Facultad de Matemática, Astronomía y Física

Trabajo especial de la Licenciatura en Ciencias de la Computación

Optimización de la Traducción Funcional para Lógicas Modales

Autor: Marcio Díaz

Director: Dr. Carlos Areces

9 de Agosto de 2013

Resumen

En esta tesis estudiamos las traducciones funcionales que transforman fórmulas de lógica modal a lógica de primer orden con sorts. En trabajos previos se mostró que remover las anotaciones de sorts de las traducciones funcionales preserva satisfacibilidad. Investigamos el desempeño de SPASS, un demostrador de lógica de primer orden, al suministrarle fórmulas traducidas con y sin anotaciones de sorts. Concluimos que, para todas los casos testeados, remover sorts mejora el desempeño del demostrador.

Clasificación: F.4.1 Mathematical Logic.

Palabras claves: lógica modal, lógica híbrida, traducciones funcionales, lógica de primer orden, demostrador de teoremas, SPASS.

Agradecimientos

Agradezco a mi familia, que me apoyo durante toda la carrera.
A Carlos Areces por su dirección en este trabajo y por ayudarme en todo lo posible.
A los profesores por su excelencia en la enseñanza y, en general, a toda la comunidad de FaMAF por la increible ayuda que le brindan a sus alumnos.

Marcio

Índice general

Capítulo 1

Introducción

1.1. Motivación

En esta tesis estudiaremos traducciones funcionales que transforman fórmulas de la lógica híbrida multimoda $\mathcal{H}(@,\downarrow)$ en fórmulas de la lógica de primer orden (FOL) con sorts.

En [Areces and Gorín, 2011] se mostró que bajo ciertas condiciones es posible remover los sorts de la fórmulas traducidas preservando satisfacibilidad. Cuando el objetivo es usar un demostrador para FOL se conjetura que trabajar sobre la traducción obtenida al eliminar sorts deberia redundar en una aceleración del tiempo de demostración. El propósito de este trabajo es investigar dicha conjetura, comparando y analizando el desempeño de SPASS, uno de los principales demostradores de teoremas para FOL, utilizando un gran número de fórmulas modales traducidas funcionalmente.

1.2. Estructura del documento

En los capítulos 2 y 3 introducimos conceptos teóricos. En el primero hablamos sobre lógicas modales e híbridas; en el segundo, definimos traducciones funcionales y enunciamos teoremas que aseguran preservación de satisfacibilidad al remover sorts.

En el capítulo 4 mostramos, usando pseudocódigo, las implementaciones de las traducciones funcionales y otras transformaciones necesarias.

En los capítulos 5, 6 y 7 evaluamos el desempeño de las traducciones funcionales sobre varias lógicas utilizando distintos métodos de benchmark. Primero, en el capítulo 5, utilizamos fórmulas creadas manualmente para evaluar las traducciones sobre la lógica modal **K**. Luego, en el capítulo 6, usamos un generador de fórmulas aleatorias para evaluar las traducciones en las lógicas híbridas $\mathcal{H}$, $\mathcal{H}(@)$, y $\mathcal{H}(\downarrow)$. Finalmente, en el capítulo 7, evaluamos las traducciones imponiendo condiciones sobre las relaciones de los modelos: reflexividad y transitividad, obteniendo las lógicas **K4** y **KT** respectivamente.

Capítulo 2

Las lógicas modales

2.1. 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 encontrar 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 analizó la diferencia que hay entre implicaciones de la forma $A \to B$, 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 la Luna esta hecha de queso, entonces Elvis no murió". Intuitivamente, la proposición expresada por esta oración es falsa. Sin embargo, al traducirla al lenguaje de la lógica clásica, obtenemos:

$$\text{La Luna esta hecha de queso} \to \text{Elvis no murió}$$

Esta proposición es verdadera, puesto que el antecedente es falso. Pero esto contradice nuestra intuición de que la proposición es falsa. Por lo tanto, la fórmula $A \to B$ no es una traducción satisfactoria de la oración original. Para resolver este problema, en sus trabajos Lewis extiende la lógica proposicional agregando el *operador modal* $\Box$, que se interpreta como "es necesario que" y con el que define la *implicación estricta*:

$$\varphi \mapsto \psi \equiv \Box(\varphi \to \psi)$$

Usando el condicional estricto, nuestra oración queda traducida como:

$$\Box(\text{La Luna esta hecha de queso} \to \text{Elvis no murió})$$

En lógica modal, esta proposición significa (aproximadamente) que en todo mundo posible donde la Luna esta hecha de queso, Elvis no murió. Dado que es posible imaginar un mundo donde la Luna esta hecha de queso, y Elvis haya muerto, esta fórmula es falsa. Por lo tanto esta parece ser una traducción más correcta que la original.

