a su vez se expresaban utilizando sistemas axiomáticos. Pero era, justamente, este enfoque esencialmente sintáctico que tenían las investigaciones en el área lo que frenaba su desarrollo. A modo de ejemplo, cada vez que se proponía una nueva lógica, se daba su sistema axiomático característico, y se la clasificaba buscando su relación de inclusión respecto a otras lógicas conocidas. Sin embargo, mostrar que hay una relación de inclusión entre los lenguajes que inducen dos sistemas axiomáticos arbitrarios es una tarea inherentemente compleja.

Se puede decir, sin exagerar, que se vivió una verdadera *revolución* en el área cuando, a principios de la década de 1960, diversos trabajos, entre los que sobresalen los de Samuel Kripke [Kripke, 1959; Kripke, 1963a; Kripke, 1963b], mostraron que era posible encarar el estudio de las lógicas modales desde un punto de vista completamente novedoso. Lo que estos investigadores descubrieron es que las lógicas modales pueden ser *interpretadas* en términos de estructuras matemáticas precisas. A partir de ese momento, una lógica modal dada pasó a caracterizarse no sólo por el conjunto de sus tautologías, sino también por el *tipo de modelos* que admitía. Tareas como la clasificación de una nueva lógica que mencionamos más arriba podían hacerse ahora *semánticamente*, comparando modelos, de una forma simple y elegante.

Las estructuras tomadas como modelos de una lógica modal se conocen hoy en día como *modelos de Kripke*. Desde un punto de vista computacional, un modelo de Kripke es un multigrafo dirigido con nodos etiquetados por conjuntos de símbolos proposicionales. Intuitivamente, cada nodo del grafo representa un *contexto* donde es posible evaluar una fórmula modal, las relaciones de accesibilidad entre puntos están asociadas a las distintas modalidades y la etiqueta de cada punto es una *valuación* que permite determinar el valor de verdad de las proposiciones en dicho contexto.

Definición 1.2 (Modelo de Kripke). Un modelo de Kripke es una estructura $M = \langle W, \{R_i\}, V\rangle$ donde

$$\begin{array}{rl} W & \text{es un conjunto no vacío} \\ R_i \subseteq W \times W & \text{es una relación binaria para cada } R_i \in \mathsf{REL} \\ V(p_i) \subseteq W & \text{para cada } p_i \in \mathsf{PROP} \end{array}$$

A las relaciones R_i las llamamos *relaciones de accesibilidad* entre puntos del modelo.

Ejemplo 1. Intuitivamente, una fórmula de la lógica modal se interpreta en uno de estos modelos con un procedimiento muy similar al que utilizamos para entender lo que representaba $[P](\mathsf{llueve} \rightarrow \langle F\rangle(\neg\mathsf{llueve}))$ al tratar la lógica temporal. Llamemos φ a esta fórmula y tomemos una estructura que pueda ser modelo de la lógica temporal básica: $M = \langle \mathbb{Z}, \{<, >\}, V\rangle$. El conjunto de los enteros representa el tiempo dividido en instantes discretos, mientras que las relaciones de menor y mayor las asociamos, respectivamente, las modalidades F y P.

Determinar si, por ejemplo, M *satisface* φ en 0 (lo cual notamos $M, 0 \models \varphi$) es equivalente a determinar si en cada uno de los instantes x tales que $0 > x$ (es decir, en todos los instantes x anteriores a 0), M satisface $\mathsf{llueve} \rightarrow \langle F\rangle(\neg\mathsf{llueve})$. Para cada x, $M, x \models \mathsf{llueve} \rightarrow \langle F\rangle(\neg\mathsf{llueve})$ si y sólo si, o bien no llueve en el instante x ($x \notin V(\mathsf{llueve})$, o sea, x no pertenece al conjunto de los instantes en que sí llueve), o bien sí llueve, y existe un instante y tal que $x < y$ (y es posterior en el tiempo a x) y no llueve en y.

De alguna forma, lo que estuvimos haciendo fue comenzar la evaluación de la fórmula en un punto del modelo (en este caso, 0), utilizar la valuación V para determinar el valor de verdad de un símbolo de proposición en un punto del modelo, y utilizar las relaciones de accesibilidad

para *movernos* entre puntos de acuerdo a la presencia de ciertas modalidades. En particular, para tratar $[P]$ tuvimos que buscar en *todos los instantes anteriores* a 0, mientras que para tratar $\langle F \rangle$ tuvimos que buscar *algún instante posterior* a cada x.

Definición 1.3 (Interpretación de una fórmula de $\mathcal{M}$). Dado un lenguaje modal $\mathcal{M}$ definido sobre PROP y REL, y una estructura $M = \langle W, \{R_i\}, V \rangle$, la relación de satisfacibilidad $\models$ se define recursivamente como:

$$\begin{array}{rll} M, w \models p_i & \text{sii} & w \in V(p_i), p_i \in \mathsf{PROP} \\ M, w \models \neg\varphi & \text{sii} & M, w \not\models \varphi \\ M, w \models \varphi_1 \wedge \varphi_2 & \text{sii} & M, w \models \varphi_1 \text{ y } M, w \models \varphi_2 \\ M, w \models \langle R_i \rangle \varphi & \text{sii} & \text{existe } w' \in W \text{ tal que } wR_iw' \text{ y } M, w' \models \varphi \end{array}$$

Cuando para todo $w \in W$ se cumpla $M, w \models \varphi$ diremos simplemente $M \models \varphi$.

A la parte exclusivamente *relacional* de un modelo de Kripke (el multigrafo sin etiquetas) se la denomina *frame*; por esto mismo se suele decir que un modelo de Kripke es un frame más una valuación. Los frames pueden clasificarse de acuerdo a características *estructurales* de su relación de accesibilidad. Por ejemplo, $\langle \mathbb{N}, < \rangle$ y $\langle \mathbb{R}, < \rangle$ constituyen dos frames distintos, pero la relación de accesibilidad de ambos es *transitiva*; ambos frames pertenecen, entonces, a la *clase de los frames transitivos.*

Dado un modelo M, decimos que una fórmula φ es *válida* en M si en todo punto w de M se cumple $M, w \models \varphi$. La noción de validez se puede extender a los frames. Por ejemplo, $\Diamond\Diamond p \rightarrow \Diamond p$ es válida en todos los modelos que se puedan armar a partir del frame $\langle \mathbb{N}, < \rangle$; con lo cual decimos que esta fórmula es *válida* en $\langle \mathbb{N}, < \rangle$. En realidad, $\Diamond\Diamond p \rightarrow \Diamond p$ es *válida* en la clase de todos los frames transitivos.

Lo que Kripke y otros descubrieron es que los sistemas axiomáticos que se habían usado durante treinta años para capturar nociones intuitivas como posibilidad u obligación, en realidad estaban caracterizando sintácticamente clases de frames de interés más general. Por ejemplo, el sistema $S4$, uno de los que Lewis había investigado en sus trabajos sobre *posibilidad*, genera todas las fórmulas y sólo las fórmulas que son *válidas* en los frames cuya relación de accesibilidad es transitiva y reflexiva.

En estos trabajos, al concepto de *posibilidad* se le dio una semántica más precisa usando modelos de Kripke. Intuitivamente, la idea es considerar que el conjunto de nodos del modelo representa la colección de mundos posibles. Es debido a esta representación que a los puntos de un modelo se los suele llamar también *mundos* o *mundos posibles.* De aquí en más usaremos *punto*, *nodo* y *mundo* como sinónimos.

Todos estos resultados atrajeron a investigadores de nuevas ramas de la matemática hacia las lógicas modales, lo cual revitalizó el trabajo en este área. Sin embargo, muchos investigadores tradicionales, que se habían interesado por las connotaciones filosóficas de las distintas modalidades, lentamente fueron perdiendo interés en el tema.

1.3 Las lógicas modales, hoy

Hoy en día ya no se habla de *la* lógica modal. Existe una gran de variedad de lógicas con modalidades, desde las clásicas unimodales hasta lógicas con infinitas modalidades (e.g. PDL). En las últimas dos décadas las lógicas modales se han convertido en herramientas de utilidad práctica, especialmente en diversas áreas de la Lingüística y la Computación. Dentro de esta última, se usan lógicas modales en ámbitos tan variados como la especificación y verificación de sistemas, la representación del conocimiento, y *planning* en inteligencia artificial.

¿Por qué resultan las lógicas modales útiles en disciplinas tan diversas? Tal vez se deba a una combinación de razones. Para empezar, en muchas disciplinas se trabaja continuamente con estructuras que se pueden interpretar como grafos dirigidos y estas lógicas son ideales para expresar condiciones sobre este tipo de estructuras (ya vimos en la Definición 1.2 que un modelo de Kripke no es más que un grafo). Además, las modalidades son, de alguna manera, modulares; esto es, uno

puede agregar exactamente aquellas modalidades que necesite y obtener así una lógica *a medida*. Por último, estas lógicas tienen en general buenas propiedades metalógicas; en particular, la validez de una fórmula constituye un problema usualmente decidible (sobre este punto volveremos en el siguiente capítulo).

Dijimos que las lógicas modales permiten trabajar cómodamente con estructuras relacionales, pero ciertamente no son las únicas lógicas que se pueden utilizar a tal efecto. Sin ir más lejos, no hay fórmula del lenguaje modal básico que no pueda ser expresada en lógica de primer orden[2]. De hecho, es fácil obtener una fórmula de la lógica de primer orden equivalente a una fórmula de la lógica modal básica utilizando la *traducción estándar*.

Definición 1.4 (Traducción estándar). Las funciones mutuamente recursivas ST_x y ST_y traducen fórmulas de $\mathcal{M}$ a la lógica de primer orden de la siguiente manera:

$$\begin{array}{rcl|rcl} ST_x(p_j) & = & P_j(x),\ p_j \in \mathsf{PROP} & ST_y(p_j) & = & P_j(y),\ p_j \in \mathsf{PROP} \\ ST_x(\neg\varphi) & = & \neg ST_x(\varphi) & ST_y(\neg\varphi) & = & \neg ST_y(\varphi) \\ ST_x(\varphi \wedge \psi) & = & ST_x(\varphi) \wedge ST_x(\psi) & ST_y(\varphi \wedge \psi) & = & ST_y(\varphi) \wedge ST_y(\psi) \\ ST_x(\langle R\rangle\varphi) & = & (\exists y)(R(x,y) \wedge ST_y(\varphi)) & ST_y(\langle R\rangle\varphi) & = & (\exists x)(R(y,x) \wedge ST_x(\varphi)) \end{array}$$

La traducción estándar es uno de los resultados de un reciente interés por investigar el poder expresivo de las lógicas modales. Hoy se sabe que éstas están incluidas en un interesante fragmento decidible de la lógica de primer orden denominado *guarded fragment*[3] [Andréka *et al.*, 1998]. Esta caracterización no le quita interés a estas lógicas. Para empezar, en muchos contextos, las lógicas modales suelen ser más simples de usar que la lógica de primer orden. Más importante aún, puede demostrarse que ciertos frames no son caracterizables por ninguna fórmula del guarded fragment (ni de extensiones usuales del mismo). Este es el caso, por ejemplo, de los frames transitivos. Esto significa que en esos casos no tenemos garantizado a priori que se apliquen las buenas propiedades del guarded fragment (e.g. decidibilidad) sobre dichos frames, sino que debe demostrarse cada caso por separado.

Lógica modal computacional. Un interesante resultado de la investigación en expresividad de las lógicas modales es el que las relaciona en forma estrecha con las *lógicas para la descripción* (*description logics* o DL). Las DL [Baader *et al.*, 2003] son una familia de lógicas utilizadas principalmente para construir *sistemas de representación de conocimiento* (*knowledge representation systems* o KRS). Estos sistemas están formados por una *base de conocimiento* (*knowledge base*) y una serie de *servicios de razonamiento* (*reasoning services*). Una base de conocimiento la podemos dividir en el conjunto de *conceptos*, *relaciones* y *reglas* que regulan el universo que estamos modelando y en una serie de *hechos* concretos acerca del mismo. A modo de ejemplo, con una DL podemos armar un sistema descrito por las siguientes propiedades:

- "los hombres son mortales"
- "los trabajadores son hombres"
- "los filósofos y los vendedores de bronceador son trabajadores"
- "los feriados y los días hábiles son días de la semana"
- "los trabajadores sólo *van a trabajar* los días hábiles"

y del cual sepamos hechos concretos como:

- "Sócrates es un filósofo"
- "Juan es un vendedor de bronceador"

[2] Sin embargo, utilizando sistemas axiomáticos modales se pueden definir frames que no son caracterizables con fórmulas de la lógica de primer orden, aunque sí con expresiones de la de segundo orden.

[3] Es interesante notar el extraño itinerario que han recorrido las lógicas modales, que habiendo nacido como una extensión de la lógica proposicional, terminan como subconjunto no trivial de la lógica de primer orden.

- "mañana es feriado"

Los servicios de razonamiento que provee un DL-KRS pueden ser muy variados, aunque lo mínimo que se espera de ellos es que permitan determinar, por ejemplo, la verdad de afirmaciones como "Juan es mortal", "Sócrates *va a trabajar* mañana" o "todos los hombres son vendedores de bronceador". También es común que puedan responder consultas enumerativas como "¿quiénes son todos los filósofos?". Tanto los servicios característicos como las construcciones descriptivas de los DL-KRS más modernos se han ido incorporando siguiendo fundamentalmente dos lineamientos: qué se necesita de un KRS en una aplicación concreta y qué se puede agregar a una implementación sin sacrificar eficiencia. Esta es una disciplina en la cual el aparato teórico corre en general por detrás de la práctica.

Ahora bien, como ya dijimos, existe una estrecha relación entre las lógicas modales y las DL. En [Schild, 1991], Schild muestra que es posible ver ciertas DL como lógicas modales. Por ejemplo, $\mathcal{ALC}$, una de las DL más elementales, y la lógica modal básica son simples variantes sintácticas. Schild usó esta equivalencia para poder incorporar a las DL resultados de complejidad que ya se conocían para la lógica modal. Por su parte, todo el trabajo puesto en desarrollar DL-KRS más eficientes puede ser reutilizado en problemas de decisión propios de algunas lógicas modales (más sobre esto en el Capítulo 2)

Las DL son tal vez el mejor ejemplo del aspecto más *computacional* de las lógicas modales. Los DL-KRS, por su parte, muestran claramente que para este tipo de aplicaciones se requiere más que sólo resultados teóricos y buenas propiedades metalógicas. Ya no alcanza con investigar desde un punto de vista teórico problemas básicos como el de la validez de una fórmula, sino que se vuelve necesario contar con buenos algoritmos para resolverlos. Más aun, estos algoritmos deben ser implementados y contrastados empíricamente contra otros. Las lógicas modales, si bien en general son decidibles para el problema de la validez, tienen un costo computacional elevado. Tanto la lógica modal básica como la lógica temporal básica, por ejemplo, son PSPACE-complete para este problema. Sin embargo, ésta es la complejidad del peor caso, y comprobaciones empíricas (de las cuales los DL-KRS son un buen ejemplo) muestran que para un gran número de instancias de este problema se pueden obtener respuestas en tiempos razonables.

1.4 Las lógicas híbridas y las limitaciones de las lógicas modales tradicionales

La lógica temporal básica que vimos en la primera sección no es más que una lógica multi-modal en la que se imponen ciertas restricciones sobre las relaciones de accesibilidad (transitividad, no reflexividad, anti-simetría, xFy si y sólo si yPx). Como ya mencionamos en ese momento, Prior encontró que esta lógica era insuficiente para expresar algunos conceptos importantes. En particular, si consideramos que en los modelos habituales de la lógica temporal los *mundos* corresponden a instantes de tiempo, la lógica temporal básica no permite referirse a momentos particulares (e.g. "ayer", o "dentro de cinco minutos"). Este no parece un problema menor para una lógica temporal.

Este mismo tipo de limitaciones aparecen más generalmente en el resto de las lógicas modales tradicionales. Si bien éstas parecen ser ideales para trabajar con estructuras relacionales, no proveen ninguna forma de referirse a elementos específicos del modelo. Esto significa que no es posible predicar sobre puntos particulares del mismo, ni tampoco distinguir mundos distintos si ambos satisfacen las mismas fórmulas.

¿Hay alguna razón especial por la cual esto deba ser así? Al repasar la historia de las lógicas modales queda la impresión de que esto es casi una coincidencia: los primeros lógicos modales buscaban una lógica que pudiera tratar con conceptos tales como necesidad, obligación, etc; varias décadas más tarde otros lógicos observaron que éstas permitían capturar muchos conceptos de interés en ciertas estructuras matemáticas; hoy en día notamos que aun así no pueden expresar algunas nociones importantes. Visto el problema desde esta perspectiva histórica, es posible

conjeturar que, simplemente, la sintaxis de la lógica modal básica no estaba pensada para tratar con modelos relacionales.

Podemos encontrar una continuación de esta idea en el surgimiento de las llamadas *lógicas híbridas*. Se trata de una familia de extensiones, con diverso poder expresivo, de las lógicas modales tradicionales. El rasgo común a todas las lógicas híbridas es la presencia de *nominales*. Intuitivamente, un nominal sería un *nombre único* para un mundo del modelo. Desde un punto de vista sintáctico, un nominal es similar a un símbolo de proposición, y puede ser usado en cualquier contexto donde éste sea aceptable (no así al revés). Por ejemplo, si i y j son nominales, y p un símbolo de proposición, podemos escribir fórmulas como $i \wedge p \wedge \Diamond(p \wedge \Box j)$. La menos expresiva de todas estas lógicas es la *lógica híbrida mínima*, que es la que se obtiene al agregar sólo nominales a la lógica modal básica.

Definición 1.5. Dados PROP un conjunto numerable de símbolos de proposición, NOM un conjunto numerable de nominales y REL un conjunto numerable de símbolos de relación, decimos que ATOM = PROP $\cup$ NOM y definimos inductivamente el conjunto $\mathcal{H}$ de fórmulas bien formadas de la lógica híbrida mínima definida sobre ATOM y REL como:

$$\mathcal{H} ::= a \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \langle r \rangle \varphi$$

donde $a \in$ ATOM, $r \in$ REL y $\varphi, \varphi' \in \mathcal{H}$. El resto de los operadores lógicos se definen de la manera usual.

La semántica de la lógica híbrida mínima está dada por un tipo particular de modelo de Kripke en el cual la valuación garantiza que cada nominal vale en uno y sólo uno de los puntos del modelo. Llamamos *modelos híbridos* a estas estructuras.

Definición 1.6 (Modelo híbrido). Un modelo híbrido es una estructura $M = \langle W, \{R_i\}, V \rangle$ donde

$$\begin{array}{rl} W & \text{es un conjunto no vacío} \\ R_i \subseteq W \times W & \text{es una relación binaria para cada } R_i \in \mathsf{REL} \\ V(p_i) \subseteq W & \text{para cada } p_i \in \mathsf{PROP} \\ V(n_i) = \{w_i\} \subseteq W & \text{para cada } n_i \in \mathsf{NOM} \end{array}$$

Es decir, los elementos de NOM denotan conjuntos unitarios del dominio del modelo, i.e. *nombran* mundos particulares del modelo. Una fórmula de $\mathcal{H}$ se interpreta sobre un modelo híbrido de la misma forma indicada en la Definición 1.3, con el agregado, obviamente, del caso:

$$M, w \models n_i \text{ sii } w \in V(n_i), n_i \in \mathsf{NOM}.$$

Ejemplo 2. Sea $F = \langle W, R \rangle$ una estructura relacional en la que W es el conjunto de países y wRw' si y sólo si w y w' son países limítrofes para todo par $w, w' \in W$ (F es un elemento de la clase de frames irreflexivos y simétricos). Usando este lenguaje, podemos expresar un concepto como "llueve en Grecia" diciendo: grecia $\wedge$ llueve, donde grecia es un nominal y llueve un símbolo de proposición. En cualquier modelo híbrido M que construyamos a partir de F, esta fórmula será satisfecha sólo por el mundo denotado por grecia, y sólo si en él vale la proposición llueve. Una traducción estricta de este predicado de $\mathcal{H}$ al castellano sería "es grecia y llueve".

En las lógicas modales tradicionales, para evaluar una fórmula nos situamos en un punto del modelo e intentamos recorrer otros puntos relacionados de acuerdo a las distintas modalidades que aparezcan. En el ejemplo anterior hacemos lo mismo, sólo que la presencia del nominal grecia en conjunción con el resto nos dice que sólo tiene sentido pararnos en el mundo donde grecia es verdadero (en todos los demás, la fórmula ya no vale). Ahora bien, ¿tenemos en $\mathcal{H}$ alguna manera de referirnos a lo que sucede en dos puntos distintos del modelo? Por ejemplo, ¿cómo podríamos decir "llueve en Grecia y en un pais que limita con China"? Usando el recurso anterior, uno podría estar tentado de escribir: (grecia $\wedge$ llueve) $\wedge$ (china $\wedge \Diamond$llueve). Sin embargo, para que esta fórmula sea verdadera en un mundo w de un modelo M, tanto grecia como china deberían ser verdaderos

en w. Si bien esto es posible (un punto del modelo puede tener más de un nominal que lo nombre), la fórmula no está diciendo lo que queremos.

$\mathcal{H}$ es más expresiva que $\mathcal{M}$, pero en muchos casos, como acabamos de ver, no es suficiente. Una de las lógicas híbridas más interesantes es la llamada $\mathcal{H}(@)$, que se obtiene al agregar a la lógica híbrida mínima $\mathcal{H}$ el *operador de satisfacción*. Este operador, que se nota con una @ permite expresar que determinada proposición es verdadera en algún mundo particular. El enunciado "llueve en Grecia y en un país que limita con China" puede escribirse $(@_{\mathsf{grecia}}\,\mathsf{llueve}) \wedge (@_{\mathsf{china}} \Diamond \mathsf{llueve})$. Intuitivamente, para que esta fórmula sea verdadera, la fórmula llueve debe ser verdadera en el mundo denotado por grecia y la fórmula $\Diamond$llueve debe serlo en el mundo al que china hace referencia.

Definición 1.7. Dados PROP un conjunto numerable de símbolos de proposición, NOM un conjunto numerable de nominales y REL un conjunto numerable de símbolos de relación, decimos que ATOM = PROP $\cup$ NOM y definimos inductivamente el conjunto $\mathcal{H}(@)$ definido sobre ATOM y REL como:

$$\mathcal{H}(@) ::= a \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \langle r \rangle \varphi \mid @_i\,\varphi$$

donde $a \in$ ATOM, $r \in$ REL, $i \in$ NOM y $\varphi, \varphi' \in \mathcal{H}(@)$. El resto de los operadores lógicos se definen de la manera usual. A aquellas fórmulas de la forma $@_i\,\varphi$ las llamaremos *fórmulas-@*.

Definición 1.8 (Interpretación de una fórmula de $\mathcal{H}(@)$). Dado un lenguaje modal $\mathcal{M}$ definido sobre PROP y REL, y una estructura $M = \langle W, \{R_i\}, V \rangle$, la relación de satisfacibilidad $\models$ se define recursivamente como:

$$\begin{array}{rll}
M, w \models p_i & \text{sii} & w \in V(p_i), p_i \in \mathsf{PROP} \\
M, w \models n_i & \text{sii} & w \in V(n_i), n_i \in \mathsf{NOM} \\
M, w \models \neg\varphi & \text{sii} & M, w \not\models \varphi \\
M, w \models \varphi_1 \wedge \varphi_2 & \text{sii} & M, w \models \varphi_1 \text{ y } M, w \models \varphi_2 \\
M, w \models \langle R_i \rangle \varphi & \text{sii} & \text{existe } w' \in W \text{ tal que } wR_iw' \text{ y } M, w' \models \varphi \\
M, w \models @_i\,\varphi & \text{sii} & M, w' \models \varphi \text{ donde } V(i) = \{w'\}
\end{array}$$

Cuando para todo $w \in W$ se cumpla $M, w \models \varphi$ diremos simplemente $M \models \varphi$.

En general, en esta tesis nos centraremos en la lógica $\mathcal{H}(@)$, y en particular en las fórmulas-@ de esta lógica. Es fácil ver que cualquier fórmula-@ $@_i\,\varphi$ verifica que, si $M, w \models @_i\,\varphi$, entonces $M \models @_i\,\varphi$.

De todos modos, es interesante notar que existen otros operadores y otras lógicas híbridas además de $\mathcal{H}(@)$. Por ejemplo, $\mathcal{H}(\downarrow)$ se obtiene al agregar el operador $\downarrow$ a la lógica híbrida mínima. Este operador permite referirse a un punto del modelo sin necesidad de asociarle de antemano un nombre unívoco. Como ejemplo, tomemos la fórmula $\Box(\downarrow x.\Diamond\neg x)$ y llamémosla φ. Si esta fórmula vale en el mundo w de un modelo M, entonces en cada mundo accesible desde w debe valer $(\downarrow x.\Diamond\neg x)$. Esta última fórmula dice "llamemos x al mundo sobre el cual estamos evaluando; x está relacionado con algún mundo que no sea x". Lo que $\downarrow x$ hace es llamar x al mundo en que "estamos parados". En forma similar a lo que sucede con los cuantificadores de la lógica de primer orden, el operador $\downarrow$ *liga* las sucesivas apariciones de x[4]. En definitiva, para que φ sea verdadera en w, todo mundo w' accesible desde w debe estar relacionado con algún mundo distinto de sí mismo. Fórmulas como φ no pueden expresarse en $\mathcal{H}(@)$. El poder expresivo de ambos operadores se combina en la lógica $\mathcal{H}(@, \downarrow)$.

Las lógicas híbridas resuelven algunas de las irregularidades de las lógicas modales tradicionales. Si bien habíamos visto que los sistemas axiomáticos modales se pueden utilizar para caracterizar clases de frames, es un hecho conocido que no siempre es posible hacer esto, aun para ciertas clases importantes de frames. A modo de ejemplo, no es posible capturar las nociones de irreflexividad, asimetría, antisimetría ni intransitividad. Con nominales se resuelve este problema; todos estas clases de frames son caracterizables con sistemas axiomáticos híbridos.

[4]Técnicamente, se considera que x, por ser ligable, no es un nominal sino una *variable de estado*.

En [Areces and de Rijke, 2001; Areces, 2000] se muestra que es posible encontrar vínculos aun más fuertes entre diversas lógicas híbridas y las DL. Usando nominales, por ejemplo, es posible traducir expresiones de una DL que incluya referencias a elementos concretos del dominio (el caso de Juan, Sócrates y mañana en nuestro ejemplo anterior).

¿Qué sucede con la complejidad computacional de las lógicas híbridas? Sorprendentemente, las distintas lógicas híbridas muestran comportamientos muy dispares. El problema de la validez de una fórmula en $\mathcal{H}(@)$ se mantiene en PSPACE. Sin embargo, el mismo problema en la lógica temporal básica enriquecida con nominales y el operador @ (*nominal tense logic*) pasa a ser EXPTIME-complete[5]. En el extremo opuesto del espectro, $\mathcal{H}(\downarrow)$ es tan expresiva que su problema de validez es indecidible (obviamente, lo mismo sucede con $\mathcal{H}(@, \downarrow)$).

1.5 Estructura de esta tesis

En este capítulo introdujimos las lógicas modales e híbridas y observamos que existe un interés concreto por contar con más y mejores algoritmos para tratar el problema de la validez de una fórmula. En el Capítulo 2 repasamos brevemente los principales métodos para resolver este problema y presentamos, en particular, un cálculo basado en resolución propuesto en [Areces *et al.*, 2001]. En el Capítulo 3 tomamos este cálculo y le agregamos estrategias de orden y selección que son estándar en resolución para lógica de primer orden. Probamos, luego, que este agregado preserva completitud refutacional. En el Capítulo 4 introducimos leves modificaciones en el cálculo y mostramos que, además de preservar la consistencia y completitud refutacional, el cálculo obtenido es un método de decisión para el problema de la validez de $\mathcal{H}(@)$. Finalmente, en el Capítulo 5 comentamos los resultados obtenidos en pruebas preliminares con una implementación del nuevo cálculo.

[5] De hecho, basta con agregarle sólo un nominal a la lógica temporal básica para obtener una lógica con esta complejidad.

Capítulo 2

Métodos de prueba

El problema de la *validez* de una fórmula de una lógica modal $\mathcal{L}$ cualquiera (e.g. $\mathcal{M}$ ó $\mathcal{H}(@)$) consiste en determinar, para cualquier $\varphi \in \mathcal{L}$, si para todo modelo M en la signatura de $\mathcal{L}$ y todo mundo w en el dominio de M se verifica $M, w \models \varphi$. Su problema *dual* es el de la *satisfacibilidad*; esto es, dada una fórmula $\varphi \in \mathcal{L}$, determinar si existen un modelo M y un mundo w tales que $M, w \models \varphi$. Decimos que son duales por cuanto una fórmula φ es válida si y sólo si $\neg\varphi$ es insatisfacible.

Como veremos en este capítulo, existen diversos mecanismos para resolver instancias del problema de la validez (o satisfacibilidad) de una fórmula. Lo mínimo que se le exige a éstos es que sean *consistentes*, es decir, que siempre que den una respuesta, ésta sea *correcta*. En general, se espera de ellos que, además, puedan dar respuesta a *cualquier instancia* del problema; se dice de los que cumplen con esta condición que son *completos*.

Las lógicas modales tradicionales y muchas de sus extensiones híbridas (e.g. $\mathcal{H}(@)$) se distinguen por ser *decidibles* para el problema de la validez (y, por lo tanto, para el de la satisfacibilidad). Esto significa que existe al menos un mecanismo completo y consistente que garantiza una respuesta en un *número finito* (aunque no necesariamente chico) *dc pasos*. A un mecanismo de este tipo se lo llama un *método de decisión*.

En lo que sigue, haremos un breve repaso de los principales mecanismos propuestos para tratar el problema de la validez de una fórmula para diversas lógicas modales. Comenzaremos por ver, en la Sección 2.1 un mecanismo sin utilidad práctica, pero que provee una demostración simple de la decidibilidad de $\mathcal{M}$. En la Sección 2.2 presentamos varios algoritmos basados en el método de tableaux; este método es el que se usa habitualmente en aplicaciones de DL (ver Sección 1.3). Un resultado directo de las investigaciones recientes sobre expresividad de lógicas modales son los métodos basados en traducción a otros formalismos que se comentan en la Sección 2.3. Finalmente, dedicamos la Sección 2.4 al método de resolución y nos centramos en el cálculo propuesto en [Areces *et al.*, 2001], del cual nos ocuparemos en los capítulos siguientes. Dedicamos parte de esta sección al método de resolución para lógica de primer orden, especialmente, a las restricciones de orden y selección; esto nos servirá como motivación para el tema del Capítulo 3.

2.1 Decidibilidad del problema de la validez

Se dice que una lógica tiene la *propiedad de modelos finitos* (*finite model property*) si toda fórmula satisfacible de la lógica tiene algún modelo $M = \langle W, R, V \rangle$ donde W es finito. Se puede ver que tanto $\mathcal{M}$ como $\mathcal{H}(@)$ tienen esta propiedad[1] [Blackburn *et al.*, 2002; Areces *et al.*, 1999]. Además, como en toda fórmula puede aparecer sólo un número finito de modalidades y proposiciones (i.e. trabaja con un lenguaje finito), podemos asegurar que si φ es satisfacible, entonces tiene un modelo $M = \langle W, R, V \rangle$ donde W, R y el dominio de V son conjuntos finitos. En lo que sigue veremos

[1] No la tienen, sin embargo, ni $\mathcal{H}(\downarrow)$ ni $\mathcal{H}(@, \downarrow)$.

cómo aprovechar estas propiedades para dar un método de decisión para $\mathcal{M}$; este método se adapta trivialmente para el caso de $\mathcal{H}(@)$.

Dados M, un modelo *finito*; w, un punto de este modelo; y φ, una fórmula cualquiera de $\mathcal{M}$, es posible determinar en un número finito de pasos si $M, w \models \varphi$. Para ello alcanza con aplicar recursivamente los pasos de la Definición 1.3, teniendo en cuenta que el existencial que se debe usar al *procesar* los diamantes está acotado pues el modelo es finito. De aquí se desprende que si M es finito, también se puede determinar en un número finito de pasos si para algún punto w de M se cumple $M, w \models \varphi$.

Para todo k, el número de modelos *distintos* de exactamente k mundos, dado un lenguaje finito, módulo isomorfismos, es finito. Luego, es posible generar, en un número finito de pasos, todos los posibles modelos de exactamente k mundos. Por lo tanto, dados k y $\varphi \in \mathcal{M}$, podemos determinar en finitos pasos si existe algún modelo de exactamente k mundos que satisfaga φ.

Tomemos una fórmula φ que sea satisfacible. Podemos buscar el menor modelo finito de φ, comenzando con los modelos de tamaño uno, siguiendo con los de dos y así sucesivamente; eventualmente, en una cantidad posiblemente grande pero finita de pasos lo encontraremos. Esto está garantizado porque asumimos que φ es satisfacible, sabemos que $\mathcal{M}$ tiene la propiedad de modelos finitos, y ya vimos que los chequeos que debemos realizar requieren una cantidad finita de pasos cada uno. Si, por el contrario, φ fuera insatisfacible, entonces seguiríamos buscando indefinidamente.

Por otro lado, ya vimos que existen sistemas axiomáticos completos y consistentes para $\mathcal{M}$ (para axiomatizaciones de $\mathcal{M}$ y $\mathcal{H}(@)$ ver [Blackburn *et al.*, 2002]). Una *demostración* según estos sistemas axiomáticos es una secuencia finita de fórmulas bien formadas tal que cada una de ellas corresponde a un axioma o a la aplicación de una regla de inferencia sobre una o más de las fórmulas precedentes. Se dice que un sistema axiomático genera como *teorema* a φ si existe una *demostración* cuyo último término sea φ. Si un sistema axiomático es completo y consistente, entonces puede generar como teorema todas las fórmulas que son válidas en la teoría (y sólo dichas fórmulas).

Dados un sistema axiomático y una secuencia de fórmulas, es sencillo verificar, en un número finito de pasos, si la secuencia corresponde a una *demostración* válida en dicho sistema. Además, es posible enumerar recursivamente todas las fórmulas bien formadas de $\mathcal{M}$ y de la misma forma, hacerlo con todas las secuencias finitas de fórmulas. Luego, dada una fórmula φ arbitraria, es posible enumerar todas las secuencias finitas de fórmulas cuyo último término sea φ. Ahora bien, supongamos que φ es válida; esto quiere decir que podemos buscar la menor *demostración* de φ comenzando por la primer secuencia de fórmulas cuyo último término sea φ, chequeando si se trata de una *demostración* correcta, siguiendo con la segunda y así sucesivamente. Eventualmente, en un número extremadamente grande pero finito de pasos, encontraremos dicha demostración. Esto está garantizado porque asumimos que φ es válida, el sistema axiomático es completo y todos los chequeos intermedios se pueden realizar en un número finito de pasos. Si φ no fuera válida, seguiríamos buscando indefinidamente.

Ya podemos armar nuestro método de decisión. Dada una fórmula φ cualquiera, o bien φ es universalmente válida, o bien $\neg\varphi$ es satisfacible en algún modelo. Si φ es universalmente válida, el método de los sistemas axiomáticos terminará en finitos pasos. Si $\neg\varphi$ es satisfacible, entonces el método de los modelos finitos terminará en finitos pasos. Ejecutar ambos procedimientos "en paralelo" (por ejemplo, alternando un paso de cada uno) y terminar en cuanto alguno de los dos encuentre una respuesta es un método efectivo de decisión para el problema de la satisfacibilidad.

2.2 Tableaux

Existen varios tipos de algoritmos de tableau para lógicas modales e híbridas. Todos ellos intentan construir un modelo de Kripke que satisfaga una fórmula dada utilizando reglas de transformación sintácticas. Las estructuras internas que manejan son, en general, árboles de fórmulas o de conjuntos de fórmulas. En esta sección repasaremos el método tradicional de tableau modal, el método de tableau etiquetado, y una variación de este último para $\mathcal{H}(@)$. Los ejemplos que veremos serán

para el caso unimodal, pero su generalización es trivial.

2.2.1 Tableau modal tradicional o *implícito*

Un sistema de tableau modal tradicional es "un procedimiento de refutación que descompone un conjunto de fórmulas dado en una red de conjuntos, donde cada conjunto representa un mundo posible en el modelo de Kripke asociado" (Goré [Goré, 1999]). La "red de conjuntos" es, en realidad, un árbol; su raíz corresponde a un conjunto unitario con la fórmula cuya satisfacibilidad se quiere comprobar. Un sistema de tableau está compuesto de *reglas*, cada una de las cuales nos da pautas de cómo agregar nuevos hijos a una hoja del árbol. Cada rama en un árbol de tableau representa un modelo de Kripke; dos conjuntos de fórmulas consecutivos representan el mismo mundo posible o dos mundos posibles unidos por la relación de accesibilidad dependiendo de la regla de tableau que les dio origen.

Durante la construcción del árbol es posible encontrarse con conjuntos *inconsistentes*, es decir, conjuntos en los que aparecen un par de fórmulas φ y $\neg\varphi$. Claramente, las ramas a las que pertenecen estos conjuntos no pueden denotar ningún modelo de Kripke y se las denomina, por ello, *ramas cerradas*. Cuando una rama no-cerrada no puede ser *extendida* mediante la aplicación de ninguna regla de tableau, se dice que se encuentra *saturada*. Una rama saturada codifica un modelo de la fórmula original. Un algoritmo de tableau constituye un método de decisión cuando que se puede demostrar que toda rama del árbol eventualmente se satura o se cierra en un número finito de pasos, que el número de ramas es finito y que cada nodo contiene un número finito de fórmulas. Si todas las ramas se cierran, la fórmula original es insatisfacible.