Existe una relación muy fuerte entre necesidad y posibilidad. La "posibilidad" es una modalidad que suele denotarse con el operador modal $\Diamond$. Decir "es necesario que suceda φ" es intuitivamente equivalente a decir "no es posible que no suceda φ", en símbolos:

$$\Box\varphi \equiv \neg\Diamond(\neg\varphi)$$

Es decir, $\Diamond$ es el operador dual de $\Box$ (y viceversa).

Interpretar informalmente una fórmula modal como hicimos en el ejemplo anterior no es complicado. De hecho, podríamos haberle asignado a $\Diamond$ un modo cualquiera, por ejemplo *deseo*, e interpretar $\Diamond$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, respectivamente. En este lenguaje podemos, por ejemplo, escribir la frase "siempre que llovió paró o parará" como $H(\mathsf{llueve} \rightarrow F(\neg\mathsf{llueve}))$. La fórmula $\mathsf{llueve} \rightarrow F(\neg\mathsf{llueve})$ se interpreta como "si llueve (en este momento), entonces en algún momento del futuro no lloverá"; al precederla con el operador P 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 denotar a todas las lógicas (multi-) modales.

Definición 2.1. 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:

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

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] Como veremos en la Sección 2.2, una característica de los operadores modales es que su semántica está definida en términos de relaciones de accesibilidad.

2.2. 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 2.2 (Modelo de Kripke)**.** Un modelo de Kripke es una estructura $M = \langle W, \{R_i\}, V\rangle$ donde

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

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 $H(\mathsf{llueve} \to F(\neg\mathsf{llueve}))$ 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, a 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 $\mathsf{llueve} \to F(\neg\mathsf{llueve})$. Para cada x, $M, x \models \mathsf{llueve} \to F(\neg\mathsf{llueve})$ 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.

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 H tuvimos que buscar en *todos los instantes anteriores* a 0, mientras que para tratar F tuvimos que buscar *algún instante posterior* a cada x.

Definición 2.3 (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:

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

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 exactamente 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.

2.3. 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 uni-modales hasta lógicas con infinitas modalidades (e.g., Propositional Dynamic Logic [Harel *et al.*, 1984]). 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 2.2 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.

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 2.4 (Traducción estándar)**.** La traducción estándar conmuta con los operadores booleanos y satisface:

$$ST_j(p) \stackrel{def}{=} p(j)$$
$$ST_j(\langle r\rangle\varphi) \stackrel{def}{=} \exists k.(r(j,k) \wedge ST_k(\varphi)) \qquad (k \text{ es nueva})$$

donde $p \in \mathsf{PROP}$ y $r \in \mathsf{REL}$.

La traducción estándar es uno de los resultados de reciente interés para 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* (*KB*) y una serie de *servicios de razonamiento* (*reasoning services*). A 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"

[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.

- "Juan es un vendedor de bronceador"
- "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", dada una KB que capture la información que enumeramos. También es común que puedan responder consultas enumerativas como "¿quiénes son todos los filósofos mencionados en la KB?". 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.

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.

2.4. 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 multimodal en la que se imponen ciertas restricciones sobre las relaciones de accesibilidad (e.g., transitividad, no reflexividad, anti-simetría). 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 2.5. 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 $\mathsf{ATOM} = \mathsf{PROP} \cup \mathsf{NOM}$ 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 $a \in \mathsf{ATOM}$, $r \in \mathsf{REL}$ 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 2.6 (Modelo híbrido)**.** Un modelo híbrido es una estructura $M = \langle W, \{R_i\}, V \rangle$ donde

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

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 2.3, con el agregado, obviamente, del caso:

$$M, w \models n_i \text{ sii } n_i \in V(w), n_i \in \mathsf{NOM}. \tag{2.1}$$

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 " estoy en Grecia y llueve" diciendo: $\mathsf{grecia} \wedge \mathsf{llueve}$, 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 punto denotado por grecia, y sólo si en él vale la proposición llueve.