En general, los sistemas de tableau de este tipo se describen dando las reglas que se utilizan en la construcción del árbol de refutación. La notación usual es:

$$(\rho)\ \frac{N}{D_1 \mid D_2 \ldots \mid D_m}$$

donde ρ representa el *nombre* de la regla, y N y D_k representan conjuntos de fórmulas. A N se lo llama el *numerador* de la regla y los $D_1 \ldots D_m$ constituyen el *denominador*. Una regla de este estilo debe leerse: "si una rama *no cerrada* del árbol parcialmente construido tiene como hoja un conjunto *compatible* con N, entonces a dicha hoja se le deben agregar como hijos los conjuntos de fórmulas $D_1, D_2 \ldots, D_m$". En general, N se escribe utilizando expresiones de la forma $X; \varphi_1; \varphi_2 \ldots; \varphi_n$. Una hoja del árbol será *compatible* con un numerador de este tipo si todos los φ_k están presentes en dicha hoja; X representa en este contexto el conjunto de las restantes fórmulas de la hoja.

Existen diferentes sistemas de tableau consistentes y completos para las lógicas modales. El siguiente es un ejemplo de tableaux para la lógica modal básica $\mathcal{M}$:

$$(\wedge)\ \frac{X; \varphi \wedge \psi}{X; \varphi; \psi} \qquad (\vee)\ \frac{X; \neg(\varphi \wedge \psi)}{X; \neg\varphi \mid X; \neg\psi} \qquad (\neg)\ \frac{\neg\neg\varphi}{\varphi}$$

$$(\theta)\ \frac{X; Y}{X} \qquad (\Box)\ \frac{\neg\Diamond X; \Diamond\varphi}{\neg X; \varphi}$$

La regla $\Box$ debe leerse: "si en la hoja de una rama no-cerrada tenemos una fórmula $\Diamond\varphi$ y un conjunto de cero o más fórmulas de la forma $\neg\Diamond\psi_k$, entonces se debe agregar como hijo un conjunto con φ y todas las fórmulas ψ_k". La regla θ se usa para eliminar todas las fórmulas que no sean de la forma $\neg\Diamond\psi$ como paso previo a la aplicación de la regla $\Box$.

2.2.2 Tableau etiquetado

El tableau etiquetado constituye otra familia de algoritmos de tableau para la lógica modal. En éstos también se intenta construir un posible modelo para la fórmula dada utilizando una estructura con forma de árbol, pero a diferencia de lo que sucede en el tableau tradicional, en este caso se

identifican explícitamente los distintos mundos del posible modelo y se registra la relación de accesibilidad que hay entre cada par de mundos.

Ahora bien, las lógicas modales tradicionales no poseen un mecanismo sintáctico que permita identificar mundos; para presentar este cálculo necesitamos introducir previamente algunos auxiliares metalógicos. Dado un conjunto numerable L, diremos que una *etiqueta* es una secuencia no vacía de elementos de L. Si σ es una etiqueta y n es un elemento de L, $\sigma.n$ denota una nueva etiqueta que se obtiene al *agregar* n a σ. De la misma manera, si φ es una fórmula bien formada de $\mathcal{M}$ y σ es una etiqueta, diremos que $\sigma :: \varphi$ es una *fórmula etiquetada*. En el contexto del cálculo que veremos a continuación, $\sigma :: \varphi$ dice que la fórmula φ es verdadera en el mundo *etiquetado* con σ. Además, entenderemos que $\sigma.n$ designa un mundo accesible desde σ. Es decir, nos valdremos de la forma de las etiquetas para codificar la relación de accesibilidad entre los mundos.

En un sistema de tableau etiquetado también se utilizan determinadas reglas para descomponer en un árbol una fórmula (etiquetada) inicial, pero a diferencia de lo que sucedía en el tableau tradicional, cada nodo de este árbol consiste de una única fórmula. Aquí también cada rama del árbol representa un posible modelo, pero en este caso, un nodo no equivale a un mundo del modelo, sino que nos indica que la fórmula asociada a él es verdadera en el punto designado por su etiqueta. Con el conjunto de todas las etiquetas que aparecen mencionadas en una rama podemos deducir la relación de accesibilidad entre los mundos del modelo. Si en una rama encontramos apariciones de $\sigma :: \varphi$ y $\sigma :: \neg\varphi$ decimos que dicha rama se encuentra cerrada. Es interesante observar que un tableau etiquetado se parece más a un sistema de tableau para lógica de primer orden como los de Smullyan [Smullyan, 1994].

Veamos un conjunto posible de reglas para la lógica modal básica:

$$(l\wedge)\ \frac{\sigma :: (\varphi \wedge \psi)}{\begin{array}{c}\sigma :: \varphi \\ \sigma :: \psi\end{array}} \qquad (l\vee)\ \frac{\sigma :: \neg(\varphi \wedge \psi)}{\sigma :: \neg\varphi \mid \sigma :: \neg\psi} \qquad (l\neg)\ \frac{\sigma :: \neg\neg\varphi}{\sigma :: \varphi}$$

$$(l\Box)\ \frac{\sigma :: \neg\Diamond\varphi}{\sigma.n :: \neg\varphi}\ (\sigma.n \text{ aparece en la rama actual}) \qquad (l\Diamond)\ \frac{\sigma :: \Diamond\varphi}{\sigma.n :: \varphi}\ (\sigma.n \text{ no aparece en la rama actual})$$

Estas reglas se aplican en forma similar a las reglas de tableau para lógica de primer orden. Si en una rama no-cerrada aparece una fórmula compatible con el numerador de la regla, y dicha regla nunca fue aplicada en esa rama sobre la misma fórmula, entonces se extiende la rama agregando las fórmulas que indica el denominador. Se deben hacer algunas salvedades: la regla $l\wedge$ extiende la rama sobre la que se está trabajando agregando consecutivamente dos nuevos nodos; la regla $l\Box$ asume que en la rama aparece alguna fórmula etiquetada con $\sigma.n$ y debe ser aplicada a lo sumo una vez por cada $\sigma.n$ distinto en la rama; la regla $l\Diamond$ *crea* un nuevo mundo al utilizar una etiqueta que nunca antes se usó en dicha rama.

En el tableau etiquetado, la relación de accesibilidad entre los mundos del modelo es dada en forma explícita. Lo interesante de esto es que se pueden agregar, en forma modular, reglas sencillas para tratar con clases de frames particulares. Por ejemplo, para trabajar con la clase de los frames reflexivos basta con agregar la regla:

$$(lT)\ \frac{\sigma :: \neg\Diamond\varphi}{\sigma :: \neg\varphi}$$

mientras que para la clase de los frames transitivos se puede agregar:

$$(l4)\ \frac{\sigma :: \neg\Diamond\varphi}{\sigma.n :: \neg\Diamond\varphi}\ (\sigma.n \text{ aparece en la rama})$$

Las reglas son además combinables; un sistema de tableau que incluya lT y $l4$ permite trabajar con la clase de frames reflexivos y transitivos.

El tableau etiquetado tiene la ventaja, sobre el tableau implícito, de admitir implementaciones más simples y eficientes, pero, ciertamente, el cálculo pierde un poco de elegancia al vernos obligados a introducir un auxiliar metalógico.

2.2.3 Tableaux y lógica híbrida

Es evidente la semejanza entre las etiquetas del cálculo de la sección anterior y los nominales, y entre las fórmulas etiquetadas y las fórmulas-@. Efectivamente, usando $\mathcal{H}(@)$ podemos trabajar con tableau etiquetado sin necesidad de un soporte metalógico [Blackburn, 2000].

Está claro que si $\varphi \in \mathcal{M}$, entonces $\sigma :: \varphi$ se puede representar en $\mathcal{H}(@)$ como $@_i\,\varphi$, para algún nominal i. Nos falta una manera de dar cuenta de la relación que hay entre σ y $\sigma.n$; pero esto es fácil de expresar en $\mathcal{H}(@)$ usando una fórmula como $@_i \Diamond j$, donde i y j son nominales. En definitiva, el conjunto de fórmulas etiquetadas de $\mathcal{M}$ $\{\sigma :: \varphi, \sigma.n :: \psi\}$ es equivalente al conjunto de fórmulas de $\mathcal{H}(@)$ $\{@_i\,\varphi, @_j\,\psi, @_i \Diamond j\}$.

Podemos reescribir el cálculo de tableau etiquetado para $\mathcal{M}$ en términos de $\mathcal{H}(@)$.

$$(l\wedge')\ \frac{@_i\,(\varphi \wedge \psi)}{\begin{array}{c}@_i\,\varphi\\ @_i\,\psi\end{array}} \qquad (l\vee')\ \frac{@_i\,\neg(\varphi \wedge \psi)}{@_i\,\neg\varphi \mid @_i\,\neg\psi} \qquad (l\neg')\ \frac{@_i\,\neg\neg\varphi}{@_i\,\varphi}$$

$$(l\Box')\ \frac{\begin{array}{c}@_i\,\neg\Diamond\varphi\\ @_i\,\Diamond j\end{array}}{@_j\,\neg\varphi} \qquad (l\Diamond')\ \frac{@_i\,\Diamond\varphi}{\begin{array}{c}@_i\,\Diamond j\\ @_j\,\varphi\end{array}} \quad \begin{array}{l}(j \text{ no aparece en la rama}\\ \text{actual, y } \varphi \text{ no es un nominal)}\end{array}$$

Es necesario notar que este conjunto de reglas no es completo si trabajamos con fórmulas arbitrarias de $\mathcal{H}(@)$. Los problemas aparecen, por ejemplo, cuando se generan fórmulas del estilo $@_i\,j$, donde i y j son nominales. Una fórmula como esta dice "i y j son dos nombres distintos para un mismo mundo del modelo". Es fácil ver que este cálculo no detecta la inconsistencia de una rama donde aparezcan $@_i\,p$, $@_j\,\neg p$ y $@_i\,j$, y podría llegar considerarla, erróneamente, como saturada. Este problema se soluciona incorporando reglas similares a las que se utilizan en el cálculo de tableau para lógica de primer orden con igualdad [Fitting, 1996].

2.2.4 Implementaciones del método de tableau

Tradicionalmente, los métodos de decisión más utilizados para chequear satisfacibilidad en lógicas modales son variaciones del algoritmo de tableau. Las mejores implementaciones de estos algoritmos las encontramos en los demostradores para DL [Möller and Haarslev, 2003]. Probablemente el más destacable de los demostradores estrictamente modales basados en tableau sea, *ModLeanTab* [Beckert and Goré, 1997]; un demostrador escrito en Prolog, de performance aceptable pero cuya característica más memorable es que su tamaño es de sólo 360 bytes. Hydra [Blackburn *et al.*, 2001] es una implementación experimental de tableau para $\mathcal{H}(@)$.

Creemos que hay fundamentalmente dos razones por las cuales este tipo de algoritmos han sido históricamente los más utilizados. Por un lado, la lógica modal básica tiene lo que se conoce como la *tree-model property*: para cada fórmula satisfacible existe algún modelo con forma de árbol tal que la fórmula es verdadera en la raíz del mismo. El método de tableau trata de encontrar un modelo con estas características. Por otro lado, es posible implementar los cálculos de tableaux dados arriba para que funcionen usando únicamente espacio polinomial, y satisfacibilidad de fórmulas en $\mathcal{M}$ es un problema completo para PSPACE. En general, el resto de las familias de algoritmos conocidos presentan peores órdenes teóricos.

De todas formas, cuando nos movemos a lógica híbrida, el algoritmo de tableau deja de presentar tantas ventajas. Para empezar, cuando incorporamos la posibilidad de referirnos a cualquier mundo (nombrable) del modelo, la tree-model property deja de valer; podemos escribir fórmulas que fuercen modelos arbitrarios, con ciclos y con componentes disconexas. Ya no podemos extender el modelo siempre "hacia adelante" como sucede con el tableau para $\mathcal{M}$. La presencia de igualdades, por ejemplo, nos obliga a mirar hacia atrás y reconsiderar lo que sabíamos hasta el momento cada vez que descubrimos que dos nominales son *iguales*.

El algoritmo de tableau para $\mathcal{H}(@)$ también puede implementarse en PSPACE, pero de cierta forma la complejidad de este problema es mayor. Podríamos verlo como que se encuentra casi en

el *límite* de esta clase. Como estamos hablando de complejidad de peor caso, en este contexto tiene sentido intentar otras alternativas, aun cuando su complejidad teórica sea más elevada.

2.3 Traducción a otros formalismos

Un enfoque que se ha explorado considerablemente consiste en traducir el problema de la validez de una fórmula modal a algún problema sobre otro formalismo para el cuál existe un método completo y consistente. Sin ir más lejos, utilizar un demostrador para DL como se sugirió en la sección anterior requiere, estrictamente hablando, un paso de traducción previo. En esta sección veremos otros ejemplos de métodos basados en traducción.

2.3.1 Lógica de primer orden

Ya vimos, en la Definición 1.4, una forma de traducir fórmulas de la lógica modal básica $\mathcal{M}$ en fórmulas de la lógica de primer orden. Esta traducción se puede extender fácilmente a $\mathcal{H}(@)$ utilizando constantes de primer orden para manejar los nominales. Basta con agregar las siguientes cláusulas:

$$\begin{array}{c|c} ST_x(i) = (x = i),\ i \in \mathsf{NOM} & ST_y(i) = (y = i),\ i \in \mathsf{NOM} \\ ST_x(@_i\,\varphi) = (\exists x)(x = i \wedge ST_x(\varphi)) & ST_y(@_i\,\varphi) = (\exists y)(y = i \wedge ST_y(\varphi)) \end{array}$$

Aunque teóricamente correcto, en la práctica, no ha dado buenos resultados utilizar las fórmulas resultantes de esta traducción directamente como entrada de un demostrador para lógica de primer orden. Como el demostrador no tiene en cuenta el origen modal de la fórmula de entrada, termina explorando espacios de solución innecesarios. En [Areces *et al.*, 2000] se propone utilizar una traducción a lógica de primer orden que incorpora información sobre la *profundidad modal* de cada diamante como una manera de *guiar* al demostrador.

La *traducción funcional* (*functional translation*) [Hustadt *et al.*, 1998] es una forma alternativa de traducir fórmulas modales a lógica de primer orden, con la que se han conseguido buenos resultados prácticos. La idea deriva del hecho de que un modelo de Kripke puede representarse alternativamente reemplazando cada símbolo de relación por un conjunto de funciones. Una traducción basada directamente en esta idea requeriría predicar sobre la existencia de ciertas funciones, lo cual no es un concepto expresable en lógica de primer orden. Sin embargo, se puede lograr un resultado equivalente utilizando una única función de dos parámetros para codificar cada conjunto de funciones; aquí el segundo parámetro se usa para *discriminar* entre las distintas funciones del conjunto original. MSPASS [Hustadt and Schmidt, 2000; MSPASS, 2004] es un demostrador que convierte fórmulas modales a lógica de primer orden usando la traducción funcional, y utiliza SPASS [Weidenbach *et al.*, 2002; SPASS, 2004] como demostrador.

Traducir a lógica de primer orden presenta la ventaja de que permite reutilizar décadas de investigación en mecanismos de demostración. En particular, permite construir con poco esfuerzo demostradores para lógicas modales. Sin embargo, la lógica de primer orden no es decidible. Para obtener métodos de decisión basados en la traducción a esta lógica es necesario ajustar los demostradores para que tengan en cuenta la decidibilidad del fragmento con el que están trabajando y garanticen terminación. De alguna forma, el proceso de traducción deja de ser transparente.

2.3.2 Autómatas

Algunos frames tienen la particularidad de que sus modelos se pueden expresar de forma tal que sean reconocibles por algún tipo de autómata. A modo de ejemplo, dado un modelo que es un orden lineal entre mundos, podemos armar una cinta en la que cada símbolo represente la conjunción de proposiciones que son verdaderas en un mundo determinado (si el conjunto de proposiciones es finito, también lo es el conjunto de símbolos). La posición relativa de cada mundo en el modelo

determina la posición de su símbolo asociado en la cinta. Estas cintas, a su vez, pueden ser la entrada de un autómata de estados finitos (o de un autómata de Büchi, cuando los modelos tienen infinitos mundos). De manera similar, los modelos con forma de árbol pueden expresarse de forma tal que sean procesables por un autómata de árboles de Büchi (tree-Büchi automata).

Al trabajar con uno de estos frames, podemos ver el conjunto de modelos que satisfacen una determinada fórmula como el lenguaje conformado por el conjunto de cadenas de símbolos asociadas. A partir de esta observación, se puede ver que es posible generar, dada una fórmula modal, un autómata apropiado que acepte únicamente las entradas que representan un modelo de la fórmula. Luego, determinar si una fórmula es satisfacible es equivalente a determinar si el lenguaje que acepta su autómata asociado es no-vacío.

A diferencia de lo que sucede con otros métodos de traducción, aquí la traducción es en sí el problema difícil ya que el número de estados del autómata es exponencial con respecto al tamaño de la fórmula; por el contrario, determinar si el autómata acepta un lenguaje vacío es polinomial. Lo que se investiga en este área es cómo generar autómatas con el menor número de estados posibles.

Este tipo de técnicas se usan generalmente para verificación de sistemas con requerimientos de tiempo, donde se utilizan lógicas temporales para especificar estos requerimientos. Este es un contexto muy apropiado para estas técnicas por dos razones: por una lado los modelos temporales son órdenes lineales, por el otro, si un sistema se modela utilizando un autómata, entonces es posible componer el autómata del sistema con el que representa sus requerimientos. En [Gerth *et al.*, 1995], por ejemplo, se presenta el algoritmo para convertir fórmulas de la lógica temporizada LTL a autómatas de Büchi.

Vale decir que este tema constituye toda un área de investigación en sí mismo. De ella han salido, incluso, sistemas de uso industrial como Spin [SPIN, 2004; Holzmann, 2003; Holzmann, 1997]. Probablemente, el trabajo en esta área sea, hasta la fecha, la aplicación más exitosa de las lógicas modales.

2.4 Resolución

Los métodos de tableaux que vimos en la Sección 2.2 están inspirados en el método de tableau para lógica de primer orden. Éste es uno de los métodos clásicos para determinar la satisfacibilidad de una fórmula de dicha lógica. No se trata, sin embargo, del método preferido para construir demostradores automáticos; éstos en general utilizan el método de *resolución*.

En esta sección repasaremos el cálculo de resolución para lógica de primer orden, veremos a continuación algunos cálculos basados en resolución propuestos para la lógica modal e introduciremos el *cálculo de resolución híbrido* (también llamado *cálculo de resolución directo*) presentado en [Areces *et al.*, 2001]. Este es el cálculo con el que trabajaremos en los próximos capítulos.

2.4.1 Resolución para lógica de primer orden

El *calculo de resolución* para lógica de primer orden fue propuesto por Robinson [Robinson, 1965] a mediados de la década del '60 y es hoy la base de la gran mayoría de los demostradores automáticos para esta lógica. Este cálculo trabaja con conjuntos de *literalcs* a los que llamamos *cláusulas*, cada una de ellas representa una disyunción. La totalidad de las cláusulas se interpreta como la conjunción de todas las disyunciones. Para obtener esta representación, las fórmulas de entrada del cálculo deben ser pasadas a forma prenexa, *skolemnizadas* y expresadas en forma normal disyuntiva.

El cálculo consiste, básicamente, en la saturación del conjunto de cláusulas por medio de una simple regla de inferencia, más una regla de factorización:

$$\text{(RES)} \quad \frac{C_1 \cup \{\varphi\} \quad C_2 \cup \{\neg\psi\}}{(C_1 \cup C_2)\sigma}$$

$$\text{(FACT)} \quad \frac{C \cup \{\varphi, \psi\}}{(C \cup \{\varphi\})\sigma}$$

donde σ es el unificador más general de los átomos φ y ψ. Ambas reglas se interpretan de la siguiente forma: si se encuentran cláusulas como las de la premisa, se debe agregar una cláusula como la que indica la conclusión.

Es fácil convencerse de que la *regla de resolución* es consistente: si suponemos que tanto $C_1 \cup \{\varphi\}$ como $C_2 \cup \{\neg\psi\}$ son verdaderas, y sabemos que $\varphi\sigma$ y $\neg\psi\sigma$ no pueden ser simultáneamente verdaderas (ya que $\varphi\sigma = \psi\sigma$), entonces debemos concluir que C_i es verdadera o C_2 lo es, con lo cual $C_1 \cup C_2$ también debe serlo.

La regla de factorización es necesaria para garantizar la completitud del cálculo en lógica de primer orden. No es necesaria, por ejemplo, en una versión proposicional del cálculo.

Sea N un conjunto de cláusulas, y sea N^* el conjunto que se obtiene al saturar N con las reglas de resolución y factorización; se puede demostrar que si N no es satisfacible (i.e. no existe un modelo que satisfaga todas las cláusulas), entonces N^* contiene una cláusula vacía [Bachmair and Ganzinger, 2001]. El cálculo de resolución como algoritmo para determinar la satisfacibilidad de una fórmula de la lógica de primer orden φ consiste, entonces, en armar el conjunto inicial de cláusulas asociadas a φ, obtener la saturación de dicho conjunto con respecto a la regla de resolución, y comprobar si éste contiene la cláusula vacía. En realidad, alcanza con parar en cuanto se genera la cláusula vacía.

Es interesante notar que así como el *peor caso* del método de tableau se da cuando la fórmula de entrada es insatisfacible (ya que es necesario cerrar todas las ramas del árbol), el peor caso del cálculo de resolución se presenta cuando la fórmula de entrada sí es satisfacible, ya que en este caso debemos obtener la saturación completa del conjunto de cláusulas.

2.4.2 Resolución para la lógica modal

Introducir un cálculo basado en resolución para la lógica modal trae aparejado algunas dificultades técnicas. Por un lado, la forma clausal que se utiliza en lógica de primer orden no es aplicable directamente al caso modal; por el otro es necesario introducir reglas adicionales para manejar adecuadamente las modalidades.

Se han propuesto varios sistemas basados en resolución con diferentes formas clausales para diversas lógicas modales. Podemos mencionar la de Mints [Mints, 1989] y la de Fariñas del Cerro [Fariñas del Cerro, 1982] con las correcciones que aparecen en [Enjalbert and del Cerro, 1989]. Esta última ha sido bastante estudiada; de hecho en [Auffray *et al.*, 1990] se analizan diversas optimizaciones y estrategias para casos particulares.

Sin embargo, el interés de estos cálculos es principalmente teórico. El cálculo de Fariñas del Cerro, por ejemplo, consta de un gran número de reglas, lo cual lo aleja del espíritu de "una simple regla" del cálculo original. Además, determinar en él si una subfórmula o un par de subfórmulas pueden ser premisas de una regla requiere una evaluación recursiva de las mismas, lo cual es computacionalmente costoso. El cálculo de Mints, por otro lado, si bien se basa en un conjunto reducido de reglas, trabaja comparando fórmulas que sólo difieren en una subfórmula. Para que el cálculo sea completo necesita unas reglas auxiliares que permiten *debilitar* una fórmula; determinar cuándo aplicar estas reglas no es trivial. En ambos casos, si bien el cálculo puede implementarse, no pueden competir en la práctica con ninguna solución basada en tableau.

2.4.3 El cálculo de resolución híbrido

En [Areces *et al.*, 2001] se propone un cálculo basado en resolución que puede ser usado sobre $\mathcal{H}(@)$ y, en una versión simplificada, sobre $\mathcal{M}$. En cierta forma, entre este cálculo y las anteriores propuestas existe una diferencia similar a la que encontramos entre el tableau implícito y el

etiquetado que introdujimos en las Secciones 2.2.1 y 2.2.2: la nueva expresividad provista por los operadores híbridos permite obtener un conjunto de reglas simple y elegante.

La versión que aquí presentaremos del cálculo trabaja con fórmulas en *forma normal negativa* (*negative normal form*), es decir, sólo es posible aplicar el operador de negación sobre átomos. Esto significa que tanto $\vee$ como $\Box$ serán símbolos *primitivos*.

Definición 2.1. Dados PROP un conjunto numerable de símbolos de proposición, NOM un conjunto numerable de nominales y REL un conjunto numerable de símbolos de relación, decimos que ATOM = PROP $\cup$ NOM y definimos inductivamente el conjunto $\mathcal{H}^{NNF}(@)$ definido sobre ATOM y REL como:

$$\mathcal{H}^{NNF}(@) ::= a \mid \neg a \mid \varphi \vee \varphi' \mid \varphi \wedge \varphi' \mid \langle r \rangle \varphi \mid [r]\varphi \mid @_i\, \varphi$$

donde $a \in$ ATOM, $r \in$ REL, $i \in$ NOM y $\varphi, \varphi' \in \mathcal{H}^{NNF}(@)$.

Al igual que el cálculo de resolución para lógica de primer orden, el cálculo de resolución híbrido trabaja con conjuntos de *cláusulas*. Una cláusula, en este contexto, es un conjunto de fórmulas-@ arbitrarias pertenecientes a $\mathcal{H}^{NNF}(@)$. Aquí también, una cláusula representa una disyunción, pero a diferencia de lo que sucede en los cálculos de resolución para $\mathcal{M}$ mencionados, no hay ninguna restricción en cuanto a la forma clausal en que se deben encontrar las fórmulas; son las propias reglas del cálculo las que se encargan de armar cláusulas con fórmulas cada vez más simples.

Dada una fórmula $\varphi \in \mathcal{H}^{NNF}(@)$, llamamos $\mathit{ClSet}(\varphi) = \{\{@_t\, \varphi\}\}$, donde t es un nominal que no aparece en φ. Claramente, determinar la satisfacibilidad de φ es equivalente a determinar la de $@_t\, \varphi$. Podemos ahora definir $\mathit{ClSet}^*(\varphi)$ – el conjunto saturado de cláusulas correspondiente a φ – como el menor conjunto que incluye a $\mathit{ClSet}(\varphi)$ y que está clausurado bajo las reglas de la Figura 2.1.

$$(\wedge)\quad \frac{Cl \cup \{@_t\,(\varphi_1 \wedge \varphi_2)\}}{\begin{array}{c} Cl \cup \{@_t\, \varphi_1\} \\ Cl \cup \{@_t\, \varphi_2\} \end{array}} \qquad (\vee)\quad \frac{Cl \cup \{@_t\,(\varphi_1 \vee \varphi_2)\}}{Cl \cup \{@_t\, \varphi_1, @_t\, \varphi_2\}}$$

$$(\text{RES})\quad \frac{Cl_1 \cup \{@_t\, \varphi\} \quad Cl_2 \cup \{@_t\, \neg\varphi\}}{Cl_1 \cup Cl_2}$$

$$([R])\quad \frac{Cl_1 \cup \{@_t\, [r]\varphi\} \quad Cl_2 \cup \{@_t\, \langle r \rangle s\}}{Cl_1 \cup Cl_2 \cup \{@_s\, \varphi\}} \qquad (\langle R \rangle)\quad \frac{Cl \cup \{@_t\, \langle r \rangle \varphi\}}{\begin{array}{c} Cl \cup \{@_t\, \langle r \rangle n\} \\ Cl \cup \{@_n\, \varphi\} \end{array}} \text{ para un } n \text{ nuevo}$$

$$(@)\quad \frac{Cl \cup \{@_t\, @_s\, \varphi\}}{Cl \cup \{@_s\, \varphi\}}$$

$$(\text{SYM})\quad \frac{Cl \cup \{@_t\, s\}}{Cl \cup \{@_s\, t\}} \qquad (\text{REF})\quad \frac{Cl \cup \{@_t\, \neg t\}}{Cl} \qquad (\text{PARAM})\quad \frac{Cl_1 \cup \{@_t\, s\} \quad Cl_2 \cup \{\varphi(s)\}}{Cl_1 \cup Cl_2 \cup \{\varphi(s/t)\}}$$

Figura 2.1: Reglas de resolución para $\mathcal{H}(@)$

Podemos agrupar las reglas del cálculo de acuerdo a la función que cumplen. Por ejemplo, las reglas ($\wedge$), ($\vee$) y (@) se encargan de simplificar las fórmulas. La regla ($\langle R \rangle$) también simplifica en cierta forma un fórmula; podemos pensarla como una suerte de skolemnización mediante la cual se le asigna explícitamente un nombre a un mundo. La regla (RES) es el equivalente de la regla de resolución para lógica de primer orden, mientras que la regla ($[R]$) se encarga de propagar información entre distintos contextos. Implícitamente, esta regla está codificando una unificación no trivial y un paso de resolución. Finalmente, las reglas (SYM), (REF) y (PARAM) se corresponden con el *paquete* estándar de reglas para manejar resolución en lógica de primer orden con igualdad [Bachmair and Ganzinger, 1998].

La construcción de $ClSet^*(\varphi)$ es en sí misma un algoritmo correcto y completo para verificar la satisfacibilidad de fórmulas de $\mathcal{H}^{NNF}(@)$: φ es insatisfacible si y sólo si la cláusula vacía $\{\}$ es un elemento de $ClSet^*(\varphi)$. Sin embargo, como $ClSet^*(\varphi)$ puede ser un conjunto infinito, existen fórmulas cuya satisfacibilidad no puede determinarse en finitos pasos. Cómo convertir este cálculo en un método de decisión para $\mathcal{H}(@)$ será el tema del Capítulo 4.

2.4.4 Resolución con orden y selección

El cálculo de resolución para lógica de primer orden, tal como fue presentado en la Sección 2.4.1, es refutacionalmente completo; es decir, se puede mostrar que dado un conjunto insatisfacible de cláusulas, el conjunto que se obtiene al saturarlo respecto a la regla de resolución contiene la cláusula vacía [Bachmair and Ganzinger, 2001]. A pesar de ello, no es viable construir demostradores en base a la aplicación directa de estas reglas. El problema surge de la combinación de dos hechos fácilmente verificables:

1. el cálculo permite derivar cláusulas *redundantes*, y
2. dado un conjunto de cláusulas C, si decimos que $d(C)$ representa el conjunto de todas las cláusulas que se pueden derivar de la aplicación de la regla de resolución exclusivamente sobre cláusulas que aparecen en C, entonces el tamaño de $d(C)$ puede crecer de manera *exponencial* conforme aumenta el número de cláusulas de C.

Examinemos un ejemplo simple de esto.

Ejemplo 3. Supongamos que N es un conjunto satisfacible de cláusulas formado por:

$$\begin{aligned} C_1 &= \{\neg P(x), \neg S(x)\} \\ C_2 &= \{\neg P(x), S(x)\} \\ C_3 &= \{P(x), Q(x), \neg R(x)\} \\ C_4 &= \{\neg Q(x), S(x)\} \end{aligned}$$

El siguiente es parte del conjunto de cláusulas que el demostrador debería generar a partir de N:

$$\begin{aligned} C_5 &= \{\neg P(x)\} && (C_2, C_1) \\ C_6 &= \{Q(x), \neg R(x), \neg S(x)\} && (C_3, C_1) \\ C_7 &= \{Q(x), \neg R(x), S(x)\} && (C_3, C_2) \\ C_8 &= \{\neg P(x), \neg Q(x)\} && (C_4, C_1) \\ C_9 &= \{P(x), \neg R(x), S(x)\} && (C_3, C_4) \\ C_{10} &= \{Q(x), \neg R(x)\} && (C_5, C_3) \\ C_{11} &= \{\neg P(x), Q(x), \neg R(x)\} && (C_2, C_6) \\ C_{12} &= \{\neg R(x), S(x), \neg S(x)\} && (C_6, C_4) \\ C_{13} &= \{P(x), \neg Q(x), \neg R(x)\} && (C_4, C_6) \\ C_{14} &= C_{11} && (C_7, C_1) \\ C_{15} &= C_{12} && (C_7, C_4) \\ C_{16} &= C_{10} && (C_7, C_6) \\ C_{17} &= C_{13} && (C_3, C_8) \\ C_{18} &= \{P(x), \neg P(x), \neg R(x)\} && (C_3, C_8) \\ C_{19} &= \{\neg P(x), \neg R(x), \neg S(x)\} && (C_6, C_8) \\ C_{20} &= \{\neg P(x), \neg R(x), S(x)\} && (C_7, C_8) \\ C_{21} &= C_{12} && (C_9, C_1) \\ C_{22} &= C_{18} && (C_9, C_1) \\ C_{23} &= \{\neg R(x), \neg S(x)\} && (C_9, C_2) \\ C_{24} &= C_{23} && (C_9, C_5) \\ C_{25} &= C_3 && (C_9, C_6) \\ C_{26} &= \{\neg Q(x), \neg R(x), S(x)\} && (C_9, C_8) \\ C_{27} &= C_{23} && (C_{10}, C_4) \end{aligned}$$

$$\vdots$$

Observemos que C_3 *resuelve* con C_1 eliminando $Q(x)$ para generar C_4 y con C_2 para generar C_5 eliminando $P(x)$. Esta última cláusula es *redundante*. ¿Por qué? El objetivo de un demostrador basado en resolución es lograr derivar la cláusula vacía; si suponemos que C_3 puede participar en una derivación de la cláusula vacía, concluiremos que debemos ser capaces de *eliminar* todos los literales que aparecen en C_3. Ahora bien, en C_4 ya habíamos logrado eliminar $Q(x)$, con lo cual *conviene* eliminar $P(x)$ de esta cláusula como se hace al generar C_7; generar C_5 (ó C_6) resulta redundante. Se puede ver con claridad en este ejemplo cómo las cláusulas redundantes pueden contribuir al rápido crecimiento del conjunto que el demostrador debe manejar.

No sería una solución válida pedir que cada cláusula pueda ser premisa de la regla de resolución a lo sumo una vez; se puede ver fácilmente con un ejemplo que no obtendríamos un sistema completo. Sin embargo, sí es posible establecer ciertas condiciones bajo las cuáles es posible *elegir* un literal de cada cláusula de forma tal que sólo se acepte que la cláusula sea premisa de la regla si es para *eliminar* dicho literal. Este es, en esencia, el mecanismo estándar para controlar la generación de cláusulas en resolución para lógica de primer orden, llamado *resolución con orden y selección* [Bachmair and Ganzinger, 2001].

En el contexto de resolución para lógica de primer orden, decimos que una *función de selección* es una función que asigna a cada cláusula C un conjunto vacío o un conjunto unitario con un literal negativo de C. Dada una función de selección S y cierta relación de orden entre literales $\succ$, el cálculo de resolución para lógica de primer orden con orden y selección se define mediante la regla

$$\text{(RES–OS)} \quad \frac{C_1 \cup \{\varphi\} \quad C_2 \cup \{\neg\psi\}}{(C_1 \cup C_2)\sigma}$$

tal que

1. σ es el unificador más general de los átomos φ y ψ
2. $S(C_1 \cup \{\varphi\}) = \{\}$ y $\varphi\sigma \succ \varphi'$ para todo $\varphi' \in C_1\sigma$
3. $S(C_2 \cup \{\neg\psi\}) = \{\neg\psi\}$ o bien $S(C_2 \cup \{\neg\psi\}) = \{\}$ y $\neg\psi\sigma \succ \psi'$ para todo $\psi' \in C_2\sigma$.

Se puede demostrar que el cálculo de resolución con orden y selección (que incluye la regla de factorización) es refutacionalmente completo para la lógica de primer orden [Bachmair and Ganzinger, 2001].

Ejemplo 4. A modo de ejemplo, veamos en cuántos pasos llegamos a un conjunto saturado partiendo del mismo conjunto inicial del Ejemplo 3, suponiendo $\neg S(x) \succ S(x) \succ \neg R(x) \succ R(x) \succ \neg Q(x) \succ Q(x) \succ \neg P(x) \succ P(x)$ y una función de selección tal que $S(C) = \{\}$ para todo C. En todos los casos, la fórmula maximal de la cláusula es la que aparece a la izquierda.

$$\begin{array}{lll} C_1 = \{\neg P(x), \neg S(x)\} & & \\ C_2 = \{\neg P(x), S(x)\} & & \\ C_3 = \{P(x), Q(x), \neg R(x)\} & & \\ C_4 = \{\neg Q(x), S(x)\} & & \\ C_5' = \{\neg P(x)\} & & (C_2, C_1) \\ C_6' = \{\neg P(x), \neg Q(x)\} & & (C_4, C_1) \end{array}$$

El Ejemplo 4 muestra en forma elocuente la importancia de contar con un mecanismo de regulación basado en orden y selección si se desea construir un demostrador basado en algún cálculo de resolución. El cálculo de resolución híbrida presentado en la Figura 2.1 no escapa a este problema. Sin embargo, hasta el momento no se contaba con un mecanismo apropiado de orden y selección para el mismo.

En el Capítulo 3 proponemos una forma de incorporar orden y selección al cálculo de resolución híbrida y demostramos que con esta extensión se conserva la completitud refutacional.

Capítulo 3

Resolución híbrida con orden y selección

En este capítulo intentamos trasladar al cálculo de la Figura 2.1 las estrategias de orden y selección que fueron presentadas, para el caso de resolución para lógica de primer orden, en la Sección 2.4.4. Comenzaremos, en la Sección 3.1, por introducir formalmente la idea de *orden entre fórmulas* y repasar algunos órdenes, comunes en la literatura, que utilizaremos a lo largo del capítulo. En la Sección 3.2 presentamos un tipo especial de orden para fórmulas de $\mathcal{H}^{NNF}(@)$, el cual permitirá garantizar que el cálculo de resolución con orden y selección que proponemos en 3.4.1 es refutacionalmente completo. Esto lo probamos en la Sección 3.4.2, con una demostración similar a la estándar para el caso de resolución para lógica de primer de orden [Bachmair and Ganzinger, 2001]. Ésta se basa en la utilización de modelos de Herbrand; por ello, previamente, en la Sección 3.3 presentamos una versión para la lógica $\mathcal{H}(@)$ del Teorema de Herbrand.

3.1 Relaciones de orden entre fórmulas

Se llama *orden parcial estricto* (o simplemente *orden*) a cualquier relación binaria, transitiva e irreflexiva. En lo que sigue usaremos $\succ$ (posiblemente con subíndices) para hablar de un orden parcial estricto y $\succeq$ para referirnos a su clausura reflexiva. Además, diremos que un orden $\succ$ es total siempre que para cualquier par de elementos distintos x e y del dominio de $\succ$, se cumpla $x \succ y$ o $y \succ x$.

Definición 3.1. Un orden $\succ$ se encuentra *bien fundado* (*well-founded*) si y sólo si en todo subconjunto C del dominio de $\succ$ existe un x tal que ningún $y \in C$ cumple $x \succ y$.

En otras palabras, cualquier subconjunto de elementos debe tener al menos un elemento *minimal* (en el sentido de que no existe otro elemento en el subconjunto que sea *menor* que él). Esta definición implica la inexistencia de cadenas descendentes infinitas del estilo $x_1 \succ x_2 \succ \ldots$ Como corolario de la definición de buena fundación, si $\succ$ es además un orden total, entonces todo subconjunto (no necesariamente propio) del dominio tiene un único mínimo. Si tenemos un orden bien fundado sobre un conjunto S, podemos aplicar *inducción Noetheriana* (llamada también *inducción global*) para probar que una propiedad P vale sobre todos los elementos de S.

En particular, nos interesará trabajar con órdenes entre fórmulas de alguna lógica dada. Entendemos que una *fórmula* es una cadena finita y bien formada, obtenida por la composición de un conjunto de operadores, cada uno con una aridad fija. Es de esperar que uno o más de estos operadores tengan aridad 0. La definición formal es la siguiente:

Definición 3.2. Dado un conjunto numerable $O = \{o_1, \ldots, o_n\}$ de operadores asumimos que está definida una función *aridad* : $O \to \mathbb{N}$. El conjunto F_O de *formulas posibles* sobre O se define entonces recursivamente como el mínimo conjunto tal que:

1. $o_i \in F_O$ si $aridad(o_i) = 0$.

2. $o_i(\varphi_1 \ldots \varphi_n) \in F_O$ si $\varphi_1, \ldots, \varphi_n \in F_O$, $o_i \in O$ y $aridad(o_i) = n$.

Dado un conjunto O de operadores, una lógica dada definirá el conjunto de sus fórmulas especificando $\mathcal{L} \subseteq F_O$. Dado $\varphi \in F_O$, $S_f(\varphi)$ es el conjunto de subfórmulas de φ definido como

1. Si $\varphi = o_i$ entonces $S_f(\varphi) = \{o_i\}$.

2. Si $\varphi = o_i(\varphi_1 \ldots \varphi_n)$ entonces $S_f(\varphi) = \{\varphi\} \cup S_f(\varphi_1) \cup \ldots \cup S_f(\varphi_n)$.

Dados un conjunto de fórmulas $\mathcal{L} \subseteq F_O$ y dos fórmulas $\varphi_1, \varphi_2 \in \mathcal{L}$, decimos que φ_1 es una subfórmula de φ_2 (en $\mathcal{L}$) si $\varphi_1 \in S_f(\varphi_2)$; y si además $\varphi_1 \neq \varphi_2$ decimos que φ_1 es una subfórmula propia de φ_2 (en $\mathcal{L}$).

Existen algunas familias de órdenes entre fórmulas que son clásicas en la literatura [Bachmair and Ganzinger, 2001; Dershowitz and Jouannaud, 1990].

Definición 3.3. Dado un conjunto de fórmulas $\mathcal{L}$, decimos que un orden $\succ$ tiene la *propiedad de subfórmulas* (*subformula property*) sobre $\mathcal{L}$ si $\varphi \succ \varphi'$ para todo par de fórmulas $\varphi, \varphi' \in \mathcal{L}$, tales que φ' sea subfórmula propia de φ.

Definición 3.4. Dado un conjunto de fórmulas $\mathcal{L}$, decimos que un orden $\succ$ es un *orden de reescritura* (*rewrite ordering*) para $\mathcal{L}$ si se cumple que, para todo $\psi_1, \psi_2, \varphi \in \mathcal{L}$, $\psi_1 \succ \psi_2$ implica $\varphi[\psi_1]_p \succ \varphi[\psi_2]_p$ (donde $\varphi[\psi_i]_p$ significa que la subfórmula de φ en posición p se reemplaza por ψ_i, ver [Dershowitz and Jouannaud, 1990] para la definición estándar de *posición*).

Definición 3.5. Se llama *orden de reducción* (*reduction ordering*) a todo orden de reescritura bien fundado. Si además posee la propiedad de subfórmulas se dice que es un *orden de simplificación* (*simplification ordering*).

Nos resta ver cómo se pueden construir efectivamente relaciones que tengan estas propiedades. Existe toda una familia de ordenes entre fórmulas que se pueden construir a partir de un orden parcial entre los símbolos en base a los cuáles éstas se constituyen. Entre los ejemplos más conocidos podemos citar el *multiset path ordering* y el *lexicographic path ordering* (ver [Dershowitz and Jouannaud, 1990]). A continuación, discutiremos este último tipo de orden, según la definición dada en [Bachmair and Ganzinger, 2001].

Definición 3.6 (Lexicographic Path Ordering). Sea O un conjunto de operadores y sea $\succ$ un orden entre los elementos de O al que llamaremos *precedencia* (precedence). El *lexicographic path ordering* (lpo) entre fórmulas de $\mathcal{L}$ se nota $\succ_{lpo}$ y se define como sigue.

Sean $\varphi = o_1(\varphi_1 \ldots \varphi_m)$ y $\psi = o_2(\psi_1 \ldots \psi_n)$, $n, m \geq 0$; diremos que $\varphi \succ_{lpo} \psi$ si y sólo si:

1. $o_1 \succ o_2$ y $\varphi \succ_{lpo} \psi_i$, para todo i tal que $1 \leq i \leq n$; ó

2. $o_1 = o_2$ y para algún j se cumple, $(\varphi_1 \ldots \varphi_{j-1}) = (\psi_1 \ldots \psi_{j-1})$, $\varphi_j \succ_{lpo} \psi_j$ y $\varphi \succ_{lpo} \psi_k$ para todo k tal que $j \leq k \leq n$; ó

3. $\varphi_j \succeq_{lpo} \psi$ para algún j tal que $1 \leq j \leq m$.

Se puede demostrar que si la precedencia se encuentra bien fundada, entonces $\succ_{lpo}$ es un orden de simplificación. Además, es total cuando la precedencia también lo es (ver, por ejemplo, [Dershowitz and Jouannaud, 1990]). Para terminar, veamos una manera usual de usar cualquier orden entre fórmulas para construir un orden entre cláusulas (conjuntos finitos de fórmulas).

Definición 3.7. Dado $\succ$ un orden entre fórmulas, se define $\succ_{Cl}$, su extensión para cláusulas, como la relación que cumple $C_1 \succ_{Cl} C_2$ si y sólo si:

1. $C_1 \not\subseteq C_2$; y

2. para toda φ_2 tal que $\varphi_2 \in C_2$ y $\varphi_2 \notin C_1$ existe $\varphi_1 \in C_1$ tal que $\varphi_1 \notin C_2$ y $\varphi_1 \succ \varphi_2$.

A modo de ejemplo, si $\varphi_1 \succ \varphi_2 \succ \varphi_3$, entonces $\{\varphi_1, \varphi_2\} \succ_{Cl} \{\varphi_1, \varphi_3\} \succ_{Cl} \{\varphi_1\} \succ_{Cl} \{\varphi_2, \varphi_3\}$. Cuando $\succ$ es total y bien fundado también $\succ_{Cl}$ lo es. Es fácil comprobar que si $C = C' \cup \{\varphi\}$, entonces se cumplirá $C \succ_{Cl} C'$ y $C \succ_{Cl} C' \cup \{\psi\}$ toda vez que $\varphi \succ \psi$. En particular, cualquier cláusula no vacía C verifica $C \succ_{Cl} \{\}$. Por otro lado, si $\succ$ es total, entonces en cada cláusula existirá una fórmula maximal. Luego, si $C = C' \cup \{\varphi\}$ y $D = D' \cup \{\psi\}$, con φ y ψ maximales en sus respectivas cláusulas y $\varphi \succ \psi$, tenemos $C \succ_{Cl} D$ y $\{\varphi\} \succ_{Cl} C' \cup D'$. Este tipo de propiedades serán muy importantes más adelante.

De aquí en más, salvo que se indique lo contrario, usaremos el mismo símbolo para una relación de orden entre fórmulas y la extensión para cláusulas de dicho orden.

3.2 Un orden *adecuado* para $\mathcal{H}(@)$

En el contexto de los sistemas de resolución se llama *adecuado* (*admissible*) a un orden entre fórmulas que garantiza que un cálculo con orden y selección conserva la completitud refutacional. Aquí presentaremos un tipo de orden sobre $\mathcal{H}(@)$ que, según veremos más adelante, nos permitirá definir un cálculo completo con orden y selección basado en el cálculo híbrido.

De aquí en más, salvo que se aclare lo contrario, trabajaremos únicamente con fórmulas bien formadas de $\mathcal{H}^{NNF}(@)$. En este contexto nos resultará conveniente considerar a @, $\langle\rangle$ y [] como operadores *binarios*:

$$\begin{array}{ll} @(\cdot,\cdot): & \mathcal{H}^{NNF}(@) \times \mathsf{NOM} \to \mathcal{H}^{NNF}(@)^1 \\ \langle\rangle(\cdot,\cdot): & \mathsf{REL} \times \mathcal{H}^{NNF}(@) \to \mathcal{H}^{NNF}(@) \\ [](\cdot,\cdot): & \mathsf{REL} \times \mathcal{H}^{NNF}(@) \to \mathcal{H}^{NNF}(@) \end{array}$$

por más que usemos la notación estándar $@_n\,\varphi$, $\langle r\rangle\varphi$ y $[r]\varphi$. Siguiendo la Definición 3.2, $\mathcal{H}^{NNF}(@)$ es un subconjunto propio del conjunto de fórmulas posibles sobre $O = \mathsf{PROP} \cup \mathsf{NOM} \cup \mathsf{REL} \cup \{\neg, \wedge, \vee, @, \langle\rangle, []\}$.

Definición 3.8. Un orden $\succ$ es adecuado si cumple simultáneamente con estas condiciones, para todo $\varphi, \psi \in \mathcal{H}^{NNF}(@)$:

- A1) es un orden de simplificación total
- A2) $\varphi \succ i$ para todo $\varphi \notin \mathsf{NOM}$ y todo $i \in \mathsf{NOM}$
- A3) si $\varphi \succ \psi$, entonces $@_i\,\varphi \succ @_j\,\psi$, para todo $i, j \in \mathsf{NOM}$
- A4) si $\langle r\rangle i$ es una subfórmula propia de φ con $i \in \mathsf{NOM}$, entonces $\varphi \succ \langle r\rangle j$ para todo nominal j
- A5) $[r]i \succ \langle r\rangle j$, para todo $i, j \in \mathsf{NOM}$

La condición A2 pide que un nominal sea menor que cualquier fórmula que no sea otro nominal, A3 pide que el operador @ no afecte al orden entre fórmulas, A4 introduce una noción muy débil de *complejidad estructural* y A5 prioriza a [] por sobre $\langle\rangle$.

Es importante observar que si $\succ$ cumple con las condiciones de la Definición 3.8, entonces $i \succ j$ si y sólo si $@_j\,i \succ @_i\,j$ para todo par de nominales i y j. Esto significa que un orden adecuado ordena en forma coherente las igualdades. De aquí en más si $i \succ j$, diremos que $@_i\,j$ es una *igualdad normalizada*.

Nos resta probar que las condiciones de la Definición 3.8 no son demasiado restrictivas. Para ello construiremos un orden concreto que cumpla con las condiciones pedidas. Nos basaremos en el orden presentado en la Definición 3.6, aplicado al conjunto F_O de fórmulas posibles sobre $O = \mathsf{PROP} \cup \mathsf{NOM} \cup \mathsf{REL} \cup \{\neg, \wedge, \vee, @, \langle\rangle, []\}$ (notar que $\mathcal{H}^{NNF}(@) \subset F_O$).

[1] El orden de los parámetros de este operador ha sido cuidadosamente elegido para simplificar la Definición 3.9 y la subsecuente demostración de la Proposición 1.

Definición 3.9. Dada una signatura $\langle \mathsf{PROP}, \mathsf{NOM}, \mathsf{REL} \rangle$ con $\mathsf{PROP} = \{P_i \mid i \in \mathbb{N}\}$, $\mathsf{NOM} = \{N_i \mid i \in \mathbb{N}\}$ y $\mathsf{REL} = \{R_i \mid i \in \mathbb{N}\}$, definimos la relación $> \subseteq O \times O$ como la clausura transitiva del conjunto

$$\{(@, \neg), (\neg, \wedge), (\wedge, \vee), (\vee, []), ([], \langle\rangle)\}$$
$$\cup$$
$$\{(\langle\rangle, R_i), (R_i, P_j), (P_j, N_k) \mid i, j, k \in \mathbb{N}\}$$
$$\cup$$
$$\{(R_i, R_j), (P_i, P_j), (N_i, N_j) \mid i > j\}$$

Por definición, $>$ es total, irreflexiva y está bien fundada. Sea $\succ_{lpo}$ el orden lexicográfico sobre F_O que usa $>$ como precedencia. De acuerdo a lo que hemos visto, $\succ_{lpo}$ es un orden de simplificación total. Finalmente, definimos $\succ_h$ como el orden que verifica $\varphi \succ_h \psi$ si y sólo si

1. $size(\varphi) > size(\psi)$, ó
2. $size(\varphi) = size(\psi)$ y $\varphi \succ_{lpo} \psi$

donde $size(\varphi)$ es la complejidad (en cuanto a cantidad de operadores) de φ.

Proposición 1. *$\succ_h$ es un orden adecuado.*

Demostración. Tenemos que ver que $\succ_h$ satisface las condiciones A1 a A5.
A1) Es evidente que $\succ_h$ es total y tiene la propiedad de subfórmulas. Sólo debemos ver, entonces, que es además un orden de reescritura; es decir que $\psi_1 \succ_h \psi_2$ implica $\varphi[\psi_1]_p \succ_h \varphi[\psi_2]_p$. Cuando $size(\psi_1) > size(\psi_2)$ esto es directo de ver. Veamos en cambio qué sucede si $size(\psi_1) = size(\psi_2)$ y $\psi_1 \succ_{lpo} \psi_2$. Como tenemos $size(\varphi[\psi_1]_p) = size(\varphi[\psi_2]_p)$, nos alcanza con comprobar que $\varphi[\psi_1]_p \succ_{lpo} \varphi[\psi_2]_p$, lo cual se cumple por ser $\succ_{lpo}$ un orden de reescritura.

A2) Queremos ver que si $\varphi \notin \mathsf{NOM}$, $\varphi \succ_h i$ para todo $i \in \mathsf{NOM}$. Si $size(\varphi) > 1$, esto está garantizado. Si no, debe suceder $\varphi \in \mathsf{PROP}$ y entonces la precedencia garantiza $\varphi \succ_{lpo} i$.

A3) Supongamos $\varphi \succ_h \psi$; queremos ver que para todo $i, j \in \mathsf{NOM}$, j se cumple $@_i \varphi \succ_h @_j \psi$. Si $size(\varphi) > size(\psi)$ entonces $size(@_i \varphi) > size(@_j \psi)$ y vemos que se cumple. Supongamos, en cambio, que $size(\varphi) = size(\psi)$ y $\varphi \succ_{lpo} \psi$. Lo que debemos comprobar en este caso es que $@_i \varphi \succ_{lpo} @_j \psi$, o lo que es lo mismo, que $@(\varphi, i) \succ_{lpo} @(\psi, j)$. Esto es directo de ver aplicando el segundo caso de la Definición 3.6 y teniendo en cuenta que la precedencia garantiza $@ \succ_{lpo} j$.

A4) Queremos ver que si $\langle r \rangle i$ es una subfórmula propia de φ entonces $\varphi \succ_h \langle r \rangle j$ para cualquier par de nominales i, j. Ahora bien, por ser una subfórmula propia, $size(\varphi) > size(\langle r \rangle i)$; como además $size(\langle r \rangle i) = size(\langle r \rangle j)$, concluimos que $size(\varphi) > size(\langle r \rangle j)$ y, por lo tanto $\varphi \succ_h \langle r \rangle j$.

A5) Mostramos que para todo $i, j \in \mathsf{NOM}$, $[](r, i) \succ_{lpo} \langle\rangle(r, j)$. Como la precedencia garantiza $[] \succ_{lpo} \langle\rangle$, basta ver que $[](r, i) \succ_{lpo} r$ y $[](r, i) \succ_{lpo} j$. La primera condición se sigue de la propiedad de subfórmulas de $\succ_{lpo}$, la segunda de la condición $[] \succ_{lpo} j$ sobre la relación de precedencia. □

Es interesante observar que no es posible construir un orden adecuado usando *únicamente* lpo. Basta con ver que no se puede encontrar una manera de garantizar $\langle\rangle(r', \langle\rangle(r, i)) \succ_{lpo} \langle\rangle(r, j)$ cuando $r \succ_{lpo} r'$ y $j \succ_{lpo} i$, lo cual viola A4.

En lo que resta, usaremos $\succ$ para referirnos a algún orden adecuado salvo que se indique específicamente lo contrario.

3.3 Un Teorema de Herbrand $\mathcal{H}(@)$

Dado una signatura de primer orden (sin igualdad) $\mathcal{S} = \langle \mathsf{FUNC}, \mathsf{REL} \rangle$ un *modelo de Herbrand* [Herbrand, 1930] para $\mathcal{S}$ se define como la interpretación $I = \langle D^I, \cdot^I \rangle$ donde

$$\begin{array}{rcl} D^I & = & \text{el conjunto de términos de } \mathcal{S} \\ t^I & = & t, \text{ para un término } t \\ R^I & \subseteq & D^{I^n} \text{ para todo símbolo de relación } R \text{ de aridad } n. \end{array}$$

Es decir, todos los modelos de Herbrand tienen el mismo dominio e interpretan los términos de la misma manera; la única variación está en la forma en que interpretan los símbolos de REL. Notar que podemos identificar un modelo de Herbrand con el conjunto de literales positivos que son verdaderos en el modelo. La importancia de los modelos de Herbrand está dada por el siguiente teorema:

Teorema 2 (Teorema de Herbrand). *Una teoría de primer orden T sobre la signatura $S = \langle FUNC, REL\rangle$ tiene un modelo si y sólo si tiene un modelo de Herbrand sobre la signatura $S' = \langle FUNC \cup FUNC', REL\rangle$, donde $FUNC'$ es un conjunto infinito numerable disjunto de FUNC.*

El Teorema 2 asegura que para determinar la satisfacibilidad de una teoría de primer orden, alcanza con considerar solamente los posibles modelos de Herbrand (formulaciones alternativas de este teorema en [Buss, 1995]). Para extender este teorema a la lógica $\mathcal{H}(@)$ empezamos por definir la noción apropiada de literal positivo.

Definición 3.10. Dada una signatura híbrida $S = \langle \mathsf{PROP}, \mathsf{NOM}, \mathsf{REL}\rangle$, definimos LIT-P, el conjunto de literales positivos de $\mathcal{H}(@)$ sobre S como:

$$\mathsf{LIT\text{-}P} ::= @_i\, j \mid @_i\, p \mid @_i \langle r\rangle j$$

donde $i, j \in \mathsf{NOM}$, $p \in \mathsf{PROP}$ y $r \in \mathsf{REL}$.

Definición 3.11. Sea $\mathcal{S} = \langle \mathsf{PROP}, \mathsf{NOM}, \mathsf{REL}\rangle$ una signatura híbrida dada. Un *modelo de Herbrand híbrido* para $\mathcal{H}(@)$ en S es un conjunto $I \subset \mathsf{LIT\text{-}P}$.

Como anteriormente, identificamos un modelo de Herbrand con un conjunto de literales positivos que el modelo satisface y que define unívocamente un determinado modelo híbrido. Dado I un modelo de Herbrand, sea $\sim_I$ (la *relación de equivalencia inducida por* I) la mínima relación de equivalencia que extiende el conjunto $\{(i,j) \mid @_i\, j \in I\} \cup \{(i,i) \mid i \in \mathsf{NOM}\}$. Definimos ahora el modelo híbrido unívocamente determinado por I como la estructura $\langle W^I, \{R_i^I\}, V^I\rangle$, donde

$$\begin{array}{lcl} W^I & = & \mathsf{NOM}/_{\sim_I} \\ R_i^I & = & \{([j],[k]) \mid @_j \langle R_i\rangle k \in I\} \\ V^I(p) & = & \{[j] \mid @_j\, p \in I\}, p \in \mathsf{PROP} \\ V^I(i) & = & \{[j]\}, j \in \mathsf{NOM}. \end{array}$$

donde $\mathsf{NOM}/_{\sim_I}$ es el conjunto de clases de equivalencia definido por $\sim_I$ sobre NOM y $[i]$ es la clase de equivalencia asociada a i por $\sim_I$. De ahora en más no diferenciaremos entre un modelo de Herbrand híbrido I y su modelo asociado; y diremos, por ejemplo, que una fórmula $@_i\varphi$ es válida en I, cuando el modelo asociado la satisfaga (siempre nos referiremos a formulas de la forma $@_i\varphi$ para que no sea necesaria la referencia a un punto de evaluación en el modelo).

El modelo asociado a I es un poco más complejo que el que definimos para una signatura de primer orden porque $\mathcal{H}(@)$ contiene igualdad (y necesitamos entonces particionar el dominio bajo la relación de identidad definida por I). Podemos ahora demostrar el equivalente al Teorema 2.

Teorema 3. *Dado T, un conjunto de fórmulas-@ en $\mathcal{H}(@)$ en la signatura $S = \langle PROP, NOM, REL\rangle$, T tiene un modelo híbrido si y sólo si tiene un modelo de Herbrand híbrido sobre la signatura $S' = \langle PROP, NOM \cup NOM', REL\rangle$ donde NOM' es disjunto de NOM, infinito y numerable.*

Demostración. La demostración de derecha a izquierda es simple. Si T tiene un modelo de Herbrand I, la estructura $\langle W^I, \{R_i^I\}, V^I\rangle$ definida más arriba es un modelo híbrido de T en la signatura extendida S'. Como T no usa símbolos de NOM', el modelo restringido a la signatura S también es un modelo de T.

Para la demostración de izquierda a derecha, sea $I = \langle W^I, \{R_i^I\}, V^I\rangle$ un modelo de T. Podemos asumir sin pérdida de generalidad que W es un conjunto numerable. Sea I' cualquier extension de I a la signatura S' tal que para todo $w \in W$, $V^{I'}(w) \cap (\mathsf{NOM} \cup \mathsf{NOM}') \neq \emptyset$. Sea ahora I^H el conjunto de todos los literales positivos de S' que son ciertos en I'. Este conjunto es un modelo de Herbrand híbrido de T en S'. □

Por el resto de este capítulo, utilizaremos los términos *interpretación* y *modelo* para referirnos únicamente a *interpretaciones de Herbrand híbridas.*

3.4 Resolución para $\mathcal{H}^{NNF}(@)$ con orden y selección

Veremos en primer lugar una extensión del cálculo de resolución híbrido con orden y selección. Luego mostraremos que el cálculo que así obtenemos es refutacionalmente completo. Como sólo estaremos introduciendo restricciones sobre el dominio de aplicación de las reglas, toda derivación de la cláusula vacía que hagamos usando orden y selección será una derivación válida en el cálculo estándar. Luego, al probar que el cálculo con orden y selección es refutacionalmente completo, también estaremos probando que el cálculo original lo es.

3.4.1 Definición del cálculo

En resolución para lógica de primer orden, una función de selección puede elegir una fórmula de una cláusula, con la condición de que sea un literal negativo. Si bien en $\mathcal{H}^{NNF}(@)$ no contamos con una idea de literal que sea directamente aplicable, utilizaremos el complemento de la noción de literales positivos que definimos en la Sección 3.3.

Definición 3.12. Diremos que S es una *función de selección* si y sólo si, para cualquier cláusula C se cumple:

1. $S(C) \subseteq C$
2. $|S(C)| \leq 1$
3. $S(C) \cap \mathsf{LIT\text{-}P} = \emptyset$

Lo que esta definición nos dice es que una función de selección elige *a lo sumo* una fórmula de cada cláusula, y que aquello que seleccione no puede ser un literal positivo. La Figura 3.1 contiene las reglas para el cálculo. En ella se asume que $S(C)$ es una función de selección, $\succ$ es un orden adecuado (ver Definición 3.8) y que la premisa principal de cada regla es siempre la que aparece más a la derecha.

Como se puede ver, la Figura 3.1 difiere de la Figura 2.1 sólo en el agregado de algunas restricciones tanto locales (reglas ($\langle R \rangle$), (SYM) y (PARAM)) como generales. De acuerdo a estas últimas restricciones, en cada cláusula existe siempre una única fórmula que puede participar en alguna inferencia. Nos referiremos a ella como la *fórmula distinguida* de una cláusula y la notaremos $dist^{\succ}_{S}(C)$.

Definición 3.13. Dados un orden adecuado $\succ$ y una función de selección S, definimos $max^{\succ}(C)$ como la fórmula maximal (respecto a $\succ$) de C, y $dist^{\succ}_{S}(C)$ como la función tal que $dist^{\succ}_{S}(C) = \varphi$ si $S(C) = \{\varphi\}$ o si $S(C) = \emptyset$ y $max^{\succ}(C) = \varphi$.

Al presentar el cálculo de la Figura 2.1 en la Sección 2.4.3 observamos que las distintas reglas que lo componen se pueden dividir de acuerdo a la función que cumplen. En particular, las reglas ($\wedge$), ($\vee$), ($\langle R \rangle$) y ($@$) se encargan de descomponer la fórmula distinguida de una cláusula en fórmulas estructuralmente más simples. Lo mismo puede decirse de estas reglas en el cálculo de la Figura 3.1. Además, en este caso, la regla (SYM) lleva las igualdades a su forma normal. En definitiva, todas estas reglas se encargan de *reescribir* fórmulas que son simplificables. Sin embargo, no todas las fórmulas pueden ser reescritas por estas reglas.

Definición 3.14. Dada una signatura híbrida $S = \langle \mathsf{PROP}, \mathsf{NOM}, \mathsf{REL} \rangle$, definimos SIMP, el conjunto de *fórmulas simples* de $\mathcal{H}^{NNF}(@)$ sobre S como:

$$\mathsf{SIMP} ::= @_i\, j \ (\text{con } i \succ j) \mid @_i\, p \mid @_i \neg a \mid @_i \langle r \rangle j \mid @_i\, [r]\varphi$$

donde $i, j \in \mathsf{NOM}$, $p \in \mathsf{PROP}$, $a \in \mathsf{NOM} \cup \mathsf{PROP}$, $r \in \mathsf{REL}$ y $\varphi \in \mathcal{H}^{NNF}(@)$.

$$(\wedge)\quad \frac{Cl \cup \{@_t\,(\varphi_1 \wedge \varphi_2)\}}{\begin{array}{l} Cl \cup \{@_t\,\varphi_1\} \\ Cl \cup \{@_t\,\varphi_2\} \end{array}} \qquad (\vee)\quad \frac{Cl \cup \{@_t\,(\varphi_1 \vee \varphi_2)\}}{Cl \cup \{@_t\,\varphi_1, @_t\,\varphi_2\}}$$

$$(\text{RES})\quad \frac{Cl_1 \cup \{@_t\,\varphi\} \quad Cl_2 \cup \{@_t\,\neg\varphi\}}{Cl_1 \cup Cl_2}$$

$$([R])\quad \frac{Cl_1 \cup \{@_t\,\langle r\rangle s\} \quad Cl_2 \cup \{@_t\,[r]\varphi\}}{Cl_1 \cup Cl_2 \cup \{@_s\,\varphi\}} \qquad (\langle R\rangle)\quad \frac{Cl \cup \{@_t\,\langle r\rangle\varphi\}}{\begin{array}{l} Cl \cup \{@_t\,\langle r\rangle n\} \\ Cl \cup \{@_n\,\varphi\} \end{array}} \quad \begin{array}{l} \text{para un } n \text{ nuevo} \\ \text{y } \varphi \notin \mathsf{NOM} \end{array}$$

$$(@)\quad \frac{Cl \cup \{@_t\,@_s\,\varphi\}}{Cl \cup \{@_s\,\varphi\}} \qquad (\text{REF})\quad \frac{Cl \cup \{@_t\,\neg t\}}{Cl}$$

$$(\text{SYM})\quad \frac{Cl \cup \{@_s\,t\}}{Cl \cup \{@_t\,s\}}\ \text{si } t \succ s \qquad (\text{PARAM})\quad \frac{Cl_1 \cup \{@_s\,t\} \quad Cl_2 \cup \{\varphi(s)\}}{Cl_1 \cup Cl_2 \cup \{\varphi(s/t)\}}\ \begin{array}{l} \text{si } s \succ t \text{ y} \\ \varphi(s) \succ @_s\,t \end{array}$$

Restricciones:

- Si $C = C' \cup \{\varphi\}$ es la premisa principal, entonces o bien $S(C) = \{\varphi\}$ o, en caso contrario, $S(C) = \emptyset$ y $\{\varphi\} \succ C'$
- Si $D = D' \cup \{\psi\}$ es la premisa auxiliar, entonces $\{\psi\} \succ D'$ y $S(D) = \emptyset$

(tanto φ como ψ representan la fórmula que participa en la inferencia)

Figura 3.1: Reglas de resolución para $\mathcal{H}^{NNF}(@)$ con orden y selección

Un orden adecuado garantiza que toda cláusula producto de una inferencia es menor que la cláusula principal que se utilizó en la aplicación de la regla. Los siguientes lemas nos ayudan a demostrarlo.

Lema 4. *Si D es uno de los consecuentes de una regla de inferencia que tiene a C como premisa principal y φ es es una fórmula de D que no aparece en C (ni en la premisa auxiliar, si la hubiera), entonces $dist^{\succ}_S(C) \succ \varphi$.*

Demostración. Lo que debemos comprobar es que las fórmulas sintetizadas por cada regla son menores que la fórmula distinguida de la premisa principal. Veamos esto caso por caso:

Regla ($\wedge$). Queremos ver que $@_t\,(\varphi_1 \wedge \varphi_2) \succ @_t\,\varphi_k$ con $k \in \{1, 2\}$. Alcanza con observar que $\succ$ es un orden de reescritura, y que por la propiedad de subfórmulas $(\varphi_1 \wedge \varphi_2) \succ \varphi_k$.

Regla ($\vee$). Es análogo al caso anterior.

Regla ($[R]$). Por la propiedad de subfórmulas $[r]\varphi \succ \varphi$, luego, por ser $\succ$ un orden adecuado, se cumple $@_t\,[r]\varphi \succ @_s\,\varphi$, que es lo que teníamos que comprobar.

Regla ($\langle R\rangle$). En este caso necesitamos ver dos cosas. Por un lado, tenemos que comprobar que si $\varphi \notin \mathsf{NOM}$ entonces $@_t\,\langle r\rangle\varphi \succ @_t\,\langle r\rangle n$ con $n \in \mathsf{NOM}$. Esto es sencillo, ya que en todo orden adecuado $\varphi \succ n$. Por otro lado, podemos ver que se cumple $@_t\,\langle r\rangle\varphi \succ @_n\,\varphi$ con un procedimiento análogo al que utilizamos para la regla ($[R]$).

Regla (@). Es directo ver que, por la propiedad de subfórmulas, $@_t\,@_s\,\varphi \succ @_s\,\varphi$.

Regla (SYM). Por ser $\succ$ adecuado, si $t \succ s$ entonces $@_s\,t \succ @_t\,s$.

Regla (PARAM). Como $\succ$ es un orden de reescritura, si $s \succ t$ debe valer $\varphi(s) \succ \varphi(s/t)$. □

Lema 5. *Si C y D son respectivamente las premisas principal y secundaria en la aplicación de una regla de inferencia binaria, entonces $dist^{\succ}_S(C) \succ dist^{\succ}_S(D)$.*

Demostración. El único caso de interés es el de ($[R]$), donde debemos comprobar que $@_t\,[r]\varphi \succ @_t\,\langle r\rangle s$. Si $\varphi \in \mathsf{NOM}$, entonces el orden adecuado garantiza en ese caso $[r]\varphi \succ \langle r\rangle s$. Si, por el contrario, $\varphi \notin \mathsf{NOM}$, entonces para todo $i \in \mathsf{NOM}$ se cumple $[r]\varphi \succ [r]i \succ \langle r\rangle s$. En ambos casos, la demostración se completa valiéndonos del hecho de que $\succ$ es un orden de reescritura. □

Lema 6. *Si C y C_A son respectivamente las premisas principal y secundaria en la aplicación de una regla de inferencia binaria, entonces $\{dist_S^{\succ}(C)\} \succ C_A$*

Demostración. Supongamos que $C_A = C'_A \cup \{\psi\}$ con $dist_S^{\succ}(C_A) = \psi$. Las restricciones que el cálculo impone nos garantizan que $\{\psi\} \succ C'_A$. Por otro lado, el Lema 5 nos dice que $dist_S^{\succ}(C) \succ \psi$, con lo cual $\{dist_S^{\succ}(C)\} \succ C'_A$. □

Lema 7. *Si C es la premisa principal de una inferencia que tiene a D como consecuente, entonces $C \not\subseteq D$*

Demostración. Si la inferencia es unaria, entonces $dist_S^{\succ}(C) \notin D$. Supongamos, en cambio, que la inferencia es binaria con C_A como premisa auxiliar. Por el Lema 6, $\{dist_S^{\succ}(C)\} \succ C_A$ y por lo tanto $dist_S^{\succ}(C) \notin C_A$, con lo cual $dist_S^{\succ}(C) \notin D$. □

Ahora sí, podemos enunciar y demostrar la propiedad antes mencionada.

Proposición 8. *Si C es la premisa principal de una inferencia que tiene a D como uno de sus consecuentes, entonces $C \succ D$.*

Demostración. El Lema 7 prueba la primera de las condiciones de la Definición 3.7. Nos resta ver que para toda ψ tal que $\psi \in D$ y $\psi \notin C$ existe $\varphi \in C$ tal que $\varphi \notin D$ y $\varphi \succ \psi$. Ahora bien, si $\psi \in D$ pero $\psi \notin C$, entonces o bien ψ es producto de la inferencia o bien $\psi \in C_A$ donde C_A es la premisa auxiliar de la inferencia. En cualquiera de los dos casos, los Lemas 4 y 6 garantizan $dist_S^{\succ}(C) \notin D$ y $dist_S^{\succ}(C) \succ \psi$. □

3.4.2 Completitud refutacional

El esquema de esta demostración está basado en el que se utiliza en [Bachmair and Ganzinger, 2001] para el caso de resolución para lógica de primer orden. Esencialmente, lo que haremos es dar un mecanismo para construir un modelo a partir de un conjunto arbitrario de cláusulas, de forma tal que si la menor de las cláusulas no es verdadera en este modelo, entonces el cálculo permita derivar una nueva cláusula, menor que la anterior, y que tampoco sea satisfecha por el modelo. De aquí podremos concluir que en un conjunto clausurado respecto a la aplicación de las reglas, o bien este mecanismo nos provee un modelo que lo satisface, o bien la cláusula vacía pertenece a este conjunto.

Modelos tentativos

Definición 3.15. Dada I, una interpretación de Herbrand híbrida, definimos la siguiente sustitución de nominales por nominales:

$$\sigma_I = \{i \mapsto j | i \sim_I j \wedge (\forall k)(k \sim_I j \rightarrow k \succeq j)\}$$

σ_I sustituye cada nominal por el menor nominal de su clase, el cual se toma como representante. Cuando no haya ambigüedades, usaremos σ en lugar de σ_I.

Supongamos un conjunto fijo de cláusulas N. Las siguientes tres definiciones deben tomarse como una unidad. Se presentan en forma separada por claridad pero, como se verá, son las tres mutuamente recursivas.

Definición 3.16 (I_C). Sea C una cláusula (no necesariamente perteneciente a N), llamaremos I_C a la interpretación de Herbrand híbrida dada por $\bigcup_{C \succ D} \varepsilon_D$

Definición 3.17 (Forma reducida). Sean C una cláusula y φ su fórmula maximal. Si $\varphi \in \mathsf{SIMP}$ y se cumple que

- $\varphi \in \mathsf{LIT\text{-}P}$ y $\varphi = \varphi\sigma_{I_C}$, ó
- $\varphi = @_i\,[r]\psi$ y $i = i\sigma_{I_C}$

entonces decimos que tanto φ como C se encuentran en *forma reducida.*

Definición 3.18 (ε_C). Sea C una cláusula (no necesariamente de N); si se verifica simultáneamente:

1. $C \in N$
2. C está en forma reducida
3. $max^{\succ}(C) \in$ LIT-P
4. C es falsa en I_C
5. $S(C) = \emptyset$

entonces $\varepsilon_C = \{\varphi\}$; en caso contrario, ε_C es el conjunto vacío.