En las lógicas modales tradicionales, para evaluar una fórmula, nos situamos en un punto cualquiera del modelo e intentamos recorrer otros puntos relacionados, de acuerdo a las distintas modalidades que aparezcan. En el ejemplo anterior hacemos lo mismo. Pero, al ser grecia un nominal, sólo tiene sentido comenzar a evaluar el resto de la fórmula, desde el punto denotado

por él. 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"? Podemos vernos tentados de usar la fórmula anterior: $\mathsf{grecia} \wedge \mathsf{llueve}$. Sin embargo, esta fórmula expresa algo extra: que estoy en Grecia. El problema aparece cuando queremos hablar de un punto, en el cuál: no estoy posicionado y no se relaciona con mi posición.

$\mathcal{H}$ 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 $\Diamond\mathsf{llueve}$ debe serlo en el mundo al que china hace referencia.

Definición 2.7. 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 $\mathsf{ATOM} = \mathsf{PROP} \cup \mathsf{NOM}$ 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 $a \in \mathsf{ATOM}$, $r \in \mathsf{REL}$, $i \in \mathsf{NOM}$ 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 2.8 (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:

$$M, w \models @_i\, \varphi \quad \text{sii} \quad M, w' \models \varphi \text{ donde } V(i) = \{w'\}$$

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 "estoy posicionado". 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.

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, $\mathsf{Sócrates}$ y $\mathsf{mañana}$ en nuestro ejemplo anterior).

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

¿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 $\mathcal{H}(@, \downarrow)$).

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

Capítulo 3

Traducciones

3.1. Introducción

La *traducción functional* es una herramienta para el razonamiento modal automatizado que apareció independientemente y casi simultáneamente en varias publicaciones a fines de la década de 1980 y principios de los 90's (por ej. [Ohlbach, 1988a; Ohlbach, 1988b; Fariñas del Cerro and Herzig, 1988; Zamov, 1989; Auffray and Enjalbert, 1989; Auffray and Enjalbert, 1992]). Esta traducción transforma fórmulas de lenguajes modales a lógica de primer orden, preservando satisfacibilidad, tal como lo hace la *traducción estándar* (ver 2.4). Pero la traducción funcional usa una alternativa semántica a las estructuras relacionales que produce fórmulas más compactas y con una estructura de términos menos profunda que aquella obtenida por la traducción estándar [Schmidt, 1997; Horrocks *et al.*, 2006]. Estas dos propiedades son de suma importancia cuando se intenta usar razonamiento automatizado de primer orden basado en resolución. Más aun, y a diferencia de otras traducciones aptas para el razonamiento automatizado (por ej. [Areces *et al.*, 2000]), la traducción functional puede ser usada para razonar sobre un amplio rango de clases de modelos, además de la clase de todos los modelos.

La traducción funcional es muchas veces introducida usando un lenguaje de primer orden con sorts, para simplificar la presentación. Esto significa, en la práctica, que necesitamos usar un demostrador de teoremas que maneje lógica de primer orden con múltiples sorts o simular sorts en lógica de primer orden sin sorts, introduciendo predicados adicionales de aridad uno. Ambas alternativas pueden tener impacto en el desempeño de la prueba que se intenta llevar a cabo.

En [Walther, 1989] se argumenta, por ejemplo, que la simulación de sorts mediante símbolos proposicionales conduce a inferencias irrelevantes. Ciertos demostradores automáticos de primer orden como SPASS([Weidenbach *et al.*, 2007]), evitan estas inferencias pero, por otro lado, la complejidad adicional de la maquinaria necesaria para manejar inferencias de sorts (en el caso de SPASS, *well-sorted unification* [Weidenbach, 2001]) necesita ser tomada en cuenta.

En [Hustadt and Schmidt, 1999], para lógica modal básica, y en [Areces and Gorín, 2011] para lógica híbrida, se mostró que en ciertos casos se puede borrar las anotaciones de sorts sin cambiar el estado de satisfacibilidad de la fórmula.

Uno de los principales temas de esta tesis es verificar empíricamente si los sorts ayudan o dificultan la tarea del demostrador de teoremas.

3.2. Traducción Funcional

Durante el resto de la teiss trabajaremos en el lenguaje híbrido multi-modal $\mathcal{H}(@,\downarrow)$.

Para una signatura fija, que consiste en: un conjunto de símbolos proposicionales PROP, un conjunto de nominales NOM y un conjunto de símbolos relacionales REL; todos disjuntos de a

pares, sus fórmulas estan dadas por:

$$\varphi ::= p \mid i \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle r\rangle\varphi \mid @_i\varphi \mid \downarrow i.\varphi$$

donde $p \in \mathsf{PROP}$, $i \in \mathsf{NOM}$ y $r \in \mathsf{REL}$. Para la semántica tomamos como modelos pares $\langle \mathcal{I}, g\rangle$, donde $\mathcal{I} = \langle W, \cdot^{\mathcal{I}}\rangle$ es una interpretación relacional tal que $p^{\mathcal{I}} \subseteq W$ para $p \in \mathsf{PROP}$; y $r^{\mathcal{I}} \subseteq W \times W$, para $r \in \mathsf{REL}$. Mientras que $g : \mathsf{NOM} \to W$ es una asignación para los nominales.