Diremos que C *produce* φ si $\varepsilon_C = \{\varphi\}$ y la llamaremos una *cláusula productiva.* I_C es la *interpretación parcial de N por debajo de C.* Con esto queremos decir que I_C intenta ser un modelo del conjunto de cláusulas de N menores que C (con respecto a $\succ$). La *interpretación parcial de N en C* es el conjunto potencialmente extendido $I^C = I_C \cup \varepsilon_C$.

Sólo las cláusulas cuya fórmula maximal φ sea un literal positivo y no tengan fórmulas seleccionadas pueden ser productivas. En este caso, decimos que la cláusula es *reductiva* para φ.

Definición 3.19. I_N, un modelo tentativo de N, se define como $\bigcup_{C \in N} \varepsilon_C$.

Ejemplo 5. Sean $i \succ j \succ k$ nominales y $p \succ q \succ r$ símbolos de proposición. Las fórmulas seleccionadas dentro de una cláusula se marcan con un asterisco.

$C_1 = \{@_k\, j\}$	$I_{C_1} = \emptyset$	$\varepsilon_{C_1} = \emptyset$	No es una fórmula simple
$C_2 = \{@_i\, k, @_k\, j\}$	$I_{C_2} = \emptyset$	$\varepsilon_{C_2} = \{@_i\, k\}$	
$C_3 = \{@_j\, p, @_k\, q\}$	$I_{C_3} = \{@_i\, k\}$	$\varepsilon_{C_3} = \{@_j\, p\}$	
$C_4 = \{@_i\, r, @_j\, q\}$	$I_{C_4} = \{@_i\, k, @_j\, p\}$	$\varepsilon_{C_4} = \emptyset$	No está en forma reducida
$C_5 = \{@_k\, \neg r, @_i\, r\}$	$I_{C_5} = \{@_i\, k, @_j\, p\}$	$\varepsilon_{C_5} = \emptyset$	No es un literal positivo
$C_6 = \{@_k\, \Diamond j, @_j\, p\}$	$I_{C_6} = \{@_i\, k, @_j\, p\}$	$\varepsilon_{C_6} = \emptyset$	No es falsa en I_{C_6}
$C_7 = \{@_k\, \Diamond j, @_j\, \neg r*\}$	$I_{C_7} = \{@_i\, k, @_j\, p\}$	$\varepsilon_{C_7} = \emptyset$	$S(C_7) \neq \emptyset$
$C_8 = \{@_j\, \Diamond k, @_j\, \neg r\}$	$I_{C_8} = \{@_i\, k, @_j\, p\}$	$\varepsilon_{C_8} = \{@_j\, \Diamond k\}$	
$C_9 = \{@_i\, \Diamond j\}$	$I_{C_9} = \{@_i\, k, @_j\, p, @_j\, \Diamond k\}$	$\varepsilon_{C_9} = \emptyset$	No está en forma reducida
$C_{10} = \{@_j\, \Diamond p, @_j\, i\}$	$I_{C_{10}} = \{@_i\, k, @_j\, p, @_j\, \Diamond k\}$	$\varepsilon_{C_{10}} = \emptyset$	No es una fórmula simple
$C_{11} = \{@_k\, \Box j, @_j\, q\}$	$I_{C_{11}} = \{@_i\, k, @_j\, p, @_j\, \Diamond k\}$	$\varepsilon_{11} = \emptyset$	No es un literal positivo
$C_{12} = \{@_k\, \Box(p \wedge q)\}$	$I_{C_{12}} = \{@_i\, k, @_j\, p, @_j\, \Diamond k\}$	$\varepsilon_{C_{12}} = \emptyset$	No es una fórmula simple

Finalmente, suponiendo $N = \bigcup_{i=1}^{12} C_i$, tenemos $I_N = \{@_k\, i, @_j\, p, @_j\, \Diamond k\}$. Es interesante notar que en este ejemplo I_N no satisface N.

Características de I_N

En lo que sigue intentamos clarificar la idea de estos modelos tentativos observando sus propiedades más importantes. Hay que tener en cuenta, en todo momento, que estamos aprovechando que $\succ$ sea un orden adecuado. Empezamos observando las características más elementales.

Observación 9. *Si C es productiva, entonces $S(C) = \emptyset$ y $max^{\succ}(C)$ es un literal positivo.*

Lema 10. *Si C produce una igualdad, ésta se encuentra normalizada y todas las fórmulas de C son también igualdades.*

Demostración. Si C es productiva, debe estar en forma reducida. De acuerdo a la Definición 3.17, $max^{\succ}(C)$ debe ser una fórmula simple; con lo cual, si es una igualdad, debe estar normalizada. Para ver que todas las fórmulas de C deben ser igualdades basta con suponer que $@_k\, \psi$ es una cláusula de C con $\psi \notin$ NOM. Si $max^{\succ}(C)$ es una igualdad de la forma $@_i\, j$, entonces tendríamos $\psi \succ j$ y, por lo tanto $@_k\, \psi \succ @_i\, j$, lo cual es un absurdo. □

En la demostración anterior sale a relucir una característica interesante del orden adecuado: las igualdades son las *menores* fórmulas-@ del orden. Con esto queremos decir que si $@_i\, j \succ @_k\, \psi$, entonces ψ debe ser un nominal. De aquí se desprende el siguiente lema:

Lema 11. *Sea N un conjunto de cláusulas arbitrario; si C es una cláusula tal que $max^{\succ}(C)$ no es un igualdad, entonces $\sigma_{I_C} = \sigma_{I_N}$.*

Lo que este lema nos está diciendo es que si C no es una cláusula formada por igualdades, entonces ni C ni ninguna cláusula mayor que C produce una igualdad. Esto quiere decir que la partición en clases de equivalencia de nominales inducida por I_C es la misma que la inducida por I_N. Estos modelos tentativos tienen una característica adicional que será de utilidad más adelante: si $@_j\, i \in I_N$, entonces i es el menor nominal de la clase de equivalencia a la que i y j pertenecen. Esto es lo que nos dice el siguiente lema.

Lema 12. *Sea N un conjunto de cláusulas. Si i es un nominal tal que $i\sigma_{I_N} \neq i$, entonces en I_N existe una única igualdad en la que aparece i, y ésta es de la forma $@_i\, n$ donde $n = i\sigma_{I_N}$.*

Demostración. Dividamos la demostración en dos partes. Veamos primero que no pueden aparecer en I_N dos igualdades (normalizadas) de la forma $@_i\, m$ y $@_i\, n$. Luego demostraremos que si $@_i\, n$ pertenece a I_N entonces n tiene que cumplir $n = n\sigma_{I_N}$. Como esto lo estaremos probando para un i arbitrario, no será necesario probar que no puede suceder, por ejemplo, que $@_n\, i$ y $@_i\, m$ pertenezcan simultáneamente a I_N.

Razonando por el absurdo, supongamos que en I_N aparecen dos igualdades distintas de la forma $@_i\, n$ y $@_i\, m$. Supongamos que C_1 produce la primera y C_2 la segunda y, sin perder generalidad, que $C_1 \succ C_2$. Bajo estas hipótesis, es evidente que debe suceder $\varepsilon_{C_1} = \{@_i\, n\}$ y, además, $@_i\, m \in I_{C_1}$. Ahora bien, por la Definición 3.18, si C_1 produce $@_i\, n$, entonces C_1 está en *forma reducida*. En particular, se debe cumplir $i = i\sigma_{I_{C_1}}$. Pero si $@_i\, m \in I_{C_1}$, entonces $i\sigma_{I_{C_1}} \neq i$ (porque sabemos que al menos m es menor que i), y por lo tanto, C_1 no puede ser productiva, lo cual es una contradicción.

Nos resta comprobar que si $@_i\, n \in I_N$, entonces $i\sigma_{I_N} = n$. Supongamos que éste no fuera el caso. Esto querría decir que existen dos cláusulas C_1 y C_2 tales que C_1 produce $@_i\, n$ y C_2 produce $@_n\, m$ con $i \succ n \succ m$. Si esto sucediera, sería evidente que $@_i\, n \succ @_n\, m$ y, por lo tanto, $C_1 \succ C_2$. De esto podemos deducir que $@_n\, m \in I_{C_1}$. Ahora bien, para que ε_{C_1} produzca $@_i\, n$, se debe cumplir que $n\sigma_{I_{C_1}} = n$ (ya que debe estar en forma reducida). Una vez más, esto no se cumple y C_1 no debería ser productiva. Hemos llegado otra vez a un absurdo. □

Los siguientes lemas muestran que la satisfacibilidad de una cláusula C en I_D, con $D \succeq C$ no se ve afectada por las fórmulas producidas por cláusulas mayores o iguales que D.

Lema 13. *Si C es productiva, entonces C es verdadera en I_N.*

Demostración. Si C es productiva entonces $\varepsilon_C \cap C \neq \emptyset$, y como $\varepsilon_C \subseteq I_N$, entonces C es verdadera en I_N. □

Lema 14. *Sean C y D dos cláusulas tales que $D \succeq C$. Si C es verdadera en I_D o en I^D, entonces C es también verdadera en I_N y en todas las interpretaciones $I_{D'}$ e $I^{D'}$, donde $D' \succeq D$.*

Demostración. Conviene empezar recordando que $I_D \subseteq I^D \subseteq I_{D'} \subseteq I^{D'} \subseteq I_N$ para todo $D' \succ D$. Ahora bien, si C contiene un literal positivo φ que es verdadero en I_D, entonces, por lo antedicho, φ será verdadero en I_N y en todas las interpretaciones $I_{D'}$ y $I^{D'}$.

Para tratar el caso en que C no contiene ningún literal positivo que sea verdadero en I_D, razonaremos por el absurdo suponiendo que C contiene una fórmula $\varphi \notin \mathsf{LIT\text{-}P}$ que es verdadera en I_D pero que no lo es en I_N. Como $\succ$ está bien fundado, podemos suponer sin perder generalidad que D' es la mínima cláusula tal que φ es verdadera en $I_{D'}$ pero no lo es en $I^{D'}$. Esto significa que D' es reductiva para una fórmula que altera la interpretación parcial por debajo de D' de forma tal que φ deja de ser verdadera. Por la Observación 9, sabemos que hay sólo tres posibilidades: $\varepsilon_{D'} = \{@_i\, j\}$, o $\varepsilon_{D'} = \{@_i\, p\}$, o $\varepsilon_{D'} = \{@_i\, \langle r \rangle j\}$. Analicemos caso por caso.

Caso $\varepsilon_{D'} = \{@_i\, j\}$. Si $\varphi \notin$ LIT-P, entonces es claro que $max^{\succ}(C) \succeq \varphi \succ @_i\, j$, y esto viola la hipótesis de que $D' \succ C$.

Caso $\varepsilon_{D'} = \{@_i\, p\}$. En este caso tenemos que concluir que $\neg p \in S_f(\varphi)$. Pero entonces $\varphi \succ p$, con lo cual $max^{\succ}(C) \succeq \varphi \succ @_i\, p$ y, otra vez, esto no es posible.

Caso $\varepsilon_{D'} = \{@_i\, \langle r\rangle j\}$. Aquí sería necesario $[r]\psi \in S_f(\varphi)$ para algún ψ. Luego, $\varphi \succ [r]\psi \succ \langle r\rangle j$ (notar que si $\psi \notin$ NOM, entonces $[r]\psi \succ [r]j \succ \langle r\rangle j$) y, una vez más, $max^{\succ}(C) \succeq \varphi \succ @_i\, \langle r\rangle j$. La conclusión a la que llegamos es que D' no puede ser productiva, que es el absurdo que buscábamos. □

Corolario 15. *Si C es falsa en I_N, entonces C es falsa en I_C e I^C.*

Lema 16. *Si $D = D' \cup \{\varphi\}$ es una cláusula reductiva para φ, entonces D' es falsa en I_N.*

Demostración. Supongamos que $\psi \in D'$ es verdadera en I_N. Por ser D productiva, ψ debe ser falsa en I_D. Sea $C \succeq D$ la menor cláusula tal que ψ es falsa en I_C pero verdadera en I^C. Al igual que en la demostración anterior, analicemos los posibles valores de ε_C.

Caso $\varepsilon_C = \{@_t\, s\}$. En este caso, por los Lemas 10 y 11 concluimos que ψ es también una igualdad. Sea, entonces, $\psi = @_j\, i$. Es fácil ver (teniendo en cuenta el Lema 12) que o bien $\varepsilon_C \subset \{@_i\, j, @_j\, i\}$ o bien $\varepsilon_C \subset \{@_i\, k, @_j\, k\} \subset I^C$ ($\varepsilon_C \neq \emptyset$). Este último no puede ser el caso ya que deberíamos tener $i \succ k$ y $j \succ k$ y por ser $\succ$ un orden de reescritura obtendríamos $@_j\, i \succ @_i\, k$ y $@_j\, i \succ @_j\, k$ (para el primer caso, notar que si $i \succ j$, entonces $@_j\, i \succ @_i\, j \succ @_i\, k$, mientras que si $j \succ i$, $@_j\, i \succ @_j\, k \succ @_i\, k$). Debemos suponer, entonces que $\varepsilon_C \subset \{@_i\, j, @_j\, i\}$. Claramente, no puede ser el caso $\varepsilon_C = \{@_j\, i\}$, ya que en ese caso tendríamos $@_j\, i \succeq \varphi \succ @_j\, i$. Sólo nos queda suponer que $\varepsilon_C = \{@_i\, j\}$, con lo cual $@_i\, j$ debe estar normalizada y, por lo tanto, llegamos al absurdo de $@_j\, i \succ @_i\, j \succeq \varphi \succ @_j\, i$.

Caso $\varepsilon_C = \{@_i\, p\}$. En este caso debemos concluir que $p \in S_f(\psi)$. No puede suceder $\psi = @_k\, p$ porque entonces tendríamos $k\sigma_{I_C} = i$ con lo cual $k \succeq i$ y, por lo tanto $\psi \succeq @_i\, p \succeq \varphi \succ \psi$. Luego, deberíamos suponer $\psi = @_k\, \psi'$ con $\psi' \succ p$, pero entonces obtendríamos $\psi \succ @_i\, p \succeq \varphi \succ \psi$.

Caso $\varepsilon_C = \{@_i\, \langle r\rangle j\}$. Aquí es necesario que se cumpla $\langle r\rangle\psi' \in S_f(\psi)$ para alguna fórmula ψ'. Si $\psi' \notin$ NOM, entonces claramente $\psi \succ @_i\, \langle r\rangle j \succeq \varphi \succ \psi$. Supongamos, en cambio, $\psi' = k$ para algún nominal k. Si $\langle r\rangle k$ es una subfórmula propia de ψ, entonces, por ser $\succ$ adecuado, $\psi \succ @_i\, \langle r\rangle j$. Nos queda por considerar el caso en que $\psi = @_n\, \langle r\rangle k$. Ahora bien, si $@_n\, \langle r\rangle k$ se vuelve verdadera en I^C por la presencia de $@_i\, \langle r\rangle j$, tenemos que concluir que $n\sigma_{I_C} = i$ y $k\sigma_{I_C} =$ j; pero entonces tendríamos $n \succeq i$ y $k \succeq j$, con lo cual $\psi \succeq @_i\, \langle r\rangle j \succeq \varphi \succ \psi$. □

Reducción de contraejemplos

Las propiedades de I_N que acabamos de ver nos servirán para mostrar la característica más interesante de los modelos tentativos: si $C \in N$ es falsa en I_N entonces es posible aplicar alguna de las reglas del cálculo de la Figura 3.1 (de ser necesario, con alguna otra cláusula de N) sobre C para así obtener un consecuente que también es falso en I_N. Esto es lo único que nos falta comprobar para así poder demostrar la completitud refutacional del cálculo. Los siguientes lemas se encargan de esta comprobación.

Definición 3.20. Diremos que C es un *contraejemplo* de I si C es falsa en I.

Lema 17. *Sea $C \in N$ un contraejemplo de I_N. Si $dist_S^{\succ}(C) \notin$ SIMP, entonces se puede aplicar alguna inferencia unaria sobre C y alguno de los consecuentes resultantes es también un contraejemplo de I_N.*

Demostración. Sea $\varphi = dist_S^{\succ}(C)$. Es claro que por no ser φ una fórmula simple, debe ser posible aplicar sobre C alguna de las reglas ($\wedge$), ($\vee$), ($\langle R\rangle$), (@) o (SYM). Basta ver que en cada caso obtenemos al menos un contraejemplo.

Caso ($\wedge$). Si la regla ($\wedge$) se puede aplicar a $C = C' \cup \{\varphi\}$ entonces φ debe ser de la forma $@_i\, (\varphi_1 \wedge \varphi_2)$. Los consecuentes de esta inferencia serán las cláusulas $D_k = C' \cup \{@_i\, \varphi_k\}$ con

$k \in \{1,2\}$. Sabemos que C es falso en I_N; esto significa que C' y φ también lo son. Esto sólo es posible en medida en que $@_i\,\varphi_1$ o $@_i\,\varphi_2$ lo sean, con lo cual D_1 o D_2 debe ser un contraejemplo de I_N.

Caso ($\vee$). Si la regla ($\vee$) se puede aplicar a $C = C' \cup \{\varphi\}$, entonces $\varphi = @_i\,(\varphi_1 \vee \varphi_2)$. El consecuente será la cláusula $D = C' \cup \{@_i\,\varphi_1, @_i\,\varphi_2\}$. Queremos ver que D es un contraejemplo de I_N si C lo es. Otra vez, esto es fácil de ver ya que si φ es falso en I_N entonces $@_i\,\varphi_1$ y $@_i\,\varphi_2$ también lo son.

Caso ($\langle R\rangle$). Si la regla ($\langle R\rangle$) se puede aplicar a $C = C' \cup \{\varphi\}$, entonces $\varphi = @_i\,\langle r\rangle\psi$ y $\psi \notin \mathsf{NOM}$. En este caso, los consecuentes son las cláusulas $D = C' \cup \{@_i\,\langle r\rangle n\}$ y $D' = C' \cup \{@_n\,\psi)\}$ para algún n. Dado que C' es un contraejemplo de I_N, nos alcanza con observar que no es posible que sean $@_i\,\langle r\rangle n$ y $@_n\,\psi$ simultáneamente verdaderas en I_N, ya que en ese caso $@_i\,\langle r\rangle\psi$ también debería serlo y C no podría ser un contraejemplo. Luego, o bien D o bien D' debe ser un contraejemplo de I_N.

Caso (@). Si la regla (@) se puede aplicar a $C = C' \cup \{\varphi\}$, significa que $\varphi = @_i\,@_j\,\psi$ y que su consecuente es una cláusula $D = C' \cup \{@_j\,\psi\}$. Como C es un contraejemplo de I_N entonces C' y $@_i\,@_j\,\psi$ deben ser falsos en I_N. Luego, $@_j\,\psi$ es falso en I_N y de esto concluimos que D también lo es.

Caso (SYM). Si la regla (SYM) se puede aplicar a $C = C' \cup \{\varphi\}$ concluimos que $\varphi = @_i\,j$ con $i \succ j$. El consecuente es en este caso $D = C' \cup \{@_j\,i\}$, que es lógicamente equivalente a C y, por lo tanto, es un contraejemplo de I_N. □

Lema 18. *Sea $C \in N$ un contraejemplo de I_N. Si $dist^{\succ}_S(C) \in$ SIMP y no se encuentra en forma reducida, entonces existe una cláusula D reductiva para una igualdad tal que la aplicación de la regla (PARAM) con C y D como premisas genera un consecuente que es también un contraejemplo de I_N.*

Demostración. Sea $\varphi = dist^{\succ}_S(C)$. Como φ es simple pero no está en forma reducida, se desprende de la Definición 3.17 que $\varphi \neq \varphi\sigma_{I_C}$. Esto significa que para algún nominal $i \in S_f(\varphi)$ se cumple $i\sigma_{I_C} \neq i$. Luego, debe existir una cláusula D reductiva para $@_i\,j$ tal que $C \succ D$. Para ver que es posible aplicar (PARAM) entre C y D debemos comprobar que $\varphi \succ @_i\,j$. Si φ no es un igualdad, esto es directo. Si, por el contrario, φ es una igualdad entonces hay observar que no puede suceder $S(C) = \{\varphi\}$ ya que $\varphi \in$ LIT-P. Luego, tenemos $max^{\succ}(C) = \varphi$, $max^{\succ}(D) = @_i\,j$ y como $\varphi \neq @_i\,j$ concluimos $\varphi \succ @_i\,j$.

Nos falta comprobar que el consecuente de esta inferencia es también un contraejemplo de I_N. Sea, entonces, $C_R = \{\varphi(j/i)\} \cup C' \cup D'$ dicho consecuente, con $C' = C \setminus \{\varphi\}$ y $D' = D \setminus \{@_i\,j\}$. Basta observar que C' es falsa en I_N por hipótesis, que $\varphi(j/i)$ no puede ser verdadera en I_N siendo que φ es falsa y $@_i\,j$ es verdadera, y que, por el Lema 16, D' debe ser también falsa en I_N. □

Lema 19. *Sea $C \in N$ un contraejemplo de I_N. Si $dist^{\succ}_S(C) = @_i\,\neg i$ para algún nominal i, entonces se puede aplicar la regla (REF) sobre C y el resultado es otro contraejemplo de I_N.*

Lema 20. *Sea $C \in N$ un contraejemplo de I_N. Si $dist^{\succ}_S(C)$ es de la forma $@_i\,[r]\varphi$ y se encuentra en forma reducida, entonces existe una cláusula productiva D, tal que C y D pueden ser premisas de una aplicación de la regla ($[R]$) que genera como consecuente otro contraejemplo de I_N.*

Demostración. Supongamos que C es una cláusula como piden las hipótesis. Por el Corolario 15 podemos concluir que C es falsa en I_C. Esto significa que algún j verifica $G_{I_C}, i \models \langle r\rangle j$ y $G_{I_C}, j \models \neg\varphi$. Supongamos, además, que j es el mínimo nominal que cumple con esto. Teniendo en cuenta el Lema 12 y que C está en forma reducida podemos concluir que $@_i\,\langle r\rangle j \in I_C$. Esto significa que debe existir una cláusula D reductiva para $@_i\,\langle r\rangle j$. Si se aplica la regla ($[R]$) entre C y D se obtiene como consecuente $C_R = C' \cup D' \cup \{@_j\,\varphi\}$, con $C' = C \setminus \{@_i\,[r]\varphi\}$ y $D' = D \setminus \{@_i\,\langle r\rangle j\}$.

Nos queda ver que C_R es también un contraejemplo para I_N. Sabemos que C es falso en I_N y que, por lo tanto, C' también lo es. Además, si $G_{I_N}, j \models \neg\varphi$, no puede ser que $@_j\,\varphi$ verdadera en I_N. Por último, usando el Lema 16 podemos observar que D' también es falso en I_N. □

Lema 21. *Sea $C \in N$ un contraejemplo de I_N. Si $dist^{\succ}_S(C)$ está en forma reducida y es de la forma $@_i \neg a$ con $a \in \mathsf{ATOM}$ y $a \neq i$, entonces existe una cláusula productiva D que puede ser junto con C premisa de una aplicación de la regla (RES) cuyo resolvente es también un contraejemplo de I_N.*

Demostración. Por el Corolario 15, C es un contraejemplo de I_C, con lo cual $@_i \neg a$ debe ser falso en I_C. Dado que C se encuentra en forma reducida, esto sólo podrá suceder en la medida en que $@_i a$ pertenezca a I_C. De aquí concluimos que debe existir una cláusula D, menor que C y reductiva para $@_i a$. Luego, es posible aplicar la regla (RES) entre C y D para así obtener $C_R = C' \cup D'$ donde $C' = C \setminus \{@_i \neg a\}$ y $D' = D \setminus \{@_i a\}$. Por el Lema 16, D' debe ser falsa en I_N. Como, además, por hipótesis, C' es falso en I_N podemos concluir que C_R es un contraejemplo de I_N. □

Ya estamos en condiciones de mostrar que el cálculo con orden y selección es refutacionalmente completo; esto es, que si una fórmula φ no es satisfacible, entonces $\{\} \in \mathit{ClSet}^*(\varphi)$.

Teorema 22. *Sea N un conjunto de cláusulas y sea C el contraejemplo mínimo de I_N, respecto de un orden adecuado $\succ$. Si $C \neq \{\}$, entonces existe una inferencia usando alguna de las reglas del cálculo tal que:*

1. *C es la premisa principal*
2. *la premisa auxiliar (en los casos que corresponde) es una cláusula productiva*
3. *todos los consecuentes son menores, respecto de $\succ$, que C y al menos uno es un contraejemplo de I_N.*

Demostración. Por la Proposición 8, sabemos que alcanza con demostrar que siempre existe una regla de inferencia que genera otro contraejemplo. Para esto, simplemente mostraremos que con los lemas que ya hemos visto se cubren todos los casos. Sea, entonces, $\varphi = dist^{\succ}_S(C)$.

- Si $\varphi \notin \mathsf{SIMP}$, entonces, el Lema 17 garantiza que existe la inferencia que necesitamos.
- Por el contrario, si $\varphi \in \mathsf{SIMP}$ entonces
  - Si φ no se encuentra en forma reducida, podemos usar el Lema 18
  - Si, en cambio, φ se encuentra en forma reducida, entonces
    - No es posible que sea un literal positivo ya que en dicho caso tendríamos $S(C) = \emptyset$ y por lo tanto C cumpliría con todas las condiciones para ser productiva.
    - Si es de la forma $@_i \neg i$ entonces utilizamos el Lema 19.
    - Si es de la forma $@_i \neg a$, $a \in \mathsf{ATOM}$, y $i \neq a$, usamos el Lema 21.
    - Si es de la forma $@_i [r]\psi$ usamos el Lema 20.

□

Teorema 23. *El cálculo de resolución híbrido con orden y selección es refutacionalmente completo.*

Demostración. Supongamos que el cálculo no fuera refutacionalmente completo. Debería existir una fórmula insatisfacible φ tal que $\mathit{ClSet}^*(\varphi)$ no contenga la cláusula vacía. Como $\mathit{ClSet}(\varphi)$ está incluido en $\mathit{ClSet}^*(\varphi)$, éste no puede ser satisfacible. Luego, debe existir una cláusula en $\mathit{ClSet}^*(\varphi)$ que sea un contraejemplo de $I_{\mathit{ClSet}^*(\varphi)}$. Además, como $\succ$ está bien fundado, $\mathit{ClSet}^*(\varphi)$ debe tener un contraejemplo mínimo. Llamemos C a este contraejemplo. Por el Teorema 22, si C es un contraejemplo mínimo de $\mathit{ClSet}^*(\varphi)$ y éste no contiene la cláusula vacía, entonces C puede ser la premisa principal de una inferencia que, de ser binaria, usará como premisa auxiliar una $D \in \mathit{ClSet}^*(\varphi)$ y tal que su consecuente E debe ser un contraejemplo de $I_{\mathit{ClSet}^*(\varphi)}$ con $C \succ E$. Ahora bien, si E es el resultado de aplicar una de las reglas del cálculo sobre cláusulas de $\mathit{ClSet}^*(\varphi)$,

entonces E también debe pertenecer a $ClSet^*(\varphi)$. Pero en ese caso E sería un contraejemplo de $I_{ClSet^*(\varphi)}$ estrictamente menor que C y, por hipótesis, C era el menor contraejemplo de $I_{ClSet^*(\varphi)}$. Hemos llegado a un absurdo que resulta de suponer que siendo $ClSet(\varphi)$ un conjunto insatisfacible, $ClSet^*(\varphi)$ no contiene la cláusula vacía. □

Capítulo 4

Terminación del cálculo

El cálculo de resolución con el que venimos trabajando constituye, como muestra el Teorema 23, un método correcto y completo para resolver el problema de determinar si cierta fórmula de $\mathcal{H}(@)$ es satisfacible. En este capítulo investigamos cómo usar este cálculo como método de decisión para dicho problema. Se dice que un método es de *decisión* para el problema P si para *cualquier* instancia de P podemos obtener una respuesta *correcta* acerca de P en un *número finito de pasos.* La consistencia y completitud refutacional del cálculo garantizan, respectivamente, que la respuesta sea *correcta* para *cualquier* instancia del problema. Lo que nos interesa ver, entonces, es cómo garantizar que ésta se obtenga en un *número finito de pasos.* Lo que buscaremos, concretamente, es lograr que para cualquier fórmula $\varphi \in \mathcal{H}(@)$, $\mathit{ClSet}^*(\varphi)$ sea un conjunto *finito.* Cuando esto se cumple, es fácil implementar un algoritmo que compute en tiempo finito $\mathit{ClSet}^*(\varphi)$ (la forma estándar es usando el "algoritmo de la cláusula dada", presentado en la Sección 5.1.1).

El cálculo de la Figura 2.1 claramente puede generar un conjunto saturado de fórmulas que sea infinito. Basta con observar que la regla $(\langle R \rangle)$ se aplica aun sobre fórmulas de la forma $@_i \langle r \rangle j$ donde j es un nominal.

El cálculo con orden y selección que presentamos en la Figura 3.1 restringe la aplicación de $(\langle R \rangle)$ a fórmulas que no sean relaciones y, sin embargo, como veremos en la Sección 4.1, también con este cálculo es posible generar un conjunto saturado infinito.

En las Secciones 4.2 y 4.3 presentaremos una variación del cálculo de la Figura 3.1 que preserva completitud y, finalmente, en la Sección 4.4 mostraremos que con dicho cálculo podemos garantizar terminación.

Durante este capítulo, asumiremos que está dado un orden adecuado (según la Definición 3.8) $\succ$. Además, necesitaremos utilizar inducción sobre una *secuencia de derivación* para demostrar ciertas propiedades de $\mathit{ClSet}^*(\varphi)$.

Definición 4.1. Decimos que S es una *secuencia de derivación de* $\mathrm{ClSet}^*(\varphi)$ si S es una secuencia de cláusulas tal que:

1. no hay repetidos en S
2. una cláusula C pertenece a S si y sólo si pertenece a $\mathit{ClSet}^*(\varphi)$
3. para toda cláusula C en S, si $\mathit{ClSet}(\varphi) \neq \{C\}$, entonces en S aparecen *antes* que C, cláusulas que pueden ser usadas como premisas de alguna de las reglas para generar C

Una secuencia de derivación de $\mathit{ClSet}^*(\varphi)$ es, en otras palabras, una forma de ordenar las cláusulas de $\mathit{ClSet}^*(\varphi)$ de manera que las premisas figuren antes que sus consecuentes. Las demostraciones que haremos serán usando inducción sobre la longitud de los prefijos de alguna de estas secuencias; para el caso base analizamos $\mathit{ClSet}(\varphi)$ y en el paso inductivo suponemos, como hipótesis inductiva, que la propiedad se cumple sobre todas las cláusulas que aparecen en el prefijo de longitud n.

4.1 Cómo generar infinitas fórmulas

Tomemos la fórmula satisfacible $@_i([r](i \wedge \langle r\rangle p) \wedge \langle r\rangle p)$ y asumamos que i es el menor nominal de un orden adecuado. Veamos qué sucede cuando aplicamos las reglas del cálculo sobre esta fórmula:

$$
\begin{array}{rll}
C = & \{@_i([r](i \wedge \langle r\rangle p) \wedge \langle r\rangle p)\ \} & \\
D_{1a} = & \{@_i[r](i \wedge \langle r\rangle p)\} & (C, \text{ usando } (\wedge)) \\
D_{1b} = & \{@_i \langle r\rangle p\} & \\
D_{2a} = & \{@_i \langle r\rangle k\} & (D_{1b}, \text{ usando } (\langle R\rangle)) \\
D_{2b} = & \{@_k p\} & \\
D_3 = & \{@_k (i \wedge \langle r\rangle p)\} & (D_{1a} \text{ y } D_{2a}, \text{ usando } ([R])) \\
D_{4a} = & \{@_k i\} & (D_3, \text{ usando } (\wedge)) \\
D_{4b} = & \{@_k \langle r\rangle p\} & \\
D_{5a} = & \{@_k \langle r\rangle k_2\} & (D_{4b}, \text{ usando } (\langle R\rangle)) \\
D_{5b} = & \{@_{k_2} p\} & \\
D_6 = & \{@_i \langle r\rangle k_2\} & (D_{4a} \text{ y } D_{5a}, \text{ usando (PARAM)}) \\
 & \vdots &
\end{array}
$$

Si comparamos D_{2a} y D_6 veremos que en el contexto de esta derivación son cláusulas equivalentes. Con esto queremos decir que podemos repetir los pasos llevados a cabo para derivar D_6 a partir de D_{1a} y D_{2a} para generar $D_{10} = \{@_i \langle r\rangle k_3\}$, y a partir de D_{10} derivar $D_{14} = \{@_i \langle r\rangle k_4\}$ y así sucesivamente. Claramente, la saturación del conjunto inicial $\{C\}$ constituye un conjunto infinito.

Veamos un poco más en detalle esta fórmula. Para empezar, es satisfacible sólo en modelos donde el mundo asociado a i esté relacionado con sí mismo y sólo con sí mismo, y donde además p sea verdadero en dicho mundo. En segundo lugar, esta fórmula tiene dos niveles de profundidad modal, esto significa, intuitivamente, que está *predicando* sobre mundos que estén a no más de dos pasos de distancia respecto a i.

Por otro lado, observemos qué sucede con k, k_2, k_3, etc. k_2 se *crea* a partir de $@_k \langle r\rangle p$, *bajo la hipótesis de que i y k son mundos distintos.* ¿Qué queremos decir con esto último? Es claro que sobre D_{4b} es posible aplicar paramodulación con D_{4a}. Podemos pensar que a partir de este hecho se abren dos ramas, aquella en que aplicamos paramodulación y aquella en que no. En la rama en que lo hacemos, estamos asumiendo que i y k son el mismo mundo; en la rama en que no, estamos asumiendo que son mundos distintos. Este último es el caso que estamos considerando. Ahora bien, k está a un paso de distancia de i, y si son mundos distintos, k_2 debe estar a dos pasos de distancia de i. Sin embargo, en D_6 al reemplazar k por i estamos *anulando* la hipótesis que dio origen a k_2. Es decir, k_2 es un nominal que se crea con la idea (equivocada) de que esté a dos pasos de distancia de i. Propagando este error, podemos pensar que k_3 se crea a partir de $@_{k_2} \langle r\rangle p$ con la hipótesis implícita de que se encuentra a tres pasos de distancia de i, que k_4 se crea suponiendo que está a cuatro pasos de distancia de i, y así siguiendo.

En definitiva, hemos visto un ejemplo en el que se obtiene un conjunto $\mathit{ClSet}^*(\varphi)$ infinito mediante la generación de sucesivos nominales nuevos cada vez más *alejados* de aquellos presentes en φ. El siguiente ejemplo nos muestra que también podemos obtener un número infinito de nominales sin necesidad de que éstos estén cada vez más alejados. Consideremos para ello esta variación de la fórmula anterior: $@_i([r](i \wedge (q \vee \langle r\rangle p)) \wedge \langle r\rangle p)$. Ésta también es satisfacible y permite generar la siguiente derivación:

$$
\begin{array}{rll}
C = & \{@_i([r](i \wedge (q \vee \langle r\rangle p)) \wedge \langle r\rangle p)\} & \\
D_{1a} = & \{@_i[r](i \wedge (q \vee \langle r\rangle p))\} & (C, \text{ usando } (\wedge)) \\
D_{1b} = & \{@_i \langle r\rangle p\} & \\
D_{2a} = & \{@_i \langle r\rangle k\} & (D_{1b}, \text{ usando } (\langle R\rangle)) \\
D_{2b} = & \{@_k p\} & \\
D_3 = & \{@_k (i \wedge (q \vee \langle r\rangle p))\} & (D_{1a} \text{ y } D_{2a}, \text{ usando } ([R])) \\
D_{4a} = & \{@_k i\} & (D_3, \text{ usando } (\wedge)) \\
D_{4b} = & \{@_k (q \vee \langle r\rangle p)\} &
\end{array}
$$

$$
\begin{array}{rll}
D_5 = \{@_k\, q, @_k\, \langle r\rangle p\} & & (D_{4b}, \text{ usando } (\vee)) \\
D_6 = \{@_k\, q, @_i\, \langle r\rangle p\} & & (D_{4a} \text{ y } D_5, \text{ usando (PARAM)}) \\
D_{7a} = \{@_k\, q, @_i\, \langle r\rangle k_2\} & & (D_6, \text{ usando } (\langle R\rangle)) \\
D_{7b} = \{@_k\, q, @_{k_2}\, p\} & & \\
D_8 = \{@_k\, q, @_{k_2}\, (i \wedge (q \vee \langle r\rangle p))\} & & (D_{1a} \text{ y } D_{7a}, \text{ usando } ([R])) \\
D_{9a} = \{@_k\, q, @_{k_2}\, i\} & & (D_8, \text{ usando } (\wedge)) \\
D_{9b} = \{@_k\, q, @_{k_2}\, (q \vee \langle r\rangle p)\} & & \\
D_{10} = \{@_k\, q, @_{k_2}\, q, @_{k_2}\, \langle r\rangle p\} & & (D_{9b}, \text{ usando } (\vee)) \\
D_{11} = \{@_k\, q, @_{k_2}\, q, @_i\, \langle r\rangle p\} & & (D_{9a} \text{ y } D_{10}, \text{ usando (PARAM)}) \\
D_{12a} = \{@_k\, q, @_{k_2}\, q, @_i\, \langle r\rangle k_3\} & & (D_{11}, \text{ usando } (\langle R\rangle)) \\
D_{12b} = \{@_k\, q, @_{k_2}\, q, @_{k_3}\, p\} & &
\end{array}
$$

$$\vdots$$

En este caso las cláusulas a comparar son D_{2a}, D_{7a} y D_{12a}, y conviene mirar cómo estas últimas se generan a partir de D_{1b}, D_6 y D_{11}. El problema de la infinitud del conjunto saturado ya no viene dado por la generación de una secuencia de nominales cada vez más alejados de i. Lo que este ejemplo muestra es cómo se pueden obtener infinitas apariciones de $@_i\, \langle r\rangle p$, y cómo cada una de ellas genera a su vez un nuevo nominal a un nivel de distancia respecto de i.

4.2 Restringiendo la generación de nuevos nominales: la regla ($\langle R\rangle$)

En la sección anterior presentamos dos formas distintas en que el cálculo de la Figura 3.1 puede generar infinitos nominales. En una de ellas, se crean nuevos nominales cada vez más alejados; mientras que en la otra, infinitas apariciones de una misma fórmula generan infinitos nominales equidistantes. En esta sección presentamos una variación del cálculo que evita este último comportamiento.

Consideremos la cláusula $C = \{@_i\, \langle r\rangle p, @_j\, (p \wedge q)\}$ y supongámosla parte de un conjunto de cláusulas. Dependiendo de la función de selección que utilicemos, una derivación a partir de C comenzará simplificando $@_i\, \langle r\rangle p$ ó $@_j\, (p \wedge q)$. Veamos dos posibles derivaciones, una empezando en cada fórmula. Por un lado, tenemos:

$$
\begin{array}{rll}
C = \{@_i\, \langle r\rangle p, @_j\, (p \wedge q)\} & & \\
C_{1a} = \{@_i\, \langle r\rangle k, @_j\, (p \wedge q)\} & & (C, \text{ usando } (\langle R\rangle)) \\
C_{1b} = \{@_k\, p, @_j\, (p \wedge q)\} & & \\
C_{2a} = \{@_i\, \langle r\rangle k, @_j\, p\} & & (C_{1a}, \text{ usando } (\wedge)) \\
C_{2b} = \{@_i\, \langle r\rangle k, @_j\, q\} & & \\
C_{3a} = \{@_k\, p, @_j\, p\} & & (C_{1b}, \text{ usando } (\wedge)) \\
C_{3b} = \{@_k\, p, @_j\, q\} & &
\end{array}
$$

mientras que por el otro obtenemos:

$$
\begin{array}{rll}
C = \{@_i\, \langle r\rangle p, @_j\, (p \wedge q)\} & & \\
C'_{1a} = \{@_i\, \langle r\rangle p, @_j\, p\} & & (C, \text{ usando } (\wedge)) \\
C'_{1b} = \{@_i\, \langle r\rangle p, @_j\, q\} & & \\
C'_{2a} = \{@_i\, \langle r\rangle k, @_j\, p\} & & (C'_{1a}, \text{ usando } (\langle R\rangle)) \\
C'_{2b} = \{@_k\, p, @_j\, p\} & & \\
C'_{3a} = \{@_i\, \langle r\rangle l, @_j\, q\} & & (C'_{1b}, \text{ usando } (\langle R\rangle)) \\
C'_{3b} = \{@_l\, p, @_j\, q\} & &
\end{array}
$$

Ahora bien, si comparamos las últimas cuatro cláusulas de cada caso observaremos que sólo difieren en el hecho de que en la segunda derivación se generó un nominal nuevo (l) mientras que en la primera se usó en todos los casos el mismo nominal (k).

De alguna forma, la segunda derivación está considerando modelos más complicados que la primera (e.g. modelos en los que i se relaciona con más de un mundo). También podemos verlo como que en el segundo caso estamos *perdiendo información*: las apariciones de $@_i \langle r \rangle p$ en C'_{1a} y C'_{1b} corresponden a la misma fórmula (puesto que ambas derivan de la aparición de $@_i \langle r \rangle p$ en C) y sin embargo ya no es posible relacionar las fórmulas que se derivan de ellas.

Lo que este ejemplo nos sugiere es que dada cualquier aparición de una fórmula $@_i \langle r \rangle \psi$, alcanza con tener un *único* nominal "testigo", que sea sucesor de i y que satisfaga ψ. El siguiente teorema formaliza esta cuestión.

Teorema 24. *Sea φ una fórmula de $\mathcal{H}^{NNF}(@)$ y sea j un nominal que no aparece en φ. φ es satisfacible si y sólo si para todo nominal i, toda relación r y toda fórmula $\psi \in \mathcal{H}^{NNF}(@)$, $\varphi[@_i \langle r \rangle \psi / (@_i \langle r \rangle j \wedge @_j \psi)]$ es satisfacible.*

Demostración. Veamos primero qué sucede con la implicación de izquierda a derecha. Sea, entonces, $\mathcal{M} = \langle W, R, V \rangle$ un conjunto en la signatura adecuada y definamos $\mathcal{M}' = \langle W, R, V' \rangle$ donde

$$\begin{array}{ll} V'(a) = V(a) & \text{para todo } a \in \mathsf{ATOM} \setminus j \\ V'(j) = w & \text{donde } w \text{ es un sucesor cualquiera de } V(i) \text{ (via } r^M\text{) que satisface} \\ & \psi \text{ o un elemento cualquiera de } W \text{ si tal mundo no existe} \end{array}$$

Claramente, como $\mathcal{M}$ es un modelo híbrido $\mathcal{M}'$ también lo es. En el contexto de esta demostración, si ψ es una fórmula, con $\psi^{\mathcal{M}}$ nos referiremos a la *extensión de ψ en $\mathcal{M}$*, es decir, al conjunto de elementos de W que satisfacen ψ (para $a \in \mathsf{ATOM}$, $a^{\mathcal{M}} = V(a)$). Vale notar que si ψ es una fórmula-@ (o una combinación booleana de fórmulas-@), entonces $\psi^{\mathcal{M}} = W$ o bien $\psi^{\mathcal{M}} = \emptyset$.

Comencemos por observar lo siguiente: $@_i \langle r \rangle \psi^{\mathcal{M}} = (@_i \langle r \rangle j \wedge @_j \psi)^{\mathcal{M}'}$. Veamos por qué. Si $@_i \langle r \rangle \psi^{\mathcal{M}} = \emptyset$, entonces no hay ningún sucesor apropiado de $V(i)$ que satisfaga ψ. Luego, por definición de $\mathcal{M}'$, $V(j)$ es un elemento arbitrario de W, con lo cual, o bien $V(j)$ no es un sucesor de $V(i)$ o bien lo es pero no satisface ψ. Cualquiera sea el caso vemos que $(@_i \langle r \rangle j \wedge @_j \psi)^{\mathcal{M}'} = \emptyset$. Consideremos el caso contrario, cuando $@_i \langle r \rangle \psi^{\mathcal{M}} = W$. Aquí, por definición de $\mathcal{M}'$, $V(j)$ es un sucesor de $V(i)$ que satisface ψ y, por lo tanto, $(@_i \langle r \rangle j \wedge @_j \psi)^{\mathcal{M}'} = W$.

Veamos que si $\mathcal{M}$ es un modelo que satisface φ, entonces $\varphi^{\mathcal{M}} = \varphi[@_i \langle r \rangle \psi / (@_i \langle r \rangle j \wedge @_j \psi)]^{\mathcal{M}'}$ (φ no es necesariamente una fórmula-@). Procedemos por inducción en la complejidad de φ.

En el caso base, si $\varphi \in \mathsf{ATOM}$, entonces sabemos que $\varphi \neq j$ y que $\varphi = \varphi[@_i \langle r \rangle \psi / (@_i \langle r \rangle j \wedge @_j \psi)]$. Por lo tanto, $\varphi^{\mathcal{M}} = \varphi^{\mathcal{M}'}$.

Supongamos ahora, como hipótesis inductiva, que para toda φ' subfórmula propia de φ, $\varphi'^{\mathcal{M}} = \varphi'[@_i \langle r \rangle \psi / (@_i \langle r \rangle j \wedge @_j \psi)]^{\mathcal{M}'}$. Los casos en que φ es de la forma $\neg\varphi'$, $\varphi_1 \wedge \varphi_2$ o $\varphi_1 \vee \varphi_2$ son triviales. Veamos el resto de los casos más en detalle:

Caso $\varphi = @_n \varphi'$. Aquí tenemos que considerar por separado el caso en que $n = i$ y $\varphi' = \langle r \rangle \psi$. Cuando esto sucede, como ya mostramos más arriba, $@_i \langle r \rangle \psi^{\mathcal{M}} = (@_i \langle r \rangle j \wedge @_j \psi)^{\mathcal{M}'}$. En caso contrario, alcanza con tener en cuenta la hipótesis inductiva y que $n \neq j$.

Caso $\varphi = \langle s \rangle \varphi'$. Por hipótesis inductiva, $\varphi'^{\mathcal{M}} = \varphi'[@_i \langle r \rangle \psi / (@_i \langle r \rangle j \wedge @_j \psi)]^{\mathcal{M}'}$. Como no puede suceder que j sea una subfórmula de φ', $\varphi^{\mathcal{M}} = \varphi[@_i \langle r \rangle \psi / (@_i \langle r \rangle j \wedge @_j \psi)]^{\mathcal{M}'}$.

Caso $\varphi = [s] \varphi'$. La demostración es equivalente al caso anterior.

Nos resta ver qué sucede con la implicación en el otro sentido. Empecemos por notar que para cualquier par de nominales i y j, $(@_i \langle r \rangle j \wedge @_j \psi) \rightarrow (@_i \langle r \rangle \psi)$ es una tautología. Como φ se encuentra en forma normal negativa, $@_i \langle r \rangle \psi$ sólo puede aparecer en ella con polaridad positiva (i.e. $\neg @_i \langle r \rangle \psi$ no puede ser una subfórmula de φ). Luego, $\varphi[@_i \langle r \rangle \psi / (@_i \langle r \rangle j \wedge @_j \psi)] \rightarrow \varphi$ es una tautología, con lo cual si $\varphi[@_i \langle r \rangle \psi / (@_i \langle r \rangle j \wedge @_j \psi)]$ es satisfacible, φ también lo es. □

Con este resultado, podemos proponer una regla alternativa para tratar diamantes. Para ello, dividamos NOM en dos conjuntos (infinitos) disjuntos NOM_i y NOM_c, y llamemos $\mathcal{H}_0^{NNF}(@)$ al conjunto de fórmulas de $\mathcal{H}^{NNF}(@)$ en el que sólo aparecen nominales de NOM_i. De aquí en más supondremos, sin perder generalidad, que el cálculo se utiliza sólo sobre fórmulas de $\mathcal{H}_0^{NNF}(@)$ y que todo nominal que aparece en $\mathit{ClSet}^*(\varphi)$ pero no en φ pertenece a NOM_c.

Definamos, además, $\mathcal{H}^{NNF}_{@\Diamond}(@) = \{@_i \langle r\rangle\psi \mid i \in \mathsf{NOM}, r \in \mathsf{REL}, \psi \in \mathcal{H}^{NNF}_0(@), \psi \notin \mathsf{NOM}\}$, es decir, $\mathcal{H}^{NNF}_{@\Diamond}(@)$ es el conjunto de fórmulas-@ de $\mathcal{H}^{NNF}(@)$ que pueden ser premisa de la regla $(\langle R\rangle)$. Finalmente, sea $nom(x) : \mathcal{H}^{NNF}_{@\Diamond}(@) \to \mathsf{NOM}_c$ una función *inyectiva* cualquiera. Definimos la regla alternativa $(\langle R\rangle')$ como:

$$(\langle R\rangle') \quad \frac{Cl \cup \{@_t \langle r\rangle\varphi\}}{\begin{array}{c} Cl \cup \{@_t \langle r\rangle n\} \\ Cl \cup \{@_n \varphi\} \end{array}} \quad \begin{array}{l} \text{con } n = nom(@_t \langle r\rangle\varphi) \\ \text{y } \varphi \notin \mathsf{NOM} \end{array}$$

El Teorema 24 garantiza la consistencia de esta regla. Lo que necesitamos ver es que usar esta regla en reemplazo de la original no afecta la completitud refutacional del cálculo. Para ello, alcanza con observar que la argumentación del caso de la regla $(\langle R\rangle)$ en la demostración del Lema 17 también se aplica a la regla $(\langle R\rangle')$.

4.3 Restringiendo la generación de fórmulas repetidas: la regla (PARAM)

Volvamos a considerar la derivación a partir de la fórmula $@_i([r](i \wedge \langle r\rangle p) \wedge \langle r\rangle p)$ que vimos en la Sección 4.1, pero usando la regla $(\langle R\rangle')$ propuesta en la sección anterior. Lo primero que podemos observar es que la derivación no se modifica sustancialmente y que se siguen generando infinitos nominales cada vez más alejados de i.

Miremos más en detalle la cláusula D_6 de dicha derivación, que como ya mencionamos, cumple un rol principal en la derivación *cíclica* que genera un conjunto infinito de cláusulas. Esta cláusula se genera a partir de la paramodulación entre las cláusulas D_{4a} y D_{5a}. La fórmula $@_i \langle r\rangle k_2$, sintetizada en dicha operación, es el resultado de asumir que i y k representan el mismo mundo. Ahora bien, desde el punto de vista de la regla $(\langle R\rangle')$, ¿qué representa k_2 en $D_{5a} = \{@_k \langle r\rangle k_2\}$? k_2 es un mundo introducido bajo la suposición de que k se relaciona con él, y que en él p es verdadero. Aquí es importante observar que $nom(@_k \langle r\rangle p) = k_2$. Cuando sintetizamos D_6, k_2 pasa a cumplir un rol adicional: ahora también representa el mundo relacionado con i y tal que p es verdadero en él. Sin embargo, el cálculo tenía *reservado* otro nominal para cumplir esta función: $nom(@_i \langle r\rangle p) = k$.

¿A dónde queremos llegar con esta observación? En la sección anterior vimos que es posible asignar, usando la función *nom*, un *rol* preciso a cada nominal que se crea durante el cálculo y que esto permite que el cálculo pueda limitarse a considerar modelos más simples. El análisis que acabamos de ver muestra cómo al hacer paramodulación sobre relaciones que involucran un nominal de NOM_c, también estamos considerando modelos potencialmente más complejos de lo necesario.

Esta idea nos motiva a buscar una forma de tratar las igualdades sobre relaciones de forma tal de aprovechar lo que *sabemos* sobre los nominales que son introducidos durante el cálculo. Lo que proponemos es incorporar una regla especial para la paramodulación sobre fórmulas como la de la cláusula C_6. Esta regla, (PARAM-REL), se presenta en la Figura 4.1. Ya no nos alcanza con pedir que $nom(x)$ sea una función inyectiva; en este caso necesitamos que sea un tipo especial de función a la que llamamos *distribuidor de nominales*:

Definición 4.2. Una función $f : \mathcal{H}^{NNF}_{@\Diamond}(@) \to \mathsf{NOM}_c$ es un *distribuidor de nominales* si y sólo si

1. f es biyectiva,
2. la clausura transitiva de la relación dada por Rij si y sólo si $j = f(@_i \langle r\rangle\varphi)$ para algún $\langle r\rangle\varphi$ es un orden parcial, y
3. $i \succ j$ si y sólo si $nom(@_i \langle r\rangle\varphi) \succ nom(@_j \langle r\rangle\varphi)$

Las primeras dos condiciones simplificarán los argumentos que usaremos en esta sección y en las que siguen. Lo que hacen es garantizar que todos los elementos de NOM_c puedan aparecer en

$ClSet^*(\varphi)$: la primera condición garantiza sobreyectividad, mientras que la segunda evita *ciclos* como en $i = nom(@_i \langle r \rangle \psi)$. La tercer condición es necesaria, como veremos en la Sección 4.3.2, para garantizar la completitud del cálculo.

$$(\wedge) \quad \frac{Cl \cup \{@_t (\varphi_1 \wedge \varphi_2)\}}{\begin{array}{c} Cl \cup \{@_t \varphi_1\} \\ Cl \cup \{@_t \varphi_2\} \end{array}} \qquad (\vee) \quad \frac{Cl \cup \{@_t (\varphi_1 \vee \varphi_2)\}}{Cl \cup \{@_t \varphi_1, @_t \varphi_2\}}$$

$$(\text{RES}) \quad \frac{Cl_1 \cup \{@_t \varphi\} \quad Cl_2 \cup \{@_t \neg\varphi\}}{Cl_1 \cup Cl_2}$$

$$([R]) \quad \frac{Cl_1 \cup \{@_t \langle r \rangle s\} \quad Cl_2 \cup \{@_t [r]\varphi\}}{Cl_1 \cup Cl_2 \cup \{@_s \varphi\}} \qquad (\langle R \rangle') \quad \frac{Cl \cup \{@_t \langle r \rangle \varphi\}}{\begin{array}{c} Cl \cup \{@_t \langle r \rangle n\} \\ Cl \cup \{@_n \varphi\} \end{array}} \quad \begin{array}{l} \text{con } \varphi \notin \mathsf{NOM} \text{ y} \\ n = nom(@_t \langle r \rangle \varphi) \end{array}$$

$$(@) \quad \frac{Cl \cup \{@_t @_s \varphi\}}{Cl \cup \{@_s \varphi\}} \qquad (\text{REF}) \quad \frac{Cl \cup \{@_t \neg t\}}{Cl} \qquad (\text{SYM}) \quad \frac{Cl \cup \{@_s t\}}{Cl \cup \{@_t s\}} \quad \text{si } t \succ s$$

$$(\text{PARAM'}) \quad \frac{Cl_1 \cup \{@_s t\} \quad Cl_2 \cup \{\varphi(s)\}}{Cl_1 \cup Cl_2 \cup \{\varphi(s/t)\}}$$

si $s \succ t$, $\varphi(s) \succ @_s t$, y cuando $\varphi(s) = @_i \langle r \rangle j$, entonces $j \in \mathsf{NOM}_i$, o $t \in \mathsf{NOM}_i$ y $j = s$

$$(\text{PARAM-REL}) \quad \frac{Cl_1 \cup \{@_s t\} \quad Cl_2 \cup \{@_s \langle r \rangle n\}}{\begin{array}{c} Cl_1 \cup Cl_2 \cup \{@_t \langle r \rangle k\} \\ Cl_1 \cup Cl_2 \cup \{@_n k\} \end{array}}$$

si $s \succ t$ y $n \in \mathsf{NOM}_c$, y donde para algún φ, $n = nom(@_s \varphi)$, y $k = nom(@_t \varphi)$

Restricciones:

- Si $C = C' \cup \{\varphi\}$ es la premisa principal, entonces o bien $S(C) = \{\varphi\}$ o, en caso contrario, $S(C) = \emptyset$ y $\{\varphi\} \succ C'$
- Si $D = D' \cup \{\psi\}$ es la premisa auxiliar, entonces $\{\psi\} \succ D'$ y $S(D) = \emptyset$
- $nom(x)$ es un *distribuidor de nominales*

(tanto φ como ψ representan la fórmula que participa en la inferencia)

Figura 4.1: Reglas de resolución para $\mathcal{H}(@)$ con orden, selección y control de nominales

Nos resta ver que los cambios introducidos a las reglas de paramodulación preservan consistencia y completitud. Las demostraciones son en este caso un poco más trabajosas que aquellas para la regla $(\langle R \rangle')$. Por ello, presentamos en detalle la demostración de consistencia en la Sección 4.3.1 y de completitud en la Sección 4.3.2. Finalmente, en la Sección 4.3.3 mostramos que la definición de distribuidor de nominales no es demasiado restrictiva exhibiendo explícitamente un ejemplo constructivo de una función de este tipo. De todas formas, de aquí en más supondremos que tenemos un distribuidor de nominales fijo $nom(x)$, asociado a $\succ$.

4.3.1 Consistencia

Para probar la consistencia del cálculo procederemos de la siguiente forma: primero veremos que dado un distribuidor de nominales $nom(x)$, todo modelo en la signatura $\langle \mathsf{PROP}, \mathsf{NOM}_i, \mathsf{REL} \rangle$ puede ser convertido a un modelo en la signatura $\langle \mathsf{PROP}, \mathsf{NOM}_i \cup \mathsf{NOM}_c, \mathsf{REL} \rangle$ que sea *compatible* con $nom(x)$, para cierto criterio de compatibilidad adecuado; luego veremos que cualquier modelo compatible con $nom(x)$ de una fórmula $\varphi \in \mathcal{H}_0^{NNF}(@)$ es un modelo de $ClSet^*(\varphi)$.

Lo primero que debería cumplir un modelo $M = \langle W, R, V \rangle$ para ser *compatible* con $nom(x)$ sería que, si $i = nom(@_s \langle r \rangle \psi)$ y $M \models @_s \langle r \rangle \psi$, entonces $M \models @_s \langle r \rangle i$ y $M \models @_i \psi$. Un modelo como este garantizaría la consistencia de la regla $(\langle R \rangle')$. Lo segundo que necesitamos de un modelo

compatible con $nom(x)$ es que, si además $j = nom(@_t \langle r \rangle \psi)$ y $M \models @_s t$, entonces $M \models @_i j$. Con esta condición, podemos garantizar la consistencia de (PARAM-REL). Lo último que le pediremos a un modelo compatible con $nom(x)$ es que si $k = nom(@_n \langle r' \rangle \psi')$, y $M \not\models @_n \langle r' \rangle \psi'$, entonces $V(k)$ esté *aislado* de mundos como $V(i)$ o $V(j)$.

A continuación veremos cómo armar, un modelo compatible con $nom(x)$. Para ello, nos valdremos de una función que nos permita garantizar, semánticamente, la segunda condición de las que presentamos más arriba.

Definición 4.3. Dado un modelo $M = \langle W, R, V \rangle$ en la signatura $\langle \mathsf{PROP}, \mathsf{NOM}_i, \mathsf{REL} \rangle$, y un distribuidor de nominales $nom(x)$, decimos que $M' = \langle W', R, V' \rangle$, un modelo en la signatura $\langle \mathsf{PROP}, \mathsf{NOM}, \mathsf{REL} \rangle$, es la *extensión de M compatible con $nom(x)$* si $W' = W \cup \{w_t\}$ para algún w_t que no aparezca en W y existe una función $f : W \times \mathsf{REL} \times \mathcal{H}_0^{NNF}(@) \to W$ que cumple que, para todo $w \in W$, $r \in \mathsf{REL}$ y $\psi \in \mathcal{H}_0^{NNF}(@)$, $M, w \models \langle r \rangle \psi$ implica $w R_r f(w, r, \psi)$ y $M, f(w, r, \psi) \models \psi$, tal que

$$V'(i) = \begin{cases} V(i) & \text{si } i \in \mathsf{NOM}_i \\ f(V'(j), r, \psi) & \text{si } i = nom(@_j \langle r \rangle \psi) \text{ y } M, V'(j) \models \langle r \rangle \psi \\ w_t & \text{si } i = nom(@_j \langle r \rangle \psi) \text{ y } M, V'(j) \not\models \langle r \rangle \psi \end{cases}$$

En la definición anterior, w_t es un *mundo trampa* (por analogía con un *estado trampa* en un autómata finito), en el sentido de que para toda relación $R_r \in R$, no existe ningún $w \in W'$ tal que $w R_r w_t$ ni $w_t R_r w$. Este mundo trampa *aísla* todos los nominales de NOM_c a los que no se les puede asignar un mundo compatible con $nom(x)$.

Proposición 25. *Para toda fórmula $\varphi \in \mathcal{H}_0^{NNF}(@)$ y todo modelo M tal que $M \models \mathrm{ClSet}(\varphi)$, si M' es la extensión de M compatible con $nom(x)$, entonces $M' \models \mathrm{ClSet}(\varphi)$.*

Lema 26. *Para toda fórmula $\varphi \in \mathcal{H}_0^{NNF}(@)$, si φ tiene un modelo M, y M' es la extensión de M compatible con $nom(x)$, entonces M' es modelo de todo subconjunto* finito *de* $\mathrm{ClSet}^*(\varphi)$.

Demostración. Todo subconjunto finito de $ClSet^*(\varphi)$ está contenido en algún prefijo de cualquier secuencia de derivación de $ClSet^*(\varphi)$. Lo que veremos es que si φ es satisfecha por M (y por lo tanto, por M'), entonces $M' = \langle W, R, V \rangle$ es modelo de cualquier prefijo de secuencia de derivación de $ClSet^*(\varphi)$. Esto lo podemos comprobar por inducción en la longitud de los prefijos. El caso base es trivial, ya que si L es un prefijo de longitud 1, entonces $L = ClSet(\varphi)$.

Para el paso inductivo, supongamos que si M es modelo de φ, entonces M' es modelo de todo prefijo de longitud n de cualquier secuencia de derivación de $ClSet^*(\varphi)$. Consideremos $L.C$, un prefijo de longitud $n+1$, donde L es un prefijo de longitud n y C es una cláusula que se obtiene usando alguna de las reglas del cálculo sobre cláusulas de L. Por hipótesis inductiva, suponemos que M' satisface L. Las reglas del cálculo original son consistentes en el sentido más fuerte, ya que cualquier modelo de las premisas es también modelo de los consecuentes. Con la regla ($\langle R \rangle'$) no sucede lo mismo; en este caso, como vimos en la Sección 4.2, si existe algún modelo que satisface las premisas, entonces podemos estar seguros de que existe algún otro modelo (potencialmente distinto) que satisface tanto las premisas como los consecuentes. Es por ello que necesitamos verificar que, en este caso, M' es un modelo que si satisface las premisas de dicha regla también satisface sus consecuentes.

Caso ($\langle R \rangle'$) Supongamos que en L existe $C_1 = \{@_i \langle r \rangle \psi\} \cup C_1'$ y que $C = \{@_i \langle r \rangle j\} \cup C_1'$ ó $C = \{@_j \psi\} \cup C_1'$, con $\psi \notin \mathsf{NOM}$ y $j = nom(@_i \langle r \rangle \psi)$. Si $M' \models C_1'$ entonces en cualquiera de los dos casos $M' \models C$. Supongamos, en cambio, que $M' \not\models C_1'$, con lo cual, $M' \models @_i \langle r \rangle \psi$. En este caso, de acuerdo a la Definición 4.3, $V(j) = f(V(i), r, \psi)$ para alguna función f apropiada que garantiza $M' \models @_i \langle r \rangle j$ y $M' \models @_j \psi$.

Veamos por último qué sucede si C es un consecuente de la regla (PARAM-REL).

Caso (PARAM-REL) Supongamos que en L existen $C_1 = \{@_s \langle r \rangle i\} \cup C_1'$ y $C_2 = \{@_s t\} \cup C_2'$, y que $C = \{@_t \langle r \rangle j\} \cup C_1' \cup C_2'$ ó $C = \{@_i j\} \cup C_1' \cup C_2'$, con $i = nom(@_s \langle r \rangle \psi)$ y $j = nom(@_t \langle r \rangle \psi)$ para algún $\psi \notin \mathsf{NOM}$. Si $M' \models C_1'$ ó $M' \models C_2'$, es claro que $M' \models C$. Supongamos, en

cambio, que $M' \not\models C_1$ y $M' \not\models C_2$, con lo cual $M' \models @_s \langle r \rangle i$ y $M' \models @_s t$. Llamemos w_t al *mundo trampa* de M'; lo que hay que observar aquí es que no puede suceder que $V(i) = w_t$ ya que ningún mundo se relaciona con w_t y sabemos que $M' \models @_s \langle r \rangle i$. Luego, de acuerdo a la Definición 4.3, $V(i) = f(V(s), r, \psi)$ para alguna función f apropiada y, además, $M' \models @_s \langle r \rangle \psi$. Como, además, $M' \models @_s t$, tenemos $M' \models @_t \langle r \rangle \psi$. De esto último se desprende que $V(j) \neq w_t$ ya que $V(j) = f(V(t), r, \psi) = f(V(s), r, \psi) = V(i)$. Concluimos, entonces, que $M' \models @_i j$ y $M' \models @_t \langle r \rangle j$, con lo cual, $M \models C$. □

Teorema 27. *Si $\varphi \in \mathcal{H}_0^{NNF}(@)$ es satisfacible,* $\mathrm{ClSet}^*(\varphi)$ *también lo es.*

Demostración. Supongamos que φ fuera satisfacible pero $ClSet^*(\varphi)$ no. Por compacidad, debería existir un conjunto finito $N \subset ClSet^*(\varphi)$ tal que N no es satisfacible. Ahora bien, por la Proposición 25, si M es modelo de φ, entonces M', la extensión de M compatible con $nom(x)$ también es modelo de φ. Luego, por el Lema 26, M' debería satisfacer N. Este es un absurdo que viene de suponer que φ puede ser satisfacible y no así $ClSet^*(\varphi)$. □

Corolario 28. *El cálculo de la Figura 4.1 es consistente.*

4.3.2 Completitud refutacional

Lo primero que conviene observar es que la restricción sobre el orden de los nominales en la Definición 4.2 permite garantizar que los consecuentes que se obtienen al aplicar la regla (PARAM-REL) son menores que su premisa principal. Teniendo esto en cuenta no es difícil modificar la demostración de completitud del Capítulo 3 para que considere la regla (PARAM-REL) (la incorporación de la regla ($\langle R \rangle'$) a dicha demostración se discutió en la Sección 4.2). En lo que sigue, mostramos los pasos más importantes. Comencemos con algunas observaciones sobre el nuevo cálculo.

Proposición 29. *Para toda fórmula $\varphi \in \mathcal{H}_0^{NNF}(@)$, si $\{@_i \langle r \rangle j\} \cup C \in \mathrm{ClSet}^*(\varphi)$ y $j \in \mathsf{NOM}_c$, entonces existe ψ tal que $j = nom(@_i \langle r \rangle \psi)$.*

Corolario 30. *Si $\{\{@_i \langle r \rangle k\} \cup C, \{@_j \langle r \rangle k\} \cup D\} \subset \mathrm{ClSet}^*(\varphi)$ y $k \in \mathsf{NOM}_c$, entonces $i = j$.*

Proposición 31. *Para toda fórmula $\varphi \in \mathcal{H}_0^{NNF}(@)$, si $\{@_i j\} \cup C \in \mathrm{ClSet}^*(\varphi)$ y $i, j \in \mathsf{NOM}_c$, entonces existen s, t, r y ψ tales que $i = nom(@_s \langle r \rangle \psi)$ y $j = nom(@_t \langle r \rangle \psi)$.*

Proposición 32. *Para toda fórmula $\varphi \in \mathcal{H}_0^{NNF}(@)$, si existen dos cláusulas C y D pertenecientes a $\mathrm{ClSet}^*(\varphi)$ tales que $dist_S^{\succ}(C) = @_i \langle r \rangle j$ y $dist_S^{\succ}(D) = @_j k$, con $i, j \in \mathsf{NOM}_c$, entonces en $\mathrm{ClSet}^*(\varphi)$ también debe existir una cláusula E tal que $dist_S^{\succ}(E) = @_i n$ con $i \succ n$, para algún nominal n.*

En la proposición anterior, hay que tener en cuenta que, si bien es posible, no necesariamente debe suceder que para algún ψ se cumpla $j = nom(@_i \langle r \rangle \psi)$ y $k = nom(@_n \langle r \rangle \psi)$. Lo que esta proposición nos dice es que si j es un nominal de NOM_c y aparece en una igualdad, entonces necesariamente tiene que existir alguna igualdad que reduzca a i.

La Proposición 32 es la base de la demostración de la versión equivalente al Lema 18. Sin embargo, éste predicaba sobre cualquier conjunto de cláusulas, mientras que la Proposición 32 se puede aplicar sólo sobre un conjunto saturado con respecto a las reglas. Esto no es grave; dicho lema es usado sólo en la demostración del Teorema 22, que a su vez, se utiliza en el Teorema 23 sobre un conjunto de este tipo.

Lema 33. *Sea φ una fórmula cualquiera de $\mathcal{H}_0^{NNF}(@)$, y sea $C \in \mathrm{ClSet}^*(\varphi)$ un contraejemplo de $I_{\mathrm{ClSet}^*(\varphi)}$. Si $dist_S^{\succ}(C)$ es una fórmula de la forma $@_i \langle r \rangle j$ que no se encuentra en forma reducida, entonces existe una cláusula D reductiva para una igualdad tal que la aplicación de la regla (PARAM') ó (PARAM-REL) con C y D como premisas genera un consecuente que es también un contraejemplo de $I_{\mathrm{ClSet}^*(\varphi)}$.*

Demostración. Si $@_i \langle r \rangle j$ no se encuentra en forma reducida, entonces o bien $i\sigma_{I_C} \neq i$ o bien $j\sigma_{I_C} \neq j$. Si sucede lo primero, entonces es claro que debe existir en $ClSet^*(\varphi)$ una cláusula D reductiva para $@_i\, k$ tal que $C \succ D$. Claramente, D puede ser premisa auxiliar en la aplicación de (PARAM') ó (PARAM-REL), dependiendo de si $j \in \mathsf{NOM}_i$ o $j \in \mathsf{NOM}_c$, respectivamente. Si por el contrario, $i\sigma_{I_C} = i$, entonces debe existir en $ClSet^*(\varphi)$ una cláusula D reductiva para $@_j\, k$. Ahora bien, si $k \in \mathsf{NOM}_i$, entonces D puede ser premisa auxiliar de la regla (PARAM'); en caso contrario, cumplimos con las hipótesis de la Proposición 32, en cuyo caso tenemos garantizada la existencia de una cláusula que puede ser premisa auxiliar de la regla (PARAM-REL).

Necesitamos ver que en cualquier caso, alguno de los consecuentes constituye un contraejemplo de $I_{ClSet^*(\varphi)}$. El caso de la regla (PARAM') está cubierto en la demostración del Lema 18. Supongamos, entonces, que tenemos un cláusula D reductiva para $@_i\, k$, y sean $\{@_k \langle r \rangle n\} \cup C' \cup D'$ y $\{@_j\, n\} \cup C' \cup D'$ los consecuentes de aplicar la regla (PARAM-REL), donde $C' = C \setminus dist^{\succ}_S(C)$ y $D' = D \setminus dist^{\succ}_S(D)$. Nos alcanza con ver que si $@_k \langle r \rangle n$ no es un contraejemplo de $I_{ClSet^*(\varphi)}$, entonces $@_j\, n$ necesariamente lo es. Esto es fácil de comprobar, ya que si ambas fórmulas fueran verdaderas en $I_{ClSet^*(\varphi)}$, entonces $@_i \langle r \rangle j$ también debería serlo, puesto que estamos suponiendo que $@_i\, k$ lo es. □

Usando el Lema 33, podemos demostrar un equivalente al Teorema 22, realizando sólo cambios triviales.

Teorema 34. *El cálculo de la Figura 4.1 es refutacionalmente completo.*

4.3.3 Un distribuidor de nominales concreto

Nos interesa ver que la Definición 4.2 no es demasiado estricta, es decir, que puede ser satisfecha por alguna función concreta. Para ello, mostramos que para cualquier lenguaje híbrido podemos extender el conjunto de nominales de forma adecuada, de manera que la construcción de un distribuidor de nominales se vuelva trivial.

Definición 4.4. Sea $S^0 = \langle \mathsf{PROP}, \mathsf{NOM}^0, \mathsf{REL} \rangle$, una signatura sobre la cual se define el lenguaje híbrido L^0. Definimos inductivamente $S^{n+1} = \langle \mathsf{PROP}, \mathsf{NOM}^{n+1}, \mathsf{REL} \rangle$, donde $\mathsf{NOM}^{n+1} = \{t_{@_i \langle r \rangle \psi} \mid i \in \mathsf{NOM}^n, \text{ y } \langle r \rangle \psi \in L^0\}$ y decimos que L^{n+1} es el lenguaje híbrido definido sobre S^{n+1}. Finalmente, asumimos $\mathsf{NOM}_i = \mathsf{NOM}^0$, $\mathcal{H}_0^{NNF}(@) = L^0$, $\mathsf{NOM} = \bigcup_{n \geq 0} \mathsf{NOM}^n$, $\mathsf{NOM}_c = \mathsf{NOM} \setminus \mathsf{NOM}_i$ y $\mathcal{H}^{NNF}(@) = \bigcup_{n \geq 0} L^n$.

Según esta definición, nuestro lenguaje híbrido se construye a partir de un lenguaje básico (L^0), y una serie de lenguajes de *orden creciente* ($L^1, L^2, \ldots$) en los que aparecen términos de los lenguajes de orden inferior. En lo que sigue, nos será útil la función $ord(x) : \mathcal{H}^{NNF}(@) \to \mathbb{N}$ que verifica $ord(\varphi) = c$ si y sólo si $\varphi \in L^c$.

Definición 4.5. Sea $\succ^0$ un orden como el dado por la Definición 3.9, basado en $>^0$, alguna precedencia apropiada definida sobre S^0. Definimos $>^{n+1}$ como la precedencia sobre S^{n+1} que incluye a $>^n$ y verifica que, para todo $i, j \in \bigcup_{1 \leq k \leq n+1} \mathsf{NOM}^k$, $i >^{n+1} j$ si y sólo si:

- $i \in \mathsf{NOM}^c, j \in \mathsf{NOM}^{c'}$ y $c > c'$, ó
- $i, j \in \mathsf{NOM}_c, i = t_\varphi, j = t_\psi$ y $\varphi \succ^n \psi$

De forma similar, $\succ^{n+1}$ es el orden definido de acuerdo a la Definición 3.9 que utiliza a $>^{n+1}$ como precedencia. Finalmente, sea $\succ^\omega$ la relación dada por $\varphi \succ^\omega \psi$ si y sólo si $\varphi \succ^n \psi$ donde $n = max(ord(\varphi), ord(\psi))$.

Proposición 35. *Para todo par de fórmulas $\varphi, \psi \in \mathcal{H}^{NNF}(@)$ se cumple:*

- *si $\varphi \succ^n \psi$, entonces, para todo $m > n$, $\varphi \succ^m \psi$, y*
- *si $\varphi \succ^m \psi$ y $m > n$, donde $n = max(ord(\varphi), ord(\psi))$, entonces $\varphi \succ^n \psi$*

Valiéndonos de esto es sencillo ver que:

Proposición 36. *$\succ^\omega$ es un orden adecuado sobre $\mathcal{H}^{NNF}(@)$.*

Ahora sí, estamos en condiciones de definir trivialmente un distribuidor de nominales para este lenguaje.

Proposición 37. *$nom(x) : \mathcal{H}^{NNF}_{@\Diamond}(@) \to \mathsf{NOM}_c$, la función dada por $nom(@_i \langle r\rangle\psi) = t_{@_i \langle r\rangle\psi}$, es un distribuidor de nominales.*

Demostración. Para ver esto, debemos probar que $nom(x)$ satisface las tres condiciones de la Definición 4.2. El primero, que $nom(x)$ es biyectiva, es trivial de verificar. Para el segundo, alcanza con observar que si $j = nom(@_i \langle r\rangle\psi)$, entonces $ord(j) = ord(i) + 1$, lo cual nos libra de circularidades. Veamos el tercer punto en más detalle.

Lo que debemos probar es que $i \succ^\omega j$ sii $nom(@_i \langle r\rangle\psi) \succ^\omega nom(@_j \langle r\rangle\psi)$. Lo primero que se puede observar es que $ord(i) > ord(j)$ sii $ord(nom(@_i \langle r\rangle\psi)) > ord(nom(@_i \langle r\rangle\psi))$. Supongamos, en cambio que $ord(i) = ord(j)$, con lo cual $ord(nom(@_i \langle r\rangle\psi)) = ord(nom(@_j \langle r\rangle\psi))$. Por ser $\succ^\omega$ un orden de reescritura, sabemos que $i \succ^\omega j$ si y sólo si $@_i \langle r\rangle\psi \succ^\omega @_j \langle r\rangle\psi$, y esto último sucederá si y sólo si $t_{@_i \langle r\rangle\psi} \succ^\omega t_{@_j \langle r\rangle\psi}$, ó, lo que es lo mismo, si $nom(@_i \langle r\rangle\psi)) \succ^\omega nom(@_j \langle r\rangle\psi))$. □

4.4 Terminación

En esta sección veremos que si usamos un refinamiento de un orden adecuado, el número de fórmulas distintas que el cálculo de la Figura 4.1 puede generar es finito. Con finitas fórmulas sólo es posible armar un conjunto finito de cláusulas distintas; con lo cual, el conjunto saturación debe ser necesariamente finito. Comencemos, entonces, con una observación simple.

Proposición 38. *Sea φ una fórmula de $\mathcal{H}^{NNF}_0(@)$; para toda fórmula $\psi \in \mathrm{ClSet}^*(\varphi)$ se cumple $size(\varphi) \geq size(\psi)$.*

Al igual que en la Definición 3.9, $size(\varphi)$ representa la cantidad de operadores (incluyendo los de aridad cero) de φ. Lo que esta proposición nos dice es que si $ClSet^*(\varphi)$ es infinito, esto no puede deberse a la presencia de fórmulas infinitamente grandes. El origen de la infinitud tendremos que buscarlo en la presencia de infinitos nominales distintos. Lo que veremos a continuación es que los dos ejemplos que vimos en la Sección 4.1 ilustran todas las maneras en que pueden generarse infinitos nominales y que los mecanismos propuestos para remediar esto efectivamente funcionan. Para ello nos valdremos de una medida de *distancia* entre un nominal cualquiera y algún nominal de NOM_i.

Definición 4.6. Sea $level(\cdot) : \mathsf{NOM} \to \mathbb{N}$ la función definida como:

$$level(i) = \begin{cases} 0 & \text{si } i \in \mathsf{NOM}_i \\ level(j) + 1 & \text{si } i = nom(@_j \langle r\rangle\varphi) \end{cases}$$

Claramente, $n \in \mathsf{NOM}_i$ si y sólo si $level(n) = 0$. Ahora podemos caracterizar fácilmente los dos tipos de infinitud. La primera derivación se caracteriza por el hecho de que $level(n)$, cuando n es un nominal que aparece en $ClSet^*(\varphi)$, no está acotada; la segunda, por el hecho de que existe un conjunto infinito N de nominales que aparecen en $ClSet^*(\varphi)$ para el cual se cumple $level(n) = c$ si $n \in N$, para algún $c > 0$. Es claro que si no se cumple ninguna de estas dos condiciones, el conjunto de nominales no puede ser infinito.

Como ya dijimos, la función *level* nos da una medida de *profundidad* de nominales. Esta idea jugará un papel importante en lo que sigue, y de hecho influye en la noción de orden con la que trabajaremos.

Definición 4.7. Un orden $\succ$ y un distribuidor de nominales $nom(x)$ son *adecuados para terminación* si $\succ$ es un orden *adecuado* y además verifica que para todo par de nominales i y j $level(i) > level(j)$ implica $i \succ j$, donde $level(x)$ está definido en función de $nom(x)$.

Se puede ver con facilidad que el orden $\succ^\omega$ de la Definición 4.5, es un orden adecuado para terminación. De aquí en más, asumimos que $\succ$ y $nom(x)$ son, respectivamente, un orden y un distribuidor de nominales de este tipo. Veamos, a continuación, algunas características del cálculo teniendo en cuenta la noción de profundidad recién presentada.

Proposición 39. *Para toda fórmula $\varphi \in \mathcal{H}_0^{NNF}(@)$ y toda cláusula $\{@_i \langle r\rangle j\} \cup C \in \mathrm{ClSet}^*(\varphi)$, o bien $level(j) = 0$ o bien $level(j) = level(i) + 1$.*

Proposición 40. *Para toda fórmula $\varphi \in \mathcal{H}_0^{NNF}(@)$ y toda fórmula ψ que aparezca en $\mathrm{ClSet}^*(\varphi)$, si en ψ aparece algún $i \in \mathsf{NOM}_c$ (i.e. $level(i) > 0$), entonces ψ es de una de estas tres formas: $@_i \psi'$, $@_n \langle r\rangle i$ ó $@_n i$ ($i \notin S_f(\psi')$ y $i \neq n$).*

Lo que queremos ver ahora es que el cálculo de la Figura 4.1 garantiza que $level(n)$ está acotada cuando n aparece en $ClSet^*(\varphi)$. Para ver esto, utilizaremos una noción de *profundidad modal* para fórmulas-@ que combine la profundidad meramente sintáctica (i.e., el concepto habitual de "profundidad modal") con la *profundidad* (según la función *level*) del prefijo de la fórmula-@.

Definición 4.8. Para toda fórmula $@_i \varphi$ definimos $d'(@_i \varphi) = level(i) + d(\varphi)$, donde $d(\varphi)$ es la profundidad modal de φ.

Lema 41. *Dada una fórmula $\varphi \in \mathcal{H}_0^{NNF}(@)$, si ψ aparece en alguna cláusula de $\mathrm{ClSet}^*(\varphi)$, entonces $d'(\psi) \leq d(\varphi)$ y, además, si $\psi = @_i j$, entonces $level(j) \leq d(\varphi)$.*

Demostración. Operemos por inducción en la longitud de los prefijos de alguna secuencia de derivación de $ClSet^*(\varphi)$. Para el caso base debemos considerar $ClSet(\varphi)$; como en φ sólo aparecen nominales de NOM_i, es claro que $d'(\varphi) \leq d(\varphi)$ y si $\varphi = @_i j$, $level(j) = 0 = d(\varphi)$.

Para el paso inductivo, supongamos que $L.C$ es un prefijo de longitud $n+1$, donde L es una secuencia y C una cláusula. C tiene que ser generable a partir de cláusulas presentes en L usando alguna de las reglas del cálculo. Analicemos cada uno de los casos.

Caso ($\wedge$) Supongamos que C es generada a partir de la aplicación de la regla ($\wedge$) sobre una cláusula D que aparece en L. C debe ser de la forma $\{@_i \psi\} \cup D'$, con $D = \{@_i (\psi \wedge \psi')\} \cup D'$. Por hipótesis inductiva, todas las fórmulas de D' cumplen con la condición pedida. Además, $d'(@_i \psi) \leq d'(@_i (\psi \wedge \psi')) \leq d(\varphi)$. Por otro lado, si $\psi \in \mathsf{NOM}$, de acuerdo a la Proposición 40, $\psi \in \mathsf{NOM}_i$, ya que no es admisible una fórmula de la forma $@_i (\psi \wedge \psi')$ cuando $\psi \in \mathsf{NOM}_c$.

Caso ($\vee$) Análogo al caso ($\wedge$).

Caso (RES) Si C es generada a partir de la aplicación de la regla de resolución sobre dos cláusulas D_1 y D_2 que aparecen en L, entonces $C \subset D_1 \cup D_2$, y por hipótesis inductiva, todas las fórmulas de C cumplen con la condición pedida.

Caso (REF) Análogo al caso (RES).

Caso ($[R]$) Supongamos que a partir de dos cláusulas de L, $D_1 = \{@_i [r]\psi\} \cup D_1'$ y $D_2 = \{@_i \langle r\rangle j\} \cup D_2'$, se genera $C = \{@_j \psi\} \cup D_1' \cup D_2'$. Por hipótesis inductiva sabemos que todas las fórmulas de $D_1' \cup D_2'$ cumplen con la condición pedida, con lo cual sólo debemos ver qué sucede con $@_j \psi$. Comencemos por observar que si $\psi \in \mathsf{NOM}$, entonces, por la Proposición 40, $\psi \in \mathsf{NOM}_i$, con lo cual $level(\psi) \leq d(\varphi)$. Sólo nos resta garantizar que $d'(@_j \psi) \leq d(\varphi)$. Ahora bien, por definición y aplicando la hipótesis inductiva vemos que $d'(@_i [r]\psi) = level(i) + 1 + d(\psi) \leq d(\varphi)$. Por la Proposición 39, tenemos dos posibilidades:

1. $level(j) = 0$, con lo cual $d'(@_j \psi) = 0 + d(\psi) < d'(@_i [r]\psi) \leq d(\varphi)$, ó
2. $level(j) = level(i) + 1$, con lo cual $d'(@_j \psi) = level(j) + d(\psi) = level(i) + 1 + d(\psi) = d'(@_i [r]\psi) \leq d(\varphi)$

Caso ($\langle R\rangle'$) Supongamos que C se genera a partir de la regla ($\langle R\rangle'$) sobre $D = \{@_i \langle r\rangle \psi\} \cup D'$, con D una cláusula que aparece en L. En este caso, se pueden dar dos posibilidades: o bien $C = \{@_i \langle r\rangle j\} \cup D'$ o bien $C = \{@_j \psi\} \cup D'$, para algún nominal j. En cualquier caso, es claro que todas las fórmulas de D' cumplen con la condición pedida. Veamos, entonces, el resto de los casos

por separado; para ello, partimos de $d'(@_i \langle r \rangle \psi) = level(i) + 1 + d(\psi) \leq d(\varphi)$. Ahora bien, por un lado tenemos $d'(@_i \langle r \rangle j) = level(i) + 1 + d(j) = level(i) + 1 < d'(@_i \langle r \rangle \psi) \leq d(\varphi)$. Por el otro, en cambio, encontramos $d'(@_j \psi) = level(j) + d(\psi) = level(i) + 1 + d(\psi) = d'(@_i \langle r \rangle \psi) \leq d(\varphi)$. Para finalizar, es claro que no es posible que suceda que ψ sea un nominal, ya que las restricciones sobre la regla $(\langle R \rangle')$ no permitirían que D fuera premisa de ella.

Caso (SYM) Supongamos que $C = \{@_i j\} \cup D'$ se crea aplicando (SYM) sobre $D = \{@_j i\} \cup D'$ con $i \succ j$, donde D es una cláusula que aparece en L. Claramente, por hipótesis inductiva, las fórmulas de D' cumplen con lo pedido. Ahora bien, por hipótesis inductiva, $level(i) \leq d(\varphi)$ y $d'(@_j i) = level(j) + d(i) = level(j) \leq d(\varphi)$. Luego, $d'(@_i j) \leq d(\varphi)$ y $level(j) \leq d(\varphi)$.

Caso (PARAM') En este caso, supongamos que $C = \{\psi(s/t)\} \cup D'_1 \cup D'_2$ se genera por paramodulación a partir de dos cláusulas de L, $D_1 = \{\psi(s)\} \cup D_1$ y $D_2 = \{@_s t\} \cup D'_2$. Una vez más, por hipótesis inductiva, las fórmulas de $D'_1 \cup D'_2$ cumplen con lo pedido y debemos preocuparnos sólo por ver qué sucede con $\psi(s/t)$. Comencemos por comprobar que $d'(\psi(s/t)) \leq d(\varphi)$. Si sucede que $\psi(s) = @_i \psi'(s)$, con $i \neq s$, entonces $d'(\psi(s/t)) = d'(\psi(s)) \leq d(\varphi)$. Supongamos, en cambio, que $i = s$. En ese caso, $d'(\psi(s/t)) = level(t) + d(\psi'(s/t))$ y, para que suceda que $d'(\psi(s/t)) > d(\varphi)$ se debería dar $level(t) > level(s)$. Sin embargo, sabemos que debe suceder que $s \succ t$ con lo cual, de acuerdo a la Definición 4.7 se debe cumplir $level(s) \geq level(t)$. Supongamos, para terminar, que $\psi(s)$ es de la forma $@_i j$; por hipótesis inductiva, $level(j) \leq d(\varphi)$. Si $j \neq s$, entonces $\psi(s/t) = @_t j$ y $level(j) \leq d(\varphi)$. Si, por el contrario, $j = s$, $\psi(s/t) = @_k t$ y, $level(t) \leq level(s) \leq d(\varphi)$.

Caso (PARAM-REL) Supongamos que C se crea a partir de la aplicación de (PARAM-REL) sobre dos cláusulas de L, $D_1 = \{@_i \langle r \rangle j\} \cup D'_1$ y $D_2 = \{@_i k_1\} \cup D'_2$. Claramente, C debe ser de una de dos formas: $C = \{@_{k_1} \langle r \rangle k_2\} \cup D'_1 \cup D'_2$ ó $C = \{@_j k_2\} \cup D'_1 \cup D'_2$. Como en los casos anteriores, sabemos que las fórmulas de $D'_1 \cup D'_2$ cumplen con lo pedido. Analicemos, entonces, los dos casos de interés. Sabemos que $i \succ k_1$, con lo cual $level(i) \geq level(k_1)$ y, por lo tanto, $d'(@_{k_1} \langle r \rangle k_2) = level(k_1) + 1 \leq level(i) + 1 = d'(@_i \langle r \rangle j) \leq d(\varphi)$. Por otro lado, de acuerdo a la Proposición 39, pueden darse dos posibilidades:

1. $level(j) = 0$, con lo cual $d'(@_j k_2) = 0 \leq d(\varphi)$, o bien
2. $level(j) = level(i) + 1$, con lo cual $d'(@_j k_2) = level(j) + d(k_2) = (level(i) + 1) + 0 = d'(@_i \langle r \rangle j) \leq d(\varphi)$.

Nos queda ver que $level(k_2) \leq d(\varphi)$, para ello, observemos que $level(k_2) = level(k_1) + 1 \leq level(i) + 1 = d'(@_i \langle r \rangle j) \leq d(\varphi)$. □

Si tenemos en cuenta la Proposición 40, este lema nos conduce directamente al siguiente corolario:

Corolario 42. *Para toda fórmula φ, si i es un nominal que aparece en alguna fórmula de* $\mathrm{ClSet}^*(\varphi)$, *entonces* $level(i) \leq d(\varphi)$.

Lo que esto nos dice es que el cálculo de la Figura 4.1 garantiza que no se generan nominales infinitamente profundos. Sólo nos resta ver, entonces, que tampoco es posible que exista un nivel de profundidad con infinitos nominales distintos. La regla $(\langle R \rangle')$, como veremos a continuación, lidia adecuadamente con este hecho. La base de esta demostración es la siguiente observación, que se deduce directamente de las Proposiciones 38 y 40:

Proposición 43. *Para toda fórmula φ y todo nominal i, el número de fórmulas distintas de la forma $@_i \langle r \rangle \psi$ (con $\psi \notin \mathsf{NOM}$) que aparecen en* $\mathrm{ClSet}^*(\varphi)$ *es finito.*

Ahora es fácil ver que no es posible generar infinitos nominales en un mismo nivel. Para formalizar esto, operemos por inducción en la *profundidad* de los nominales que aparecen en $ClSet^*(\varphi)$. Para ello, valgámonos de una función que nos devuelve todos los nominales de un conjunto de cláusulas que pertenezcan a determinado nivel:

Definición 4.9. Sea N cualquier conjunto de cláusulas, definimos $level_x(N) = \{i \mid i \in \mathsf{NOM} \wedge level(i) = x \wedge i \in S_f(N)\}$, donde $S_f(N)$ es el conjunto de todas las subfórmulas que aparecen en N.

Lema 44. *Para toda fórmula φ y todo $x \in \mathbb{N}$, $level_x(\mathrm{ClSet}^*(\varphi))$ es un conjunto finito.*

Demostración. Operemos por inducción en x. Para $x = 0$, $level_0(ClSet^*(\varphi))$ representa el conjunto de nominales que aparecen en φ. Para el paso inductivo, supongamos que $level_x(ClSet^*(\varphi))$ es finito. Por la Proposición 43, el número de fórmulas de la forma $@_i \langle r \rangle \psi$ para $i \in level_x(ClSet^*(\varphi))$ (con $\psi \notin \mathsf{NOM}$) será finito y según la regla $(\langle R \rangle')$, se generará a lo sumo un nuevo nominal por cada una, con lo cual $level_{x+1}(ClSet^*(\varphi))$ será también finito. □

Finalmente, de los Lemas 41 y 44 se deduce directamente el siguiente Teorema:

Teorema 45. *Para toda fórmula φ, el conjunto de nominales distintos que aparecen en* $\mathrm{ClSet}^*(\varphi)$ *es finito.*

Corolario 46. *El cálculo de resolución de la Figura 4.1 constituye un método de decisión para la lógica $\mathcal{H}(@)$.*

Capítulo 5

HyLoRes 2.0: Poniendo a prueba las ideas

HyLoRes es un demostrador para la lógica $\mathcal{H}(@, \downarrow)$ escrito en Haskell basado en el cálculo de resolución presentado en [Areces *et al.*, 2001]. Es necesario aclarar que no se trata de una herramienta que pretenda competir seriamente con demostradores que representan el estado del arte en demostración automática para lógica de primer orden o description logics (que pueden ser usados para tratar fórmulas modales, ver Secciones 1.3 y 2.3). Demostradores como RACER [Haarslev and Möller, 2001; RACER, 2004] ó *SAT [Giunchiglia *et al.*, 2002; *SAT, 2004] están especialmente afinados y cuentan con una batería de heurísticas y optimizaciones con las que consiguen resultados sobresalientes. En contraste, HyLoRes implementa un conjunto relativamente simple de optimizaciones y es todavía más una prueba de concepto que una aplicación que pueda ser usada en un ambiente real. Pero es justamente por ello que se trata de una entorno ideal para poner a prueba nuevas ideas y realizar una valoración empírica de las mismas.

Como parte de este trabajo se adaptó HyLoRes para que utilice las reglas presentadas en la Figura 4.1 y se realizaron algunas pruebas para comparar la performance de ambas versiones. En este capítulo repasaremos brevemente tanto las características del demostrador original como las de la nueva versión y presentaremos los resultados de las pruebas que realizamos.

5.1 HyLoRes 1.0

En esta sección haremos un breve repaso de las que consideramos las características más importantes de HyLoRes 1.0. Se puede encontrar una descripción más detallada en [Areces and Heguiabehere, 2002; HyLoRes, 2004].

5.1.1 Algoritmo de la cláusula dada

HyLoRes implementa una versión del algoritmo de la "cláusula dada" [Voronkov, 2001] ("given clause" algorithm), presentado más adelante en el Algoritmo 1. Este algoritmo construye de una manera ordenada el conjunto saturado de cláusulas, garantizando que ninguna cláusula se demorará infinitamente en ser procesada. Además, permite incorporar distintas heurísticas tendientes a acelerar la derivación de la cláusula vacía.

En este algoritmo, las cláusulas se distribuyen en tres conjuntos de cláusulas distintos: new, clauses e in-use. En new se almacenan todas las cláusulas que se generan como resultado de la aplicación de una regla; también es el conjunto donde se pone la fórmula original al comenzar el cálculo. Las cláusulas en este conjunto están a la espera de ser simplificadas (e.g. eliminación de tautologías y contradicciones simples); se lo puede considerar así como un conjunto *de paso*, ya que una cláusula no permanece allí mucho tiempo. Las cláusulas ya simplificadas pero que todavía están esperando la aplicación de alguna regla se almacenan en clauses. En in-use, en

Algoritmo 1 Algoritmo de la cláusula dada implementado en HyLoRes

```
input: init: conjunto de cláusulas
var: new, clauses, inuse: conjunto de cláusulas
var: given: cláusula
clauses := {}; inuse = {}; new:=normalize(init)
if {} ∈ new then
  return "insatisfacible"
end if
clauses := computeComplexity(new)
while clauses ≠ {} do
  // Selección de la cláusula dada
  given := select(clauses)
  clauses := clauses \{given}
  // Inferencia
  new := infer(given,inuse)
  new := normalize(new)
  if {} ∈ new then
    return "insatisfacible"
  end if
  // Eliminación de cláusulas redundantes
  new := simplify(new, inuse ∪ clauses)
  inuse := simplify(inuse, new)
  clauses := simplify(clauses, new)
  // Inicialización para el próximo ciclo
  if notRedundant(given) then
    inuse := inuse ∪{given}
  end if
  clauses := clauses ∪ computeComplexity(new)
end while
return "satisfacible"
```

- *normalize(A)* normaliza las fórmulas y maneja tautologías/contradicciones triviales.
- *computeComplexity(A)* determina el tamaño, profundidad modal, número de literales, etc. de cada fórmula en A; estos valores son usados por *select* para elegir la cláusula dada.
- *infer(Cl,A)* aplica las reglas de resolución a la cláusula Cl usando las cláusulas de A. Si las reglas unarias son aplicables, ninguna otra regla se aplica.
- *simplify(A,B)* elimina cláusulas subsumidas, devolviendo el subconjunto de A que no es subsumido por ningún elemento de B.
- *notRedundant(given)* es verdadero si ninguna de las reglas unarias fue aplicada.

cambio, se guardan las cláusulas a las que ya se les ha intentado aplicar alguna regla al menos una vez. Conceptualmente, en cada iteración del ciclo, se preprocesan y se pasan a clauses todas las cláusulas que estén en new (salvo aquellas que se pueda determinar en forma temprana, como veremos más adelante, que no es necesario procesar); se elige una única *cláusula dada* (given clause) de entre todas las cláusulas en clauses, se intentan aplicar todas las reglas del cálculo entre la cláusula dada y todas las cláusulas que estén en in-use y se guardan en new todas las cláusulas nuevas que se generen como resultado de la aplicación. La cláusula dada se agrega luego a in-use, si no es redundante.

Las cláusulas en clauses se ordenan utilizando una noción de complejidad de cláusula que involucra parámetros como tamaño de la cláusula, máxima profundidad modal, etc. Elegir la cláusula dada consiste en tomar la cláusula con menor complejidad; salvo una vez cada n iteraciones, en que se elige la cláusula más vieja del conjunto. De esta forma se garantiza que todas las cláusulas sean eventualmente procesadas.

En la versión 1.0 todas las fórmulas de una cláusula deben revisarse para determinar si pueden ser usadas como premisa de alguna regla. Se utiliza, de todas formas, una noción de función de selección (análoga aunque diferente de la que hemos usado en los capítulos anteriores) para limitar la aplicación de las reglas (RES) y ($[R]$). Esto se basa en el hecho de que estas dos reglas constituyen el motor de resolución propiamente dicho del cálculo y, por lo tanto, es posible aplicar directamente sobre ellas algunos resultados de resolución para lógica de primer orden. Podemos mencionar dos grandes diferencias entre este mecanismo de funciones de selección y el que hemos visto en los capítulos anteriores:

1. En HyLoRes 1.0 se *seleccionan* todas las fórmulas de una cláusula que sean de la forma $@_i \neg\varphi$ (se debe recordar que no se trabajaba con fórmulas de $\mathcal{H}^{NNF}$) y las reglas (RES) y ($[R]$) se aplican sobre *todas las posibles* fórmulas seleccionadas. En cambio, en el cálculo que hemos visto, se elige una única fórmula (ya sea por orden o por selección) y se intenta aplicar una regla sobre ella.

2. En la versión 1.0 no hay mecanismo de selección para el resto de las reglas del cálculo. En particular, no hay ningún mecanismo que regule la aplicación de la regla (PARAM). El cálculo que proponemos, en cambio, maneja de forma homogénea todas las reglas.

5.1.2 Eliminación de cláusulas redundantes

Las optimizaciones más interesantes que presenta HyLoRes 1.0 tienen que ver con la eliminación de cláusulas *subsumidas* (subsumed clauses). Se dice que una cláusula está *subsumida* cuando es redundante; es decir, cuando no puede ser que la cláusula sea falsa si todas las demás cláusulas (aquellas que la subsumen) son verdaderas. Un ejemplo simple de subsunción es la inclusión estricta, por cuanto es claro que C subsume a D toda vez que $C \subset D$.

La eliminación de cláusulas redundantes es a la vez una de las optimizaciones más importantes pero también de las más costosas en un demostrador basado en resolución. Con esto queremos decir que si bien la eliminación temprana de cláusulas subsumidas permite disminuir notoriamente el número de cláusulas generadas por un demostrador, muchas veces el costo de detectar dichas cláusulas es tal que el tiempo total de ejecución termina siendo mayor. La valoración final de una optimización de este tipo suele ser empírica.

En HyLoRes 1.0 se implementan dos mecanismos de eliminación de cláusulas redundantes. El más sencillo consiste en no agregar a in-use aquellas cláusulas sobre las cuales se aplicó con éxito alguna de las reglas ($\wedge$), ($\vee$) o ($\langle R\rangle$)[1]. La razón de esto es evidente: las premisas de la regla son claramente subsumidas por sus consecuentes y, por lo tanto, pueden eliminarse. Este es un ejemplo de lo que se llama *eliminación hacia atrás de cláusulas subsumidas* (backward subsumption deletion).

[1] Lo mismo podría decirse de la regla (@), sin embargo ésta no está manejada explícitamente sino que los @ redundantes se eliminan durante un proceso de normalización de cláusulas

Más sofisticada es la manera en que se maneja la *eliminación hacia adelante de cláusulas subsumidas* (forward subsumption deletion); esto es, no agregar a clauses aquellas cláusulas nuevas que sean subsumidas por las cláusulas en clauses e in-use. En este caso, el criterio que se toma es el de inclusión de cláusulas mencionado más arriba. Es decir, se descarta una cláusula nueva si incluye a otra que esté en clauses o en in-use. Para realizar esta operación en forma eficiente se representa el conjunto de cláusulas usando una lista ordenada de *tries* donde cada nodo es un entero asociado a una fórmula, y cada rama está también ordenada y representa una cláusula. Esta estructura permite determinar si una cláusula es subsumida sin revisar cláusula por cláusula (clause-at-a-time), sino mirando el conjunto de cláusulas en su totalidad (set-at-a-time) [Voronkov, 2001].

En HyLoRes 1.0 también se implementó una forma de eliminación hacia atrás en la cual se utiliza esta estructura para determinar si una cláusula nueva subsume a alguna cláusula en in-use; sin embargo, tests empíricos mostraron que el costo en tiempo de esta operación era mayor que el beneficio que producía y se la desactivó.

5.1.3 La regla ($\langle R \rangle$)

La regla ($\langle R \rangle$), por ser generadora de nuevos elementos sintácticos, es una de las más *problemáticas* del cálculo y mereció un tratamiento especial. En primer lugar, HyLoRes implementa un *criterio de bloqueo* para esta regla; es decir, una condición que de cumplirse determina que no sea necesario aplicar la regla ($\langle R \rangle$) sobre una cláusula dada. Dada una cláusula $\{@_i \langle r \rangle \varphi\} \cup C$, el criterio consiste en determinar si existe algún nominal j tal que $\{@_i \langle r \rangle j\} \cup C$ y $\{@_j \varphi\} \cup C$ sean subsumidas (por inclusión de conjuntos) por el conjunto de cláusulas generadas hasta el momento; de ser así, la regla no se aplica. Se puede interpretar este criterio como una forma un poco más elaborada (y también más costosa) de eliminación hacia adelante de cláusulas redundantes. Para maximizar las probabilidades de que el criterio efectivamente bloquee la aplicación de esta regla para una cláusula dada, se decidió *demorar* la aplicación de esta regla todo lo posible. Para ello se aplican todas las reglas excepto ($\langle R \rangle$) sobre una cláusula dada y, de contener fórmulas de la forma $@_i \langle r \rangle \varphi$ (donde φ no es un nominal), la cláusula se pasa a una cola especial de "cláusulas con diamantes demorados". Una vez que clauses se vacía, se revisa esta cola y se aplica la regla ($\langle R \rangle$) sobre la primer cláusula ingresada.

Tal vez el mayor problema con este procedimiento es que, potencialmente, una cláusula necesaria para derivar la cláusula vacía puede demorarse demasiado tiempo en ser generada. De hecho, una heurística habitual en resolución consiste en tratar en forma equitativa a todas las cláusulas; es decir, permitir que todas las cláusulas tengan las mismas chances de participar en las reglas.

5.1.4 Paramodulación

Al igual que en resolución para lógica primer orden con igualdad, la paramodulación es una operación cuya aplicación se debe tratar de minimizar todo lo posible. Muchas ideas y optimizaciones para lógica de primer orden se discuten en [Bachmair and Ganzinger, 1998]. Sin embargo, en HyLoRes, el manejo de la paramodulación es, tal vez, uno de los puntos más débiles de este demostrador. La regla (PARAM) se aplica tal como está definida en el cálculo, con la única salvedad de que sólo se reemplaza un nominal por otro que sea menor, dado cierto orden (i.e., suponiendo que $i \succ j$, ambas igualdades $@_i j$ y $@_j i$ dan origen a una única regla de reescritura orientada $i \rightsquigarrow j$) [Bachmair and Ganzinger, 1998].

5.2 HyLoRes 2.0

Por diversos motivos, incorporar a HyLoRes los resultados presentados en los Capítulos 3 y 4 insumió más trabajo del inicialmente esperado. Para empezar, la versión 1.0 trabajaba internamente usando sólo los operadores $\langle \neg, \wedge, [], @ \rangle$, mientras que el cálculo con orden y selección estaba pensado para trabajar con fórmulas en NNF. Este cambio de por sí implicó reescribir una cantidad

importante del código. Por otro lado, el nuevo cálculo nos permitió realizar nuevas optimizaciones que, sin embargo, requirieron modificar muchas de las estructuras de representación interna, realizar cambios en las interfaces de las mismas, etc. De la versión 1.0 a la 2.0 se conservó casi intacto el motor monádico, la implementación general del algoritmo de la cláusula dada, los tests eficientes de cláusulas redundantes y el front-end (ver Apéndice A). El resto, en mayor o menor medida debió ser adaptado o reescrito y se incorporaron nuevas estructuras de datos. A continuación enumeramos las principales características de la nueva versión.

5.2.1 Procesamiento de la cláusula dada

El algoritmo de la cláusula dada de HyLoRes 2.0 es casi equivalente al de la versión 1.0. El único cambio importante es que, para incorporar un mecanismo adicional de eliminación de cláusulas subsumidas que explicamos en la Sección 5.2.4, el ingreso de la cláusula dada a in-use se realiza dentro de *infer(Cl,A)*. Esta diferencia se puede ver comparando los Algoritmos 1 y 2.

El cálculo con orden y selección tiene la ventaja de que cada cláusula puede ser premisa de *una única* regla (más allá de las reglas de paramodulación que son un caso especial y, por lo tanto, se consideran por separado). El Algoritmo 3 muestra esquemáticamente cómo funciona internamente en esta nueva versión la función *infer(given,A)* utilizada en el Algoritmo 2. Es importante observar que dado que en la versión 1.0 todas las fórmulas de una cláusula podían participar en una inferencia, no era posible que el llamado a *findRule(given)* devolviera una única regla como se ve en el Algoritmo 3.

Algoritmo 2 Algoritmo de la cláusula dada implementado en HyLoRes 2.0

```
input: init: conjunto de cláusulas
var: new,clauses, inuse: conjunto de cláusulas
var: given: cláusula
clauses := {}; inuse = {}; new:=normalize(init)
if {} ∈ new then
  return "insatisfacible"
end if
clauses := computeComplexity(new)
while clauses ≠ {} do
  // Selección de la cláusula dada
  given := select(clauses)
  clauses := clauses \{given}
  // Inferencia
  (new,inuse) := infer(given,inuse)
  new := normalize(new)
  if {} ∈ new then
    return "insatisfacible"
  end if
  // Eliminación de cláusulas redundantes
  new := simplify(new, inuse ∪ clauses)
  inuse := simplify(inuse, new)
  clauses := simplify(clauses, new)
  // Inicialización para el próximo ciclo
  clauses := clauses ∪ computeComplexity(new)
end while
return "satisfacible"
```

Algoritmo 3 *infer(given,inuse)* en HyLoRes 2.0

```
input: given: cláusula
input: inuse: conjunto de cláusulas
var: rule: regla de inferencia
var: consequents: conjunto de cláusulas
rule := findRule(given)
consequents := applyRule(rule, given, inuse)
if ¬subsumes(consequents, given) then
  consequents := consequents ∪ applyRule(PARAM-MAIN, given, inuse)
  if ¬subsumes(consequents, given) then
    consequents := consequents ∪ applyRule(PARAM-AUX, given, inuse)
    inuse := inuse ∪{given}
  end if
end if
return (consequents, inuse)
```

- *findRule(given)* devuelve la única regla de inferencia (exceptuando (PARAM)) de la cual *given* puede ser premisa.
- *applyRule(rule, given, A)* aplica la regla *rule* usando como premisa a *given* y cualquier cláusula apropiada de *inuse*.
- *PARAM-MAIN* es la regla (PARAM') o (PARAM-REL) (según corresponda) usando a given como premisa principal.
- *PARAM-AUX* es la regla (PARAM') o (PARAM-REL) (según corresponda) usando a given como premisa auxiliar.
- *subsumes(C,A)* indica si alguna cláusula en C subsume a A.

5.2.2 Orden entre fórmulas

Recordemos que las Definiciones 3.8 y 4.7 nos dan las condiciones que debe cumplir un orden entre fórmulas para garantizar que el cálculo es completo y termina siempre. El orden $\succ^{\omega}$ de la Definición 4.5, basado en el orden $\succ_h$ de la Definición 3.9, es sólo un ejemplo de un orden que cumple con estas condiciones. Si bien $\succ^{\omega}$ fue el orden que se utilizó en un primer momento, como veremos a continuación, se prefirió finalmente un orden adecuado para terminación alternativo.

Se puede ver fácilmente que para cualesquiera $i, j, k \in \mathsf{NOM}$, $p \in \mathsf{PROP}$ y $r \in \mathsf{REL}$ se cumple $@_i \langle r \rangle j \succ^{\omega} @_k p$. Esto significa que dado una cláusula $C = \{@_i \langle r \rangle j, @_k p\} \cup C'$ donde no hay nada seleccionado en C' y $@_i \langle r \rangle j$ y $@_k p$ son las mayores fórmulas de C, C sólo podrá ser premisa auxiliar de la regla ($[R]$). Ahora bien, una heurística estándar en demostración automática para lógicas modales consiste en favorecer el razonamiento proposicional por sobre el razonamiento modal. Esto significa que, en general, deberíamos favorecer la aplicación de la regla (RES) por sobre la regla ($[R]$). Esto puede lograrse usando un orden adecuado para terminación $\succ$ que verifique $@_k p \succ @_i \langle r \rangle j$, y esta es la propiedad que cumple el orden finalmente implementado.

La definición del orden utilizado es análoga a la de la Definición 4.5, pero utilizando una leve variación del orden de la Definición 3.9: en lugar de $size(x)$, se usa la función $weight(x)$ definida como:

$$
\begin{aligned}
weight(i) &= 1 \\
weight(p) &= 3 \\
weight(r) &= 1 \\
weight(\neg\varphi) &= 1 + weight(\varphi) \\
weight(\varphi \wedge \psi) &= weight(\varphi) + 1 + weight(\psi) \\
weight([r]\varphi) &= 1 + weight(r) + weight(\varphi) \\
weight(\langle r\rangle\varphi) &= 1 + weight(r) + weight(\varphi) \\
weight(@_i\,\varphi) &= 1 + weight(i) + weight(\varphi)
\end{aligned}
$$

y la precedencia verifica $@ > \neg > \wedge > \vee > [] > p > \langle\rangle > r > i > \top$, para todo $i \in \mathsf{NOM}$, $p \in \mathsf{PROP}$, $r \in \mathsf{REL}$.

5.2.3 Función de selección

La forma en que HyLoRes maneja las funciones de selección cambió completamente para la versión 2.0. Una función de selección es ahora una función escrita en Haskell con una signatura especial, y la función de selección a ser usada se convirtió en un parámetro más del cálculo. Se implementaron además varios generadores de funciones de selección que permiten ensayar con distintas configuraciones sin necesidad de recompilar la aplicación (más sobre esto en la Sección A.2 y A.5).

5.2.4 Criterios adicionales de subsunción

Se conservaron los criterios de eliminación de cláusulas redundantes con que contaba HyLoRes 1.0, y se incorporaron dos mecanismos adicionales que conciernen sólo las reglas (RES), (PARAM') y (PARAM-REL).

El caso de la regla (RES) es el más sencillo. Supongamos que $C = \{@_i\,p\} \cup C'$ y $D = \{@_i\,\neg p\} \cup D'$ resuelven para generar $C' \cup D'$. Si $C' \subseteq D'$, entonces $C' \cup D' = D'$ y, por lo tanto, D' subsume a D. De la misma forma, si $D' \subseteq C'$, entonces C' subsume a C. Esto significa que podemos hacer un chequeo local (y por lo tanto, económico) para determinar si el consecuente subsume a una de sus premisas. Cuando la premisa subsumida es la cláusula dada, simplemente no la agregamos al conjunto de cláusulas in-use; cuando la subsumida era la premisa en in-use, podemos sacarla de dicho conjunto.

El caso de las reglas de paramodulación es similar, sólo que aquí la premisa principal es la única que puede ser subsumida por los consecuentes. Veremos sólo que sucede con la regla (PARAM'); el caso de (PARAM-REL) es análogo. Supongamos, entonces, que podemos aplicar dicha regla sobre $C = \{@_i\,j\} \cup C'$ y $D = \{\varphi(i)\} \cup D'$ para obtener $E = \{\varphi(i/j)\} \cup C' \cup D'$. Si sucede que $C' \subseteq D'$, entonces tenemos $E = \{\varphi(i/j)\} \cup D'$. Ahora bien, E de por sí no subsume a D, pero sí lo hace si la tomamos junto con C. Es decir, si suponemos C verdadera, D y E se convierten en cláusulas lógicamente equivalentes, con lo cual una de las dos se vuelve redundante. La decisión de eliminar D radica en que E es de alguna forma más *simple* por usar un nominal menor en el orden. Se puede entender mejor esta optimización considerando el caso en que C' es vacío. Cuando eso sucede, C puede leerse como "i y j representan el mismo mundo"; luego, podemos reemplazar en todas las cláusulas las apariciones de i por j. Esta forma de eliminación de redundancia es una forma *segura* y más general de llevar esto a cabo.

5.2.5 Representación interna del conjunto de cláusulas

Si se analizan las reglas y se tiene en cuenta que las cláusulas que son premisas de una regla unaria siempre son subsumidas, se llega fácilmente a la conclusión de que si una cláusula está en in-use, entonces su fórmula distinguida tiene que ser de la forma $@_i\,j$, $@_i\,\neg j$, $@_i\,p$, $@_i\,\neg p$, $@_i\,\langle r\rangle j$ ó $@_i\,[r]\varphi$. Además, si bien las cláusulas del primer tipo pueden ser premisa auxiliar tanto de la regla (RES) como de las de paramodulación, el resto sólo puede ser premisa (y de sólo un tipo) de sólo una de las reglas (RES) o ($[R]$). Teniendo esto en cuenta, se utilizó un tipo de datos especial

para representar el conjunto in-use, que clasifica las cláusulas de acuerdo al rol que cada una puede cumplir. De esta forma, podemos encontrar todas las cláusulas contra las cuáles es posible aplicar cierta regla binaria, sin tener que explorar todos los elementos de in-use.

5.2.6 La regla ($\langle R \rangle'$)

Dado que una cláusula que sea premisa de la regla ($\langle R \rangle'$) no puede ser premisa de otra regla, en HyLoRes 2.0 no es necesario demorar programáticamente la aplicación de dicha regla como se hacía anteriormente. Alcanzaría con utilizar una noción de complejidad que baje la prioridad de las cláusulas que puedan ser premisa de ($\langle R \rangle$). Esto no sólo simplifica los algoritmos y las estructuras utilizadas, sino que, como ventaja adicional, ya no se demora más de lo recomendable ninguna cláusula.

5.2.7 La regla (RES-BOX)

En HyLoRes 1.0 se podía aplicar la regla (RES) entre cualquier par de fórmulas con la condición de que una fuera la negación (sintáctica) de la otra. En particular, permitía resolver fórmulas de la forma $@_i\,[r]\psi$ con $@_i\,\neg[r]\psi$. Una implementación directa de la regla (RES) basada en fórmulas de $\mathcal{H}^{NNF}(@)$ no admite aplicar resolución entre fórmulas de esta forma.

Si bien el cálculo es completo resolviendo sólo sobre fórmulas de LIT-P, en un gran número de casos se puede acelerar mucho una refutación si se resuelve entre dos fórmulas como estas. Es por ello que en HyLoRes 2.0 se implementó parcialmente esta regla: cada vez que se procesa una cláusula cuya fórmula distinguida es de la forma $@_i\,\langle r\rangle\psi$ con $\psi \notin \mathsf{NOM}$, se buscan en in-use todas las cláusula cuya fórmula distinguida sea $@_i\,[r]\neg\psi$ y se aplica la regla (RES) entre todas ellas. Esta regla está implementada sólo en forma parcial ya que, de otra forma, sería necesario almacenar también las cláusulas cuya fórmula distinguida sea de la forma $@_i\,\langle r\rangle\psi$, lo cuál parece ser un overhead de espacio innecesario.

5.3 Evaluación empírica

Una tarea importante en el desarrollo de un demostrador como HyLoRes consiste en *ajustarlo* hasta lograr un funcionamiento óptimo. Por "ajustar" entendemos, en este contexto, buscar la mejor parametrización, descartar heurísticas u optimizaciones que representan un costo mayor que su beneficio, etc. Finalizada esta etapa de ajuste fino, es interesante comparar empíricamente el demostrador con otros que presenten un nivel de desarrollo similar. En esta sección describiremos brevemente cómo se puso a punto a HyLoRes 2.0 y analizaremos los resultados de su comparación empírica con HyLoRes 1.0.

Tanto al afinar HyLoRes 2.0 como al compararlo con HyLoRes 1.0, lo que estaremos haciendo es *comparar* el rendimiento de dos (o más) demostradores. ¿Cómo se realiza esta comparación? Conceptualmente es un procedimiento simple: se toma un conjunto de fórmulas "representativo" del universo de problemas, se procesan estas fórmulas con todos los demostradores a evaluar, se eligen algunas métricas de comparación (e.g. "tiempo necesario para resolver cada fórmula") y se compara en base a estas métricas. El problema clave en este procedimiento es cómo obtener una muestra "representativa" de fórmulas.

Es imposible definir qué significa *muestra representativa* sin determinar de antemano una aplicación dada. Si un demostrador ha sido desarrollado para una aplicación particular, entonces nos interesa testearlo en instancias del problema de aplicación que representen casos usuales. Para demostradores genéricos (como HyLoRes) lo que nos interesa es testear comportamiento promedio. Usualmente, esto se traduce en testeo sobre una variedad de casos particulares diferentes, o en testeo intensivo sobre casos generados al azar, buscando cierta significatividad estadística.

A continuación, repasamos brevemente los distintos conjuntos de tests disponibles, explicamos cuáles elegimos y finalmente analizamos los resultados obtenidos.

5.3.1 Conjuntos de problemas modales e híbridos

HyLoRes puede aceptar como entrada cualquier fórmulas de $\mathcal{H}(@)$, en particular, puede trabajar con fórmulas puramente proposicionales, y de la lógica modal básica. Esto quiere decir que, para realizar pruebas podríamos elegir conjuntos de prueba proposicionales, modales o híbridos. Probar HyLoRes con un conjunto de tests proposicional no tiene demasiado sentido. Los conjuntos de tests puramente modales, por otro lado, si bien no permiten evaluar el funcionamiento de las reglas de paramodulación, han sido más investigados que los tests para $\mathcal{H}(@)$ y permiten comparar HyLoRes con demostradores modales de menor poder expresivo. Es por ello que, a continuación, repasaremos el estado del arte en conjuntos de prueba, tanto para lógicas modales tradicionales como híbridas.

Tests a medida vs. Tests aleatorios

Cuando se trata de evaluar demostradores automáticos para una lógica cualquiera (no necesariamente modal), existen en general dos enfoques distintos: usar conjuntos de tests a medida y usar conjuntos de tests aleatorios. Los conjuntos de test a medida están pensados especialmente para cubrir espacios de problemas específicos. Normalmente, son casos representativos de problemas para los cuales se podría usar un demostrador. Por el contrario, los tests aleatorios son generados con algún algoritmo en base a un juego de parámetros iniciales.

La *calidad* de los tests a medida es en general mejor que la de los tests aleatorios; estos últimos podrían, por ejemplo, generar conjuntos de prueba que incluyan fórmulas trivialmente satisfacibles. Además, si el espacio de problemas es demasiado grande, no es sencillo generar un conjunto de tests aleatorios que garantice representatividad sin perder el control sobre la dificultad de los problemas generados. Por otro lado, para que los tests a medida sean de utilidad es necesario contar con una comunidad grande de gente que genere nuevos tests, ya sea para cubrir nuevas zonas del espacio de problemas (i.e. aumentar la representatividad) o para reemplazar los problemas que se volvieron demasiado fáciles.

Es interesante observar qué sucede con dos de las lógicas más conocidas, como son la lógica proposicional y la de primer orden para después contrastar con las lógicas modales. El espacio de problemas de la lógica proposicional es comparativamente chico y además existe una forma normal simple, 3CNF, a la que toda fórmula proposicional puede ser reducida. Los tests aleatorios son, claramente, la mejor opción para evaluar demostradores proposicionales, y 3CNF-aleatorio (random 3CNF) es, de hecho, el conjunto de test estándar en este caso [Mitchell *et al.*, 1992].

En lógica de primer orden, la situación es completamente diferente. Para empezar, se trata de una lógica muy expresiva, con lo cual es difícil cubrir el espacio de problemas usando tests aleatorios. Por otro lado, su expresividad permite codificar en ella una gran cantidad de problemas de otras áreas. Además, por ser una de las lógicas más investigadas, es posible mantener al día un conjunto de tests a medida suficientemente representativo. Es el caso del TPTP (Thousands of problems for theorem proving), un repositorio estándar de tests a medida para lógica de primer orden [Sutcliffe and Suttner, 1998].

Con las lógicas modales la situación es un poco más complicada. Para empezar, como ya observamos en la Sección 1.3, con esta denominación nos referimos a toda una familia de lógicas; por lo tanto, no alcanza con un sólo conjunto de tests, sino que necesitaríamos un test para cada lógica. Este problema de "diversidad" de lógicas podría resolverse armando distintos tests a medida; sin embargo la comunidad de investigadores en esta área no es suficientemente grande como para mantenerlos al día. Un ejemplo claro de esta limitación lo encontramos en lo que sucedió con el conjunto de tests de Balsiger, Heuerding y Schwendimann (BHS) que comentamos más adelante.

La solución estándar en este momento consiste en utilizar conjuntos de tests aleatorios para lógicas modales específicas, (e.g. para la lógica modal básica). Si bien todavía es posible obtener buena representatividad utilizando este tipo de tests, el riesgo de introducir un algoritmo generador de tests inadecuados es considerablemente más grande que en el caso proposicional. Esto fue lo que sucedió con el test QBF-modal; que fue considerado el conjunto de test estándar una vez que

las limitaciones de BHS se hicieron evidentes. Este test generaba *fórmulas booleanas cuantificadas aleatorias* (random quantified boolean formulas) y las traducía (conservando satisfacibilidad) a la lógica modal básica. Aquí se aprovechaba el hecho de que se sabía cómo generar fórmulas QBF aleatorias que fueran representativas y que el problema de la validez de una fórmula QBF es PSPACE-complete [Papadimitriou, 1994], al igual que el problema de la validez de una fórmula de la lógica modal básica. Sin embargo, eventualmente se observó que las fórmulas modales satisfacibles que se obtienen como resultado de esta traducción tienen modelos con características tan específicas que no es posible afirmar realmente que se trate de un conjunto de test modal verdaderamente representativo [Heguiabehere, 2002; Heguiabehere and de Rijke, 2001].

Hoy en día, el conjunto de test estándar para la lógica modal básica es el denominado 3CNF$\Box_m$ aleatorio, que es una adaptación del test 3CNF aleatorio de la lógica proposicional.

Para evaluar HyLoRes 2.0 utilizamos los tests de BHS y una versión híbrida de 3CNF$\Box_m$ aleatorio. Como ya dijimos, este demostrador no representa el estado del arte en el área, y los tests de BHS todavía constituyen problemas interesantes. Usamos estos tests principalmente para realizar ajustes sobre HyLoRes. De todas formas, BHS es un test puramente modal y nos interesaba comparar ambas versiones de HyLoRes usando fórmulas híbridas. Para ello utilizamos configuraciones relativamente simples de 3CNF$\Box_m$ híbrido. A continuación presentamos ambos tests con un poco más de detalle.

El conjunto de tests de Balsiger, Heuerding y Schwendimann

Esta colección de problemas constituyó uno de los primeros intentos por obtener un conjunto de test riguroso para comparar demostradores modales. Consiste en nueve familias de problemas, cada familia, a su vez, se divide en dos clases: satisfacibles e insatisfacibles. Cada clase está formada por una sucesión de fórmulas de complejidad ascendente. Una evaluación con este conjunto de tests consiste en correr el demostrador con todos los problemas de cada clase; otorgándole 100 segundos para que resuelva cada problema. Lo que se toma en cuenta es el problema más difícil de cada clase que pudo resolver[2].

Inicialmente se esperaba que el salto de complejidad entre una fórmula y la siguiente en la misma clase fuera exponencial. A pesar de todo, los progresos en el área volvieron a este conjunto de test rápidamente obsoleto. Los demostradores más avanzados pueden resolver casi todas las fórmulas de la mayoría de las clases sin problemas, y ya no se considera que haya una diferencia de complejidad exponencial entre fórmula y fórmula. De todas maneras, para un prototipo como HyLoRes, el test de BHS sigue siendo un desafío y puede ser utilizado efectivamente para valorar posibles mejoras y optimizaciones. Hay que tener en cuenta que este test es puramente modal y no existen extensiones interesantes para lenguajes híbridos.

CNF$\Box_m$ aleatorio

Este conjunto de test, si bien es anterior en el tiempo a QBF modal, debió sufrir varias correcciones y modificaciones hasta llegar a ser considerado el conjunto de test estándar para evaluar demostradores modales [Patel-Schneider and Sebastiani, 2003]. Este test está basado en 3CNF aleatorio, que es el estándar de facto para demostradores proposicionales. En 3CNF aleatorio se generan fórmulas en forma clausal conjuntiva donde cada cláusula posee exactamente tres literales. La idea de CNF$\Box_m$ es análoga: aquí se generan conjunciones de cláusulas CNF$\Box_m$, donde cada cláusula CNF$\Box_m$ es un disyunción de literales que también pueden ser modales, entendiéndose en este contexto que un átomo modal es una fórmula de la forma $\Box_i C$, donde $\Box_i$ significa "i apariciones consecutivas de $\Box$" y C una cláusula CNF$\Box_m$.

Las formulas que se generan en CNF$\Box_m$ están controladas por cinco parámetros: la máxima profundidad modal D, el número de variables proposicionales N, el número de modalidades m, el número de cláusulas L y la probabilidad p de que un átomo que aparece a una profundidad menor que d sea puramente proposicional.

[2] En la práctica, una vez que un demostrador no puede resolver un problema en menos de 100 segundos, no tiene sentido intentar resolver los problemas más difíciles dentro de la misma clase.

Para que un conjunto de fórmulas generadas con 3CNF$\Box_m$ sea representativo se debe proceder con cuidado. En general, el procedimiento utilizado es el que describimos a continuación. Se comienza por fijar todos los parámetros con excepción de L. Valores chicos de L nos darán mayoría de fórmulas trivialmente satisfacibles, mientras que valores grandes nos darán fórmulas con demasiadas restricciones como para ser satisfacibles. Existe un valor intermedio de L en el cual las fórmulas satisfacibles e insatisfacibles son equiprobables; para cada configuración, este valor debe ser aproximado empíricamente. Se fija entonces un rango para L que cubra completamente la transición de "todas las fórmulas son satisfacibles" hasta "ninguna fórmula es satisfacible" y se determina una serie de valores para L dentro del rango que permitan obtener distintas mediciones en las cercanías de la zona de equiprobabilidad. Luego, para cada valor de L se genera un número fijo pero grande de fórmulas y las mismas fórmulas son procesadas por todos los demostradores, generalmente con un límite de tiempo. Los valores de distintos indicadores (e.g. mediana del tiempo requerido para resolver un problema, mediana del número de cláusulas generadas, etc) son graficados contra L o L/N para así poder sacar conclusiones.

La definición estándar de CNF$\Box_m$ aleatorio genera fórmulas estrictamente modales. En [Areces and Heguiabehere, 2003] se presenta una extensión para trabajar con demostradores para lógicas híbridas. Esta extensión, hCNF$\Box_m$, genera formulas para cualquier sublenguaje de $\mathcal{H}(@, A, \downarrow)$ y permite definir parámetros como la probabilidad de que un átomo sea un nominal, o de que un operador sea @. hGen es una herramienta que genera casos de test a partir de hCNF$\Box_m$.

5.3.2 Tests realizados y sus resultados

En las Figuras A.3 y A.4 del Apéndice A se muestran los posibles valores con los que puede parametrizarse HyLoRes 2.0. Con los valores de la primer figura puede configurarse la función de selección a usar, mientras que con los de la segunda se determina el mecanismo para elegir la cláusula dada. Un demostrador basado en resolución es en general sensible a estos parámetros; así como una buena función de selección acelera el proceso de saturación, la forma en que se elija la cláusula dada influirá directamente en la velocidad con que, en promedio, se encuentre una refutación.

Ahora bien, si sólo consideramos configuraciones de HyLoRes en las que todos estos parámetros aparezcan, veremos que el número de combinaciones posibles es de $9! \times 2 \times 5! = 87.091.200$. Si además consideramos configuraciones en las que algunos de los parámetros puedan no aparecer, este número será aun mayor. Claramente, no podemos probar *todas* las configuraciones posibles hasta encontrar la mejor[3].

La solución que encontramos a este problema consistió en utilizar una combinación de heurísticas tradicionales con pruebas empíricas cortas (y, por lo tanto, sin valor estadístico) hasta reducir el conjunto de parámetros a probar a un número suficientemente pequeño. El conjunto final, de 96 configuraciones, tenía las siguientes características:

- Para la complejidad de cláusulas, sólo se utilizaban los parámetros `s`, `V`, `D`, `L`, `M` y `B`.
- El primer parámetro de la complejidad de cláusulas era siempre `s`. De esta forma se privilegia a las cláusulas más chicas, lo cual es razonable si tratamos de priorizar la derivación de la cláusula vacía.
- El segundo parámetro de la complejidad de cláusulas era siempre `V` ó `D`. Estos valores se eligieron en base a pruebas preliminares cortas. Como estos valores dan una medida de la complejidad de la fórmula distinguida, podemos interpretar que, permiten privilegiar las cláusulas más simples.
- `M` y `B` aparecían siempre consecutivas. Esto se decidió por ser ambos valores covariantes.

[3] De hecho, si consideramos que correr los tests de BHS sobre *una* configuración de HyLoRes llevaba, en promedio, 25 minutos, concluiremos que sólo podemos probar una fracción casi despreciable del universo de configuraciones posibles.

- En la función de selección `n` y `a` eran siempre, respectivamente, el último y anteúltimo criterio utilizado. Esto se decidió en base a pruebas empíricas cortas.
- Si en la complejidad de cláusulas `M` aparecía antes que `B`, entonces en la función de selección `d` aparecía antes que `b`. Esto también se decidió en base a pruebas empíricas cortas.

Cada una de las configuraciones obtenidas se evaluó, a continuación, utilizando el conjunto de tests de BHS, utilizando un valor de *timeout* de 50 segundos, y lo mismo se hizo con la mejor configuración conocida de HyLoRes 1.0. En base a los resultados de esta corrida, se planteó una corrida adicional de 21 casos, formada por alteraciones de las restricciones originalmente impuestas, aplicadas sobre las mejores configuraciones obtenidas hasta el momento. La idea era tratar de comprobar que las restricciones originales no habían sido particularmente desafortunadas. Llamaremos "Corrida A" a la primer corrida de tests, y "Corrida B" a la corrida adicional.

La ventaja de utilizar BHS como *filtro grueso* radica en que, por su forma incremental, la presencia de configuraciones particularmente malas no afecta negativamente el tiempo total de ejecución[4]. La principal desventaja radica en que las fórmulas de este test pertenecen a $\mathcal{M}$ (i.e. no tienen nominales, ni el operador @), con lo cual, las configuraciones que mejor resultado den no necesariamente trabajarán bien cuando sea necesario usar la regla de paramodulación.

En la Figura 5.1, sintetizamos los resultados de esta primera evaluación. En ella aparecen graficadas la performance de:

- La configuración que mejor resultado general dio de la Corrida A, `sDMBLV-dboan`.
- Una de las configuraciones que peor resultado general dio, `sVLDBM-bodan`.
- Una configuración con resultado general intermedio pero que tuvo el mejor desempeño para el caso *k_dum_p*, `sVLBMD-Lbdoan`.
- La configuración con mejor resultado de la Corrida B, `sdDMBLvV-dboan`.
- HyLoRes 1.0.

En líneas generales, puede decirse que la performance de HyLoRes 2.0 parece ser levemente mejor que la de su antecesora, con la notable excepción del caso *k_grz_p*. Ninguna de las configuraciones de HyLoRes 2.0 probadas hasta el momento pudo resolver siquiera la fórmula más simple de *k_grz_p*; mientras que HyLoRes 1.0 pudo resolver 21 fórmulas en menos de 50 segundos.

Esto es algo que creemos que merece ser estudiado en mayor profundidad. Nuestra conjetura, por el momento es la siguiente. En HyLoRes 1.0 las fórmulas de la forma $@_i\,[r]\psi$ son consideradas *positivas*, mientras que las fórmulas de la forma $@_i\,\langle r\rangle j$ son consideradas *negativas* (por ser $@_i\,\langle r\rangle j \equiv @_i\,\neg[r]\neg j$). Luego, dada una cláusula de la forma $\{@_i\,[r]\psi, @_i\,\langle r\rangle j\} \cup C$, la función de selección que usa HyLoRes 1.0 *elegirá*, tal vez junto a algunas fórmulas de C, a $@_i\,\langle r\rangle j$, pero no a $@_i\,[r]\psi$. Este comportamiento de alguna manera limita la cantidad de aplicaciones de la regla ($[R]$) de una manera que en la versión 2.0 no se puede reproducir.

El siguiente paso fue comparar HyLoRes 1.0 y 2.0 usando tests generados con hCNF$\Box_m$ aleatorio. Para estas pruebas utilizamos las mismas configuraciones de la Figura 5.1. En todos los casos usamos un valor de *timeout* de 20 segundos y 50 fórmulas por punto.En este caso la performance de HyLoRes 2.0 fue claramente superior a la de la versión 1.0. En la Figura 5.2 se ve claramente cómo a HyLoRes 1.0 le cuesta resolver una fracción importante de los problemas más simples, mientras que HyLoRes 2.0 los resuelve todos. Es interesante notar que HyLoRes 1.0 tiene la mayor proporción de *timeouts* en la región en que la mayoría de las fórmulas son satisfacibles. Recordemos que las restricciones de orden y selección aceleran el proceso de saturación, y que en la versión 1.0 no se contaba con ningún mecanismo de este tipo para la regla de paramodulación.

Como ya se ve en la Figura 5.3, tener un número grande de *timeouts* afecta negativamente la representatividad de los valores graficados. La curva cortada en el gráfico de cláusulas generadas, y la forma poco usual de la curva del gráfico de tiempo de ejecución son una muestra elocuente

[4]De hecho, es de esperar que cuanto peor sea una configuración, antes termine.

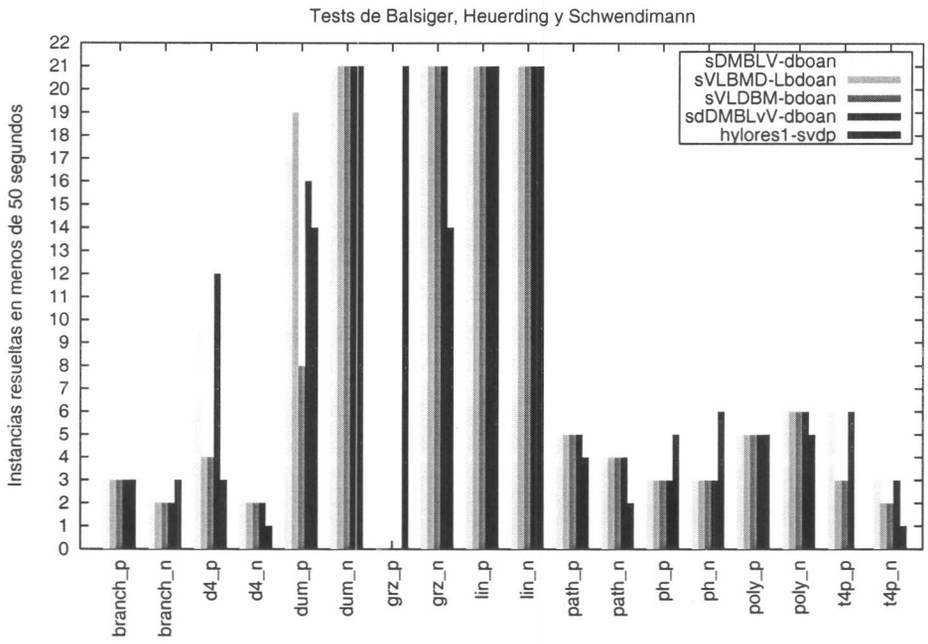

Figura 5.1: Evaluación con BHS de HyLoRes 1.0 y distintas configuraciones de HyLoRes 2.0

de esto. Por otro lado, se puede ver que para valores chicos de L el tiempo de inicialización de HyLoRes 2.0 es mayor al tiempo que necesita para resolver el problema en sí.

El número de casos que HyLoRes 1.0 no puede resolver cuando se aumenta la profundidad modal de las fórmulas generadas es demasiado grande como para que podamos tenerlo en cuenta[5]. Es por ello que las últimas pruebas se realizaron sólo sobre distintas configuraciones de la versión 2.0.

En las Figuras 5.4 y 5.5 se muestran los resultados para $V = 5$ y $D = 4$, valores para los cuáles HyLoRes 2.0 todavía tiene un número bajo de *timeouts*. Recién cuando pasamos a valores más altos como $V = 8$ y $D = 7$ (Figuras 5.6 y 5.7) encontramos un número relativamente alto de *timeouts*; aunque aun con estas fórmulas HyLoRes 2.0 presenta un comportamiento mejor que el que la versión 1.0 tiene con las fórmulas más simples.

En todos los casos vemos que todas las configuraciones probadas tuvieron un comportamiento aceptable, aunque aquellas que dieron mejores resultados con BHS siguen teniendo mejor performance en hCNF$\Box_m$.

[5]Además, a diferencia de lo que pasaba con BHS, el tiempo necesario pasar este test cuando la performance general no es buena se hace demasiado alto.

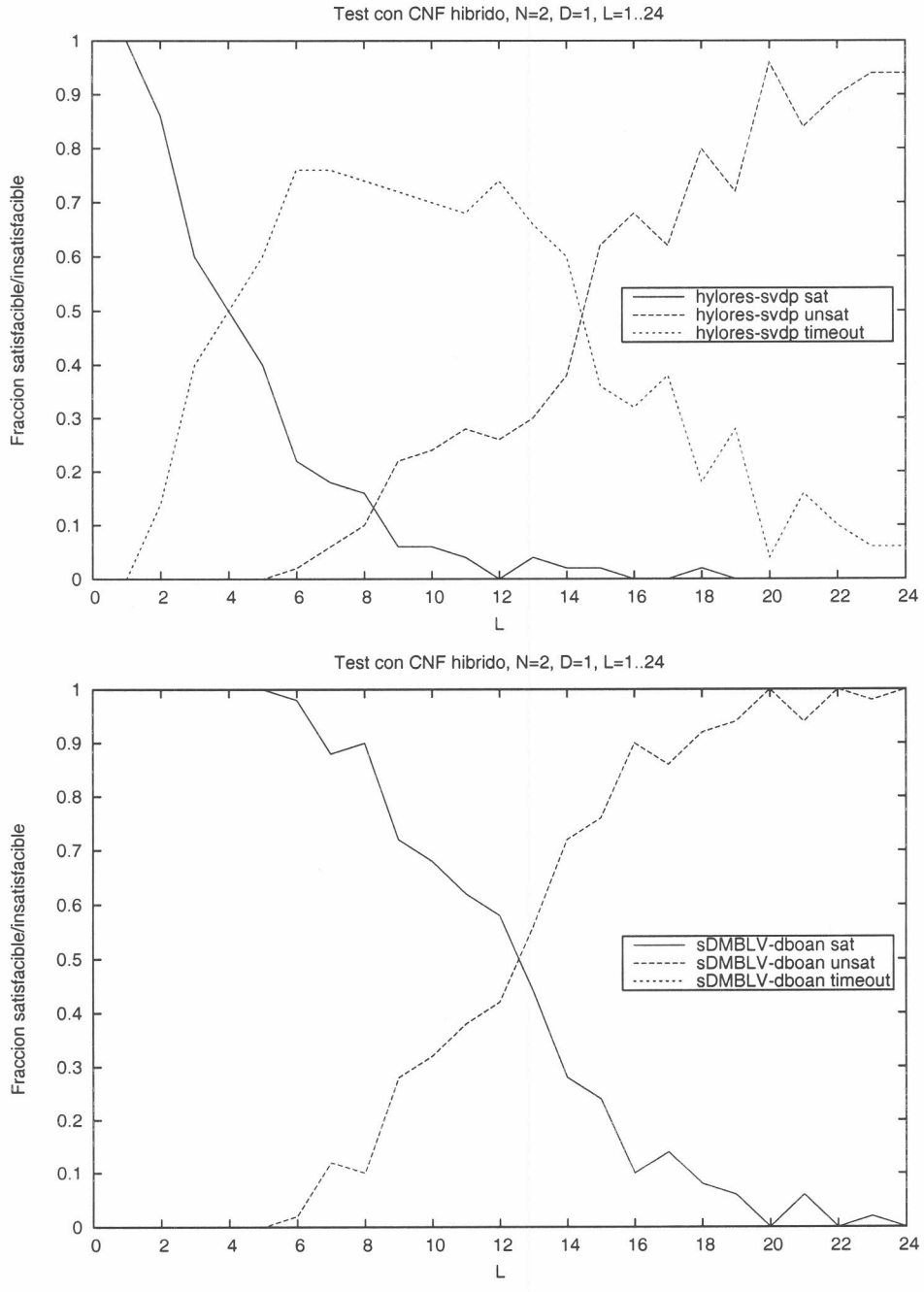

Figura 5.2: Fracción de satisfacibilidad y de timeout, para HyLoRes 1.0 y 2.0 – Fórmulas simples de hCNF$\Box_m$.

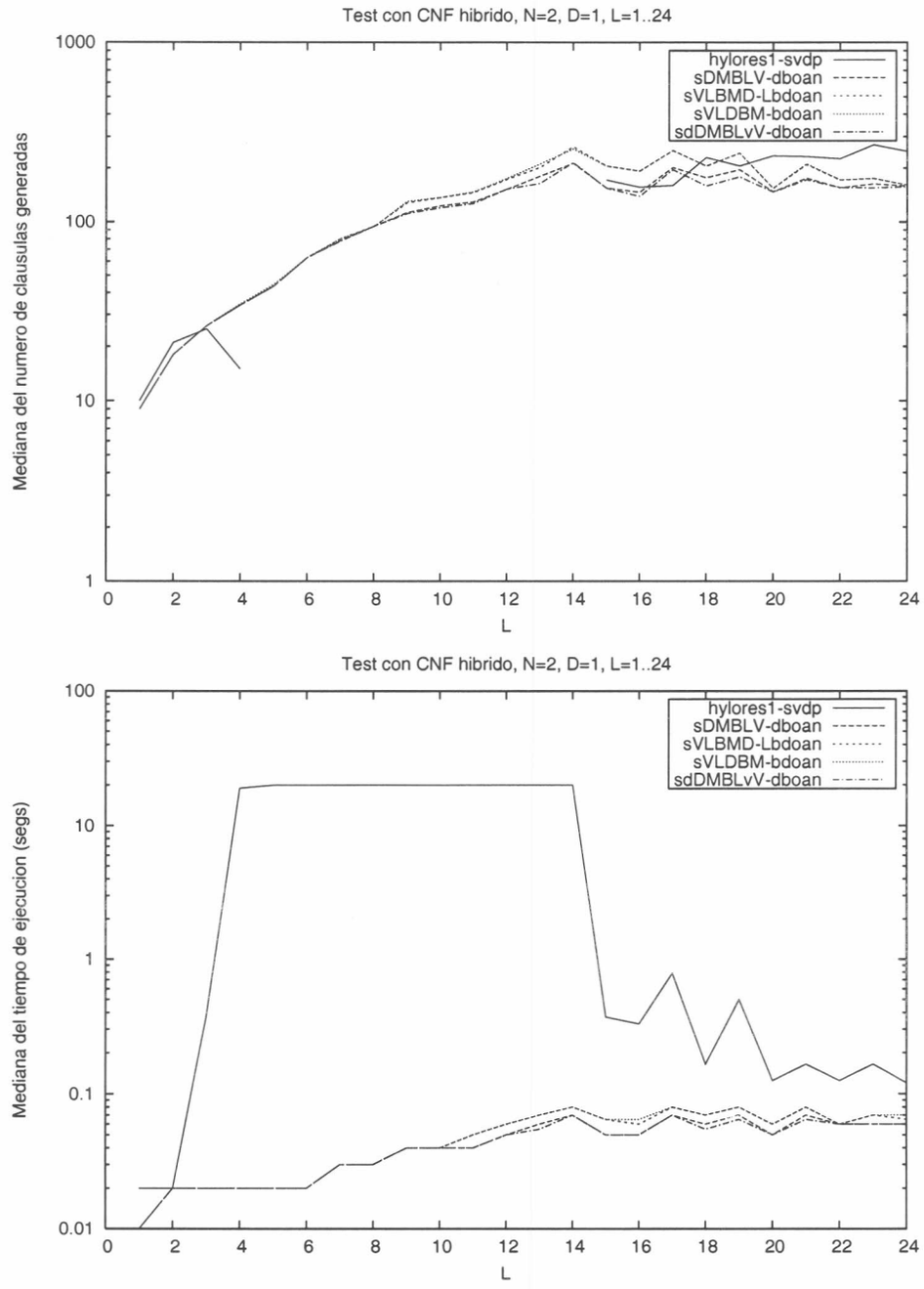

Figura 5.3: Evaluación con hCNF$\square_m$ de HyLoRes 1.0 y 2.0 – Fórmulas simples.

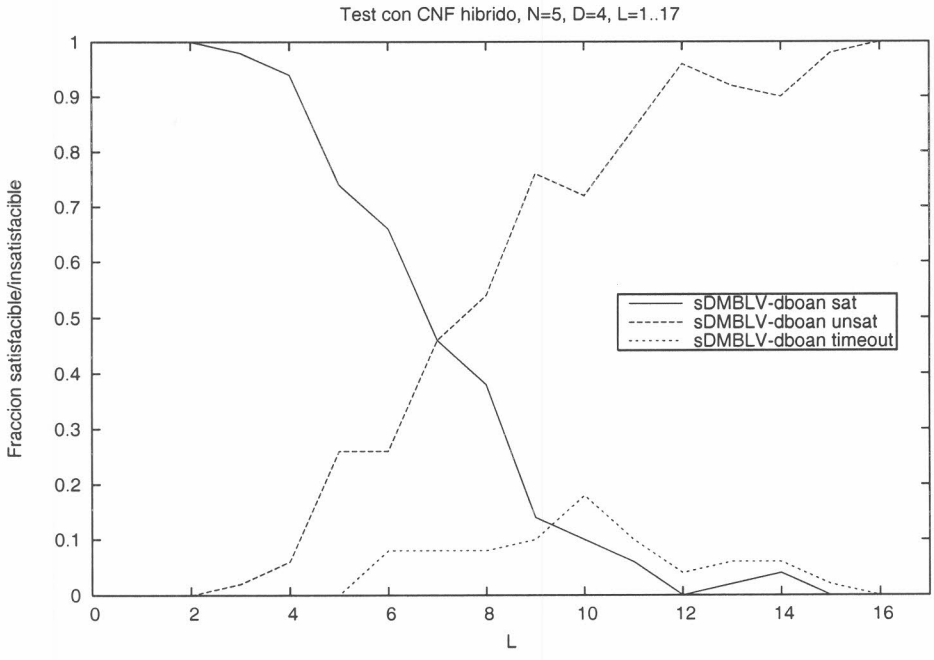

Figura 5.4: Fracción de satisfacibilidad y de timeout, para HyLoRes 2.0 – Fórmulas medianas de hCNF$\square_m$.

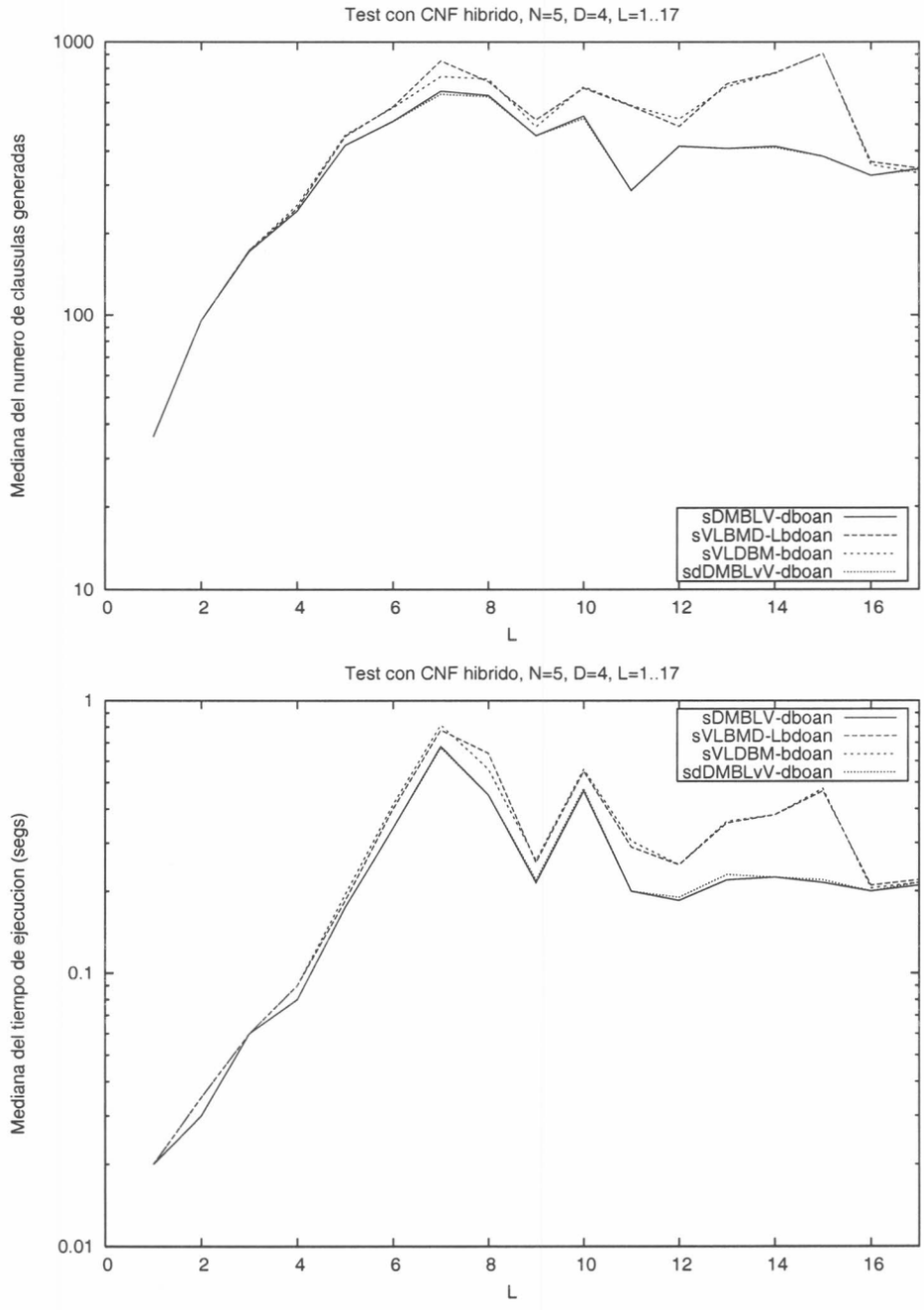

Figura 5.5: Evaluación con hCNF$\square_m$ de HyLoRes 2.0 – Fórmulas medianas.

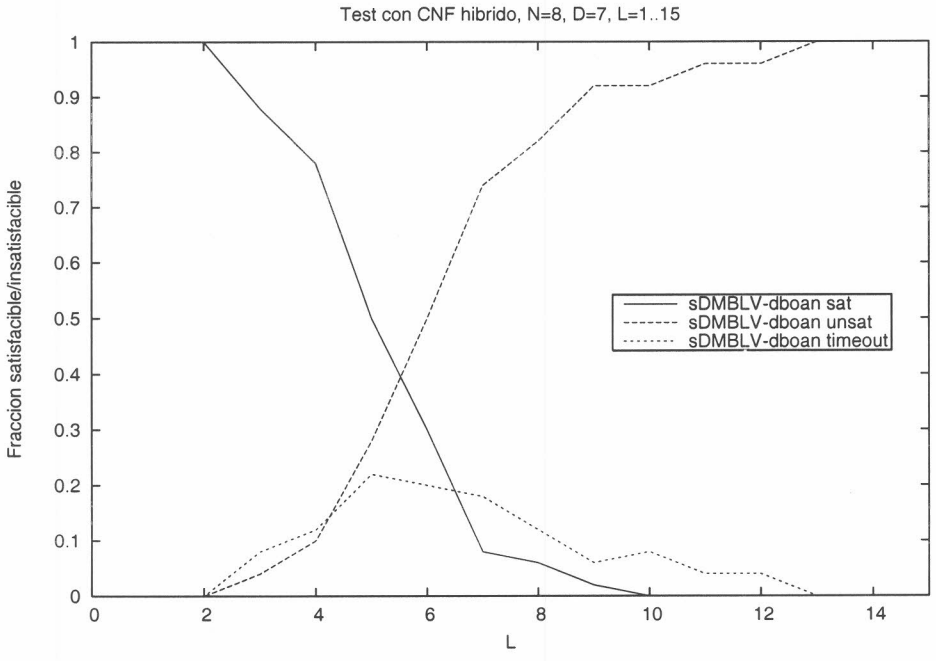

Figura 5.6: Fracción de satisfacibilidad y de timeout, para HyLoRes 2.0 – Fórmulas grandes de hCNF$\Box_m$.

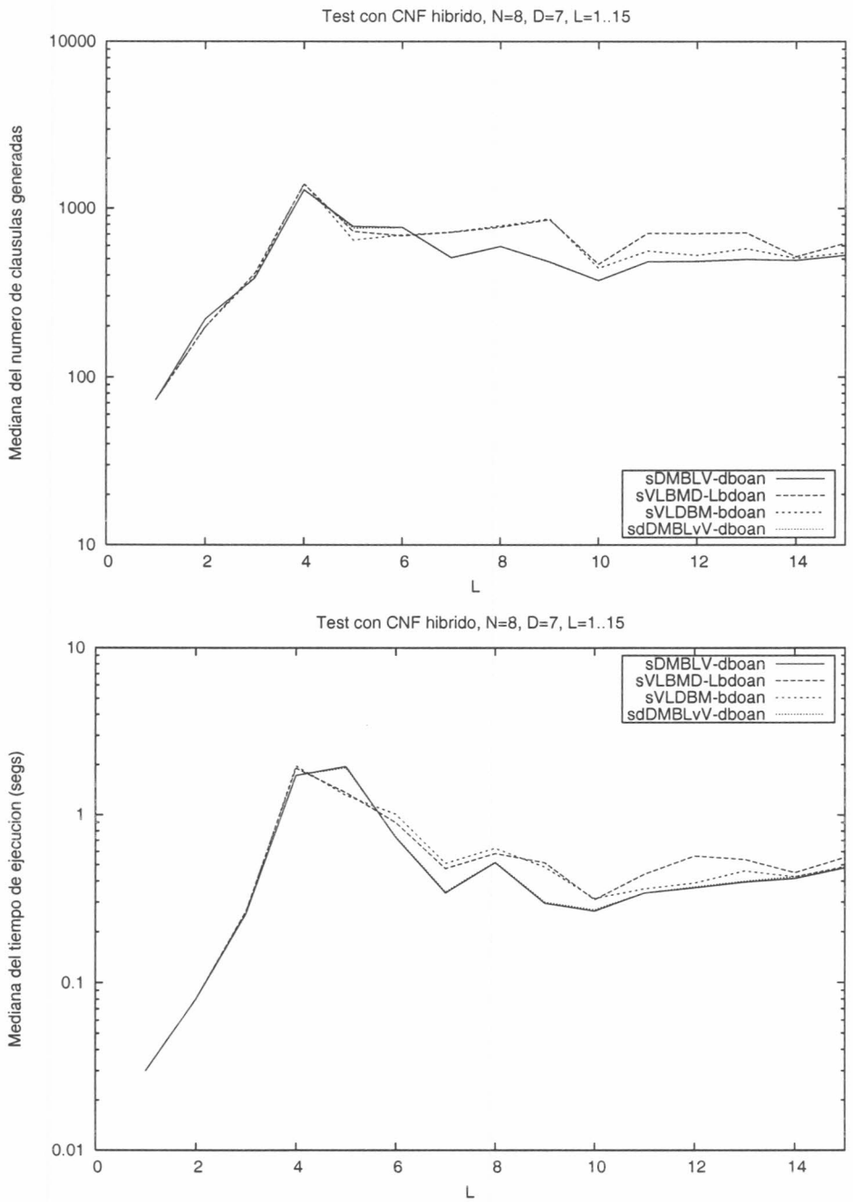

Figura 5.7: Evaluación con hCNF$\square_m$ de HyLoRes 2.0 – Fórmulas grandes.

Apéndice A

Descripción técnica de HyLoRes 2.0

HyLoRes está escrito íntegramente en Haskell y se compila usando el Glasgow Haskell Compiler (GHC) [GHC, 2004]. Este compilador genera código intermedio en C, el cuál a su vez es compilado utilizando el GCC [GCC, 2004]. De esta forma, HyLoRes puede correr sobre cualquier plataforma que soporte a este popular compilador de C de GNU.

La distribución estándar del GHC incluye la librería Edison que contiene implementaciones eficientes de diversas tipos de datos, principalmente colecciones. Algunas de estas colecciones son usadas en HyLoRes. Además, para poder compilar HyLoRes es necesario generar previamente, a partir de una gramática, un parser de fórmulas. Para ello se utiliza Happy [Happy, 2004], que es un generador de parsers LALR para Haskell.

HyLoRes es una aplicación de línea de comando. Al ejecutarla se debe indicar, como mínimo, el nombre del archivo que posee la fórmula cuya satisfacibilidad se desea determinar. Adicionalmente se pueden especificar otros parámetros como ser la función de selección a utilizar, el mecanismo de prioridad para elegir la cláusula dada, o el tiempo máximo de ejecución. Como resultado, además de indicarse si la fórmula es o no satisfacible, se imprime el tiempo total de ejecución en segundos y el número total de cláusulas generadas[1].

A continuación damos una descripción bottom-up de los módulos que constituyen HyLoRes. Las dependencias entre ellos se puede ver en la Figura A.1, mientras que la Figura A.2 muestra la cantidad de líneas por módulo.

A.1 Fórmulas

Las fórmulas se representan en HyLoRes de dos formas distintas: como tipo algebraico (`Formula`) y como número entero (`IFormula`). Esta última forma de representación permite eficientes comparaciones de igualdad entre fórmulas. Para poder pasar de una representación a la otra, se utiliza un diccionario de doble entrada. Los módulos involucrados son:

Formula.hs En este módulo se define el tipo `Formula`, sus operaciones y distintas funciones sobre fórmulas usadas por el resto de los módulos. También se define un orden admisible para terminación y un distribuidor de nominales, siguiendo la idea de la Sección 4.3.3. Se proveen distintas funciones para armar instancias de fórmulas, y si bien no hay restricciones sobre la forma en que se debe construir una fórmula, el invariante de representación del tipo garantiza que ésta se encuentra siempre en NNF y con las igualdades normalizadas.

[1] Estrictamente hablando, lo que se muestra es el número total de cláusulas que se agregan a clauses. Este número probablemente sea menor que la suma de todas las cláusulas que en algún momento estuvieron en new.

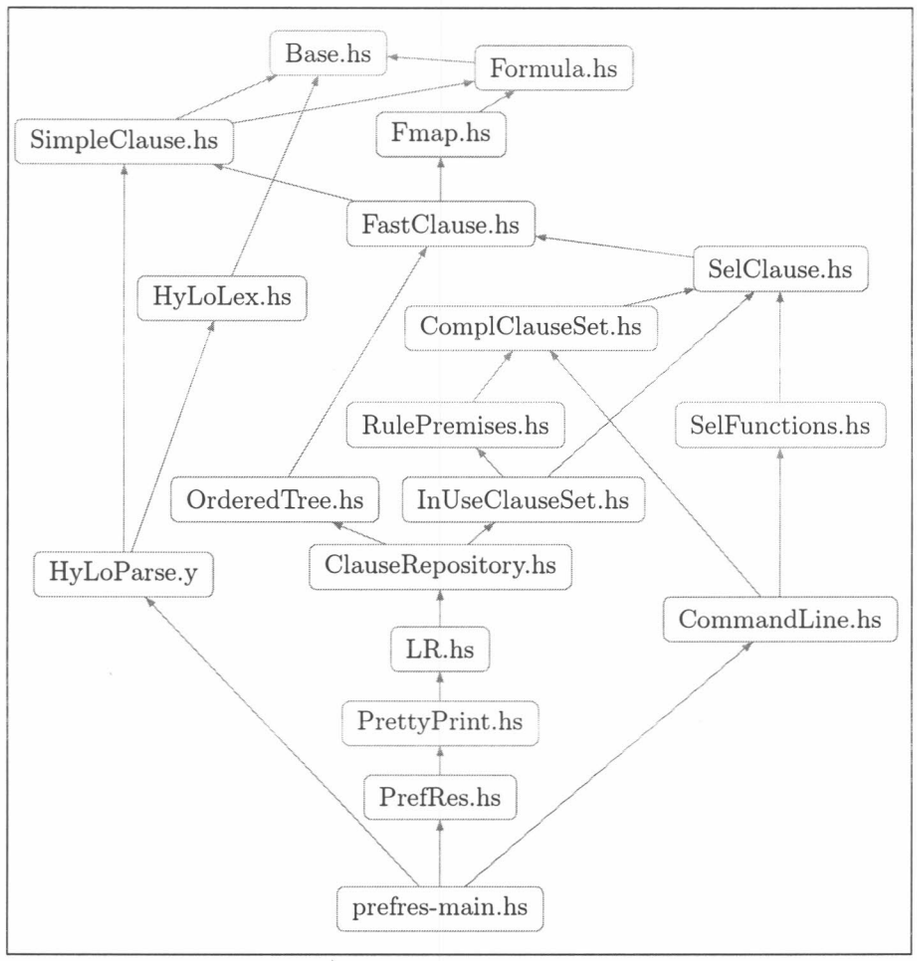

Figura A.1: Dependencias entre módulos (se omiten las dependencias transitivas)

Fmap.hs En este módulo se definen los tipos `IFormula` y `FormRep`. Este último es el tipo que trabaja como diccionario de doble entrada; y es el encargado de asociarle una instancia de `IFormula` a cada `Formula` que en él se ingresa. Esta asignación garantiza que si φ está asociado a x, $\neg\varphi$ estará asociado a $-x$.

A.2 Cláusulas y funciones de selección

Hay varios tipos en HyLoRes que representan cláusulas. Por una cuestión de eficiencia, cada uno de los conjuntos de cláusulas involucrados en el algoritmo de la cláusula dada utiliza un tipo distinto de cláusula; cada una de ellas está generalmente optimizada para el tipo de operaciones que se realizan en cada etapa del algoritmo.

SimpleClause.hs En este módulo se define el tipo más simple de cláusula. Es sólo un conjunto de instancias `Formula`. Es el tipo de cláusulas que se usa en new.

FastClause.hs Este tipo de cláusula representa cada fórmula dualmente como `Formula` y como `IFormula`. Internamente, las fórmulas se guardan utilizando un *heap*, lo cual garantiza que la fórmula maximal de una cláusula de este tipo se puede acceder en tiempo constante. Cada instancia

Módulo	# de líneas
Base.hs	139
ClauseRepository.hs	350
CommandLine.hs	131
ComplClauseSet.hs	178
FastClause.hs	150
Fmap.hs	109
Formula.hs	931
HyLoLex.hs	154
HyLoParse.y	176
InUseClauseSet.hs	222
LR.hs	239
OrderedTree.hs	204
PrefRes.hs	722
PrettyPrint.hs	215
RulePremises.hs	86
SelClause.hs	173
SelFunctions.hs	224
SimpleClause.hs	170
prefres-main.hs	170
Total:	4743

Figura A.2: Cantidad de líneas por módulo.

se construye a partir de una `SimpleClause` y de un `FormRep`. Una vez construidas, no pueden agregarse nuevas fórmulas a estas cláusulas, pero sí pueden eliminarse.

SelClause.hs Una `SelClause` es una `FastClause` con una *fórmula distinguida*, como resultado de aplicarle una función de selección. Esta fórmula distinguida puede obtenerse en tiempo constante. En este módulo se define el tipo `SelFunc`, cuyas instancias pueden representar cualquier función de selección.

SelFunctions.hs En este módulo se definen algunas funciones de selección triviales y dos generadores paramétricos de funciones de selección. Éstos reciben una lista de funciones booleanas que determinan *clases de fórmulas* (e.g. fórmulas de la forma $@_i\,[r]\psi$, $@_i\,(\psi_1 \wedge \psi_2)$, etc.). Uno de ellos genera una función de selección que determina cuál es la primer clase de fórmulas que tiene algún miembro en la cláusula, y de encontrar alguna, selecciona la menor posible. El otro generador devuelve una función de selección similar, pero en este caso selecciona la menor de las fórmulas que tenga como *prefijo* el nominal con menor nivel (respecto a la función $level(x)$). Las clases de fórmulas pueden indicarse alternativamente con un string que las codifica. Usando esta codificación (cuya clave se muestra en la Figura A.3) es posible experimentar con distintas configuraciones en forma simple, por ejemplo, a través de la línea de comando.

En `clauses` e `in-use` se utiliza un tipo de cláusula llamada `TaggedClause`, que no es más que una `SelClause` *etiquetada* con un *número de cláusula*. Este tipo se define en el módulo **ComplClauseSet.hs** que se describe a continuación.

A.3 Conjuntos de cláusulas

Los conjuntos de cláusulas que se utilizan en el algoritmo de la cláusula dada no merecieron sólo distintas implementaciones de cláusulas. Cada uno de ellos se implementó por separado, y se los dotó con interfaces y estructuras de datos especialmente pensados para resolver en forma eficiente

Código	Descripción
L	Como primer elemento del string, indica que se desea elegir la menor fórmula de las que tengan el prefijo de menor nivel dentro de la clase apropiada de fórmulas.
n	La clase de fórmulas de la forma $@_i \neg a$ con $a \in$ ATOM.
a	La clase de fórmulas de la forma $@_i (\psi_1 \wedge \psi_2)$.
o	La clase de fórmulas de la forma $@_i (\psi_1 \vee \psi_2)$.
d	La clase de fórmulas de la forma $@_i \langle r \rangle \psi$ con $\psi \notin$ NOM.
b	La clase de fórmulas de la forma $@_i [r] \psi$.

Figura A.3: Codificación para generar funciones de selección.

las operaciones que de ellos se espera. De todas formas, el cálculo conoce una única interfaz al conjunto de todas las cláusulas, que es a través del tipo `ClauseRepository`.

RulePremises.hs En este módulo se define un tipo que representa un conjunto de cláusulas dividido en dos: aquellas que pueden ser premisa principal de una regla y aquellas que pueden ser premisa auxiliar de la misma regla. Se utiliza en **InUseClauseSet.hs**.

InUseClauseSet.hs Este es el tipo de datos que se utiliza para implementar el conjunto de cláusulas in-use. Al agregar una cláusula a este conjunto, la *clasifica* de acuerdo al rol que puede cumplir de acuerdo a su fórmula distinguida. De esta manera, puede devolver en forma eficiente las cláusulas que pueden ser premisa de cualquiera de las reglas binarias.

ComplClauseSet.hs Este es el tipo usado para implementar clauses y está especialmente diseñado para facilitar la elección de la *cláusula dada*. Se construye a partir de un string que *codifica* una secuencia de *indicadores* de una cláusula. Los posibles indicadores se muestran en la Figura A.4. A cada cláusula que se agrega a este conjunto se le asigna un número creciente de cláusula, y se le calculan los indicadores correspondiente. Las cláusulas se almacenan en un *heap*, y se las ordena en primer término usando el valor de sus indicadores (en el orden indicados en el string que los codifica), de menor a mayor. De esta forma, la cláusula dada se puede obtener en tiempo constante, salvo cuando corresponde elegir la más antigua, operación para la cual es necesario recorrer todo el conjunto.

Código	Descripción
s	Tamaño de la cláusula
v	La suma del número de átomos (no necesariamente distintos) que aparecen en la cláusula
V	La cantidad de átomos (no necesariamente distintos) que aparecen en la fórmula distinguida
d	La máxima profundidad modal de las fórmulas de la cláusula
D	La profundidad modal de la fórmula distinguida
l	El menor nivel de prefijo de todas las fórmulas de la cláusula
L	El nivel del prefijo de la fórmula distinguida
M	1 si la fórmula distinguida es de la forma $@_i \langle r \rangle \psi$ ($\psi \notin$ NOM), 0 en caso contrario
B	1 si la fórmula distinguida es de la forma $@_i \Box r \psi$, 0 en caso contrario

Figura A.4: Indicadores utilizables para la codificación del mecanismo de selección de la cláusula dada

OrderedTree.hs En este módulo se definen los tipos y las operaciones necesarias para realizar las operaciones eliminación hacia adelante de cláusulas subsumidas (forward subsumption deletion) presentado en la Sección 5.1.2. El tipo `OTree` representa un *trie* donde cada nodo es una `IFormula`,

y cada rama, que se encuentra ordenada (numéricamente), representa una cláusula. Un `OTree` cuya raíz sea x representa un conjunto de cláusulas tales que su mayor fórmula (numéricamente) es x. De esta forma, es posible representar el conjunto de cláusulas generado hasta el momento utilizando una lista de `OTree`, ordenada por el valor de la raíz de cada uno. Además, se puede determinar si una cláusula determinada es subsumida por el conjunto de cláusulas representando la cláusula como una lista ordenada de `IFormula`, y buscando si alguno de los `OTree` contiene una rama que sea prefijo de algún sufijo (no necesariamente propio) de dicha lista.

ClauseRepository.hs El tipo definido en este módulo representa el conjunto de todas las cláusulas generadas, dividido en new, clauses e in-use. Almacena también el `FormRep` que permite traducir todas las `Formula` generadas a su `IFormula` asociada, la lista de `OTree` utilizada para detectar subsunción, la cantidad de cláusulas agregadas a clauses hasta el momento y la función de selección que se utiliza durante el cálculo.

A.4 Motor monádico y ciclo principal

Haskell es un lenguaje puramente funcional. El mecanismo estándar para manejar *estado* y *entrada/salida* en este tipo de lenguajes es utilizando *mónadas*. Una mónada, en general, puede ser *de estado*, *de entrada/salida* o *mixta*, como es el caso de la que se utiliza en HyLoRes.

LR.hs En este módulo se define el tipo `LR` que representa el estado del demostrador en un momento dado. Éste consta de un `ClauseRepository`, un entero que cuenta la cantidad de iteraciones del algoritmo de la cláusula dada realizadas hasta el momento y una lista de parámetros tales como el tipo fórmula que se está tratando (e.g. $\mathcal{M}$, $\mathcal{H}(@)$, etc.) o el tipo de logging especificado. Aquí también se define la mónada, las operaciones para inicializarla, modificar su estado, e imprimir mensajes.

PrefRes.hs En este módulo se implementa el algoritmo de la cláusula dada, incluyendo la selección de la cláusula dada y el conjunto de reglas del cálculo.

A.5 Interfaz con el usuario

HyLoLex.hs En este módulo se implementa el analizador lexicográfico usado para realizar el parsing de fórmulas.

HyLoParse.y Este archivo contiene la gramática que se utiliza para generar, usando Happy, el parser de fórmulas. La Figura A.5 muestra un ejemplo de la entrada que este parser reconoce.

PrettyPrint.hs Este módulo contiene funciones utilizadas principalmente para hacer logging del estado del demostrador y de las reglas aplicadas.

CommandLine.hs Las funciones de este módulo se encargan principalmente de reconocer las opciones que el usuario ingresa por línea de comando. Éstas se muestran en la Figura A.6; la única obligatoria es `-f`.

prefres-main.hs Este módulo contiene la función principal de la aplicación, procesa los parámetros, lee la fórmula del archivo, crea la mónada, prepara un thread para que controle el timeout (si se especifica) y ejecuta el ciclo principal.

```
begin
!(
  (
   (<><><><>(n1:n2)) &
   n1:(
       ([]<>(p1 v n2:(p1 <-> <>p3))) ->
       (<>p1 <-> []<>(false v n2:(n3 & P4)))
      )
  ) ->
  (
   n2:([]<>(p1 v n2:(p1 <-> <>p3))) ->
   n1:(<>p1 <-> []<>(false v n2:(n3 & P4)))
  )
 )
end
```

Figura A.5: Ejemplo de archivo de entrada para HyLoRes.

Opción	Descripción
`-t` *segs*	Fija el tiempo de ejecución máxima en segundos.
`-o` *string*	Indica el orden entre cláusulas que se debe usar para elegir la cláusula dada.
`-sf` *string*	Indica la función de selección.
`-r`	Imprime las reglas que se aplican durante el cálculo.
`-s`	Imprime el estado del demostrador luego de cada iteración.

Figura A.6: Opciones de línea de comando de HyLoRes 2.0

A.6 Misceláneas

Base.hs En este módulo se definen varias funciones de uso general.

```
$ ./hylores -r -f unsat/test12.frm

Input:
 { [@(N2, <R1>-P1)],
   [@(N1, [R1]P1)],
   [@(N2, N1)] }
End of input

Given: (1, [@(N2, <R1>-P1)+*])

DIA: { [@(N2, <R1>N_{@(N2, <R1>-P1)})],
       [@(N_{@(N2, <R1>-P1)}, -P1)] }

Given: (3, [@(N2, N1)+])

Given: (2, [@(N1, [R1]P1)+*])

Given: (4, [@(N2, <R1>N_{@(N2, <R1>-P1)})+])

PAR-RELM: { [@(N1, <R1>N_{@(N1, <R1>-P1)})],
            [@(N_{@(N2, <R1>-P1)}, N_{@(N1, <R1>-P1)})] }

Given: (6, [@(N1, <R1>N_{@(N1, <R1>-P1)})+])

BOX: (2, [@(N_{@(N1, <R1>-P1)}, P1)])

Given: (5, [@(N_{@(N2, <R1>-P1)}, -P1)+*])

Given: (7, [@(N_{@(N2, <R1>-P1)}, N_{@(N1, <R1>-P1)})+])

PARX (N_{@(N2, <R1>-P1)} --> N_{@(N1, <R1>-P1)}):
     { [@(N_{@(N1, <R1>-P1)}, -P1)] }

Given: (8, [@(N_{@(N1, <R1>-P1)}, P1)+])

Given: (9, [@(N_{@(N1, <R1>-P1)}, -P1)+*])

The formula is unsatisfiable
Clauses generated: 9
Elapsed time: 0.0
```

Figura A.7: Ejemplo de ejecución de HyLoRes 2.0

Bibliografía

[Andréka *et al.*, 1998] H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.

[Areces and de Rijke, 2001] C. Areces and M. de Rijke. From description to hybrid logic, and back. In *Advances in Modal Logic, Volume 3*. CSLI Publications, 2001.

[Areces and Heguiabehere, 2002] C. Areces and J. Heguiabehere. HyLoRes: A hybrid logic prover based on direct resolution. In *Advances in Modal Logic*, Toulouse, France, 2002.

[Areces and Heguiabehere, 2003] C. Areces and J. Heguiabehere. hGen: A random CNF formula generator for Hybrid Languages. In *Proceedings of Methods for Modalities 3*, Nancy, France, 2003.

[Areces *et al.*, 1999] C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In J. Flum and M. Rodríguez-Artalejo, editors, *Computer Science Logic*, number 1683 in LNCS, pages 307–321. Springer, 1999. Proceedings of the 8th Annual Conference of the EACSL, Madrid, September 1999.

[Areces *et al.*, 2000] C. Areces, R. Gennari, J. Heguiabere, and M. de Rijke. Tree-based heuristics in modal theorem proving. In W. Horn, editor, *Proceedings of ECAI'2000*, pages 199–203, Berlin, Germany, 2000.

[Areces *et al.*, 2001] C. Areces, H. de Nivelle, and M. de Rijke. Resolution in modal, description and hybrid logic. *Journal of Logic and Computation*, 11(5):717–736, 2001.

[Areces, 2000] C. Areces. *Logic Engineering. The Case of Description and Hybrid Logics.* PhD thesis, Institute for Logic, Language and Computation, University of Amsterdam, Amsterdam, The Netherlands, October 2000.

[Auffray *et al.*, 1990] Y. Auffray, P. Enjalbert, and J. Hebrard. Strategies for modal resolution: results and problems. *Journal of Automated Reasoning*, 6(1):1–38, 1990.

[Baader *et al.*, 2003] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[Bachmair and Ganzinger, 1998] L. Bachmair and H. Ganzinger. Equational reasoning in saturation-based theorem proving. In *Automated deduction—a basis for applications, Vol. I*, pages 353–397. Kluwer Acad. Publ., Dordrecht, 1998.

[Bachmair and Ganzinger, 2001] L. Bachmair and H. Ganzinger. Resolution theorem proving. In J. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 2, pages 19–99. Elsevier, Amsterdam, the Netherlands, 2001.

[Beckert and Goré, 1997] Bernhard Beckert and Rajeev Goré. Free variable tableaux for propositional modal logics. In *Proceedings, International Conference on Theorem Proving with Analytic Tableaux and Related Methods, Pont-a-Mousson, France*, LNCS 1227, pages 91–106. Springer, 1997.

[Blackburn *et al.*, 2001] P. Blackburn, A. Burchard, and S. Walter. Hydra: a tableaux-based prover for basic hybrid logic. In C. Areces and M. de Rijke, editors, *Proceedings of Methods for Modalities 2*, Amsterdam, The Netherlands, November 2001.

[Blackburn *et al.*, 2002] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic.* Cambridge University Press, 2002.

[Blackburn, 2000] P. Blackburn. Internalizing labelled deduction. *Journal of Logic and Computation*, 10(1):137–168, 2000.

[Buss, 1995] S. Buss. On Herbrand's theorem. In *Logic and Computational Complexity*, number 960 in Lecture Notes in Computer Science, pages 195–209. Springer-Verlag, 1995.

[Dershowitz and Jouannaud, 1990] N. Dershowitz and J. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B: Formal Models and Sematics (B)*, pages 243–320. Elsevier and MIT Press, 1990.

[Enjalbert and del Cerro, 1989] P. Enjalbert and L. Fariñas del Cerro. Modal resolution in clausal form. *Theoretical Computer Science*, 65(1):1–33, 1989.

[Fariñas del Cerro, 1982] L. Fariñas del Cerro. A simple deduction method for modal logic. *Information Processing Letters*, 14:49–51, April 1982.

[Fitting, 1996] Melvin C. Fitting. *First-Order Logic and Automated Theorem Proving.* Graduate Texts in Computer Science. Springer-Verlag, Berlin, 2nd edition, 1996.

[GCC, 2004] GCC Home Page. http://gcc.gnu.org/, 2004. Last access: 02/2004.

[Gerth *et al.*, 1995] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification Testing and Verification*, pages 3–18, Warsaw, Poland, 1995. Chapman & Hall.

[GHC, 2004] The Glasgow Haskell Compiler. `http://www.haskell.org/ghc/`, 2004. Last access: 02/2004.

[Giunchiglia *et al.*, 2002] Enrico Giunchiglia, Armando Tacchella, and Fausto Giunchiglia. SAT-based decision procedures for classical modal logics. *Journal of Automated Reasoning*, 28(2):143–171, 2002.

[Goré, 1999] Rajeev Goré. Tableau methods for modal and temporal logics. In Dov Gabbay, Marcello D'Agostino, Reiner Hähnle, and Joachim Posegga, editors, *Handbook of Tableau Methods*, pages 297–396. Kluwer Publishing Company, 1999.

[Haarslev and Möller, 2001] Volker Haarslev and Ralf Möller. RACER system description. In R.Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning. 1st. International Joint Conference, IJCAR 2001*, number 2083 in LNAI, Siena, Italy, June 2001.

[Happy, 2004] Happy: The Parser Generator for Haskell. http://www.haskell.org/happy/, 2004. Last access: 02/2004.

[Heguiabehere and de Rijke, 2001] J. Heguiabehere and M. de Rijke. The random modal QBF test set. In *Proceedings IJCAR Workshop on Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics*, pages 58–67, 2001.

[Heguiabehere, 2002] J. Heguiabehere. *Building Logic Toolboxes.* PhD thesis, Institute for Logic, Language and Computation, University of Amsterdam, Amsterdam, The Netherlands, December 2002.

[Herbrand, 1930] J. Herbrand. *Recherches sur la théorie de la démonstrations.* PhD thesis, Sorbone, Paris, 1930. Reprinted in W. Goldfarb, editor, *Logical Writings.* Reidel, 1971.

[Holzmann, 1997] Gerard J. Holzmann. The model checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[Holzmann, 2003] Gerard J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual.* Pearson Educational, 2003.

[Hustadt and Schmidt, 2000] U. Hustadt and R. A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In R. Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 67–71. Springer, 2000.

[Hustadt *et al.*, 1998] U. Hustadt, R. A. Schmidt, and C. Weidenbach. Optimised functional translation and resolution. In H. de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX'98, Oisterwijk, The Netherlands, Proceedings*, volume 1397 of *Lecture Notes in Computer Science*, pages 36–37. Springer, May 5–8 1998.

[HyLoRes, 2004] HyLoRes' Home Page. `http://www.loria.fr/~areces/HyLoRes/`, 2004. Last access: 01/2004.

[Kripke, 1959] S. Kripke. A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24:1–14, 1959.

[Kripke, 1963a] S. Kripke. Semantic analysis of modal logics I, normal propositional calculi. *Zeitschrift fürr mathemathische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[Kripke, 1963b] S. Kripke. Semantical consideration on modal logics. *Acta Philosophica Fennica*, 16:83–94, 1963.

[Lewis, 1918] Clarence Irving Lewis. *A Survey of Symbolic Logic.* Univ. of California Press (Berkeley), Berkeley, 1918. Reprint of Chapters I–IV by Dover Publications, 1960, New York.

[Mints, 1989] G. Mints. Resolution calculi for modal logics. *American Mathematical Society Translations*, 143:1–14, 1989.

[Mitchell *et al.*, 1992] D. Mitchell, B. Sleman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. of the 10th National Conference on Artificial Intelligence*, pages 459–465, 1992.

[Möller and Haarslev, 2003] R. Möller and V. Haarslev. Description logics systems. In Baader et al. [2003], chapter 8.

[MSPASS, 2004] MSPASS. `http://www.cs.man.ac.uk/~schmidt/mspass/`, 2004. Last access: 01/2004.

[Papadimitriou, 1994] Ch. Papadimitriou. *Computational Complexity.* Addison–Wesley, 1994.

[Patel-Schneider and Sebastiani, 2003] P. Patel-Schneider and R. Sebastiani. A new general method to generate random modal formulae for testing decision procedures. *Journal of Artificial Intelligence Research*, 18:351–389, May 2003.

[Prior, 1957] A. Prior. *Time and Modality.* Oxford Unviersity Press, 1957.

[Prior, 1967] A. Prior. *Past, Present and Future.* Oxford University Press, 1967.

[RACER, 2004] RACER System Description. `http://www.sts.tu-harburg.de/~r.f.moeller/racer/`, 2004. Last access: 01/2004.

[Robinson, 1965] J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[*SAT, 2004] *SAT Project. `http://www.mrg.dist.unige.it/~tac/StarSAT.html`, 2004. Last access: 01/2004.

[Schild, 1991] K. Schild. A correspondence theory for terminological logics. In *Proc. of the 12th IJCAI*, pages 466–471, 1991.

[Smullyan, 1994] Raymond M Smullyan. *First-Order Logic.* Dover Press, New York, revised edition, 1994.

[SPASS, 2004] SPASS Home Page. `http://spass.mpi-sb.mpg.de/`, 2004. Last access: 01/2004.

[SPIN, 2004] ON-THE-FLY, LTL MODEL CHECKING with SPIN. `http://www.spinroot.com`, 2004. Last access: 01/2004.

[Sutcliffe and Suttner, 1998] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[Voronkov, 2001] A. Voronkov. Algorithms, datastructures, and other issues in efficient automated deduction. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning. 1st. International Joint Conference, IJCAR 2001*, number 2083 in LNAI, pages 13–28, Siena, Italy, June 2001.

[Weidenbach *et al.*, 2002] Chr. Weidenbach, U. Brahm, Th. Hillenbrand, E. Keen, Chr. Theobald, and D. Topić. SPASS version 2.0. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, LNAI, pages 275–279. Springer-Verlag, 2002.