Ahora podemos dar significado a las fórmulas de $\mathcal{H}(@, \downarrow)$, mediante la traducción estándar a lógica de primer orden. A la definición de la traducción estándar de 2.4, agregamos:

$$ST_j(i) \stackrel{def}{=} i = j \qquad ST_j(@_i\varphi) \stackrel{def}{=} ST_i(\varphi)$$
$$ST_j(\downarrow i.\varphi) \stackrel{def}{=} \exists i.(i = j \wedge ST_j(\varphi))$$

Entonces, para $w \in W$ y $j \in \mathsf{NOM}$ que no ocurre en φ, tenemos:

$$\mathcal{I}, g, w \models \varphi \iff \mathcal{I} \models_{\mathrm{FO}} ST_j(\varphi)[g^j_w]. \tag{3.1}$$

donde g^j_w es igual a g para todo nominal distinto de j y $g(j) = w$.

Para la clase de modelos $\mathcal{C}$, φ es *$\mathcal{C}$-satisfacible* (*$\mathcal{C}$-valida*, notación $\mathcal{C} \models \varphi$) si para algún modelo (para todos los modelos) $\langle \mathcal{I}, g\rangle$ en $\mathcal{C}$: $\mathcal{I}, g, w \models \varphi$ para algún w (para todos w). Si $\mathcal{C}$ es la clase de todos los modelos, decimos que φ es *satisfacible* (*valido*, notación $\models \varphi$). En lógica modal, estamos usualmente interesados en clases definidas como aquellos modelos que satisfacen ciertas condiciones (por ejemplo., transitividad). Cualquier clase $\mathcal{C}$ de esas se dice que esta definida por una fórmula φ siempre que $\langle \mathcal{I}, g\rangle$ este en $\mathcal{C}$ sii $\mathcal{I}, g, w \models \varphi$ para todo w. Ver [Blackburn *et al.*, 2002] para más detalles.

Podemos ver a la traducción estándar como una codificación en lógica de primer orden de las claúsulas semánticas para los operadores modales. La traducción es bastante simple pero, en general, no es apta para razonamiento automatizado. Es fácil encontrar fórmulas modales simples que, al ser traducidas a lógica de primer orden usando ST y luego resueltas via resolución, resultan en conjuntos de claúsulas infinitas (ver [Areces *et al.*, 2000]). Esta es la principal motivación para el uso de la, posiblemente más compleja, traducción funcional que describiremos a continuación.

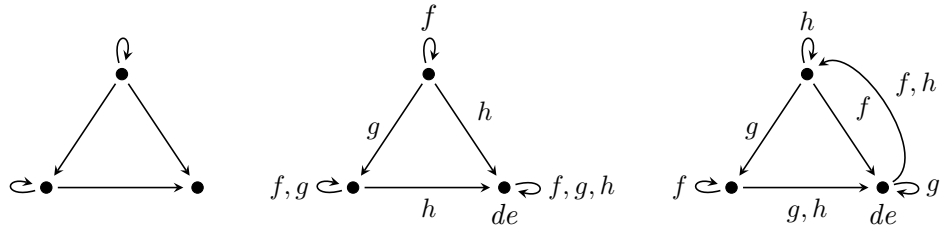
La clave para entender la traducción funcional es una representación alternativa de las estructuras relacionales. Asumamos por un momento que $\mathsf{REL} = \{r\}$. Consideremos entonces la Figura 3.1a que muestra una estructura relacional cuyo dominio consiste en tres elementos. Esta estructura puede representarse de manera alternativa usando tres funciones totales f, g y h, y un predicado de, siempre que se mantenga la siguiente propiedad:

$$\forall xy.\big(r(x, y) \leftrightarrow (\neg de(x) \wedge (f(x) = y \vee g(x) = y \vee h(x) = y))\big). \tag{3.2}$$

Usamos de (por "dead end") para "marcar" aquellos estados que no tienen un r-sucesor, y f, g y h en cada estado para "atestiguar" cada r-sucesor. Hay muchos arreglos válidos para f, g y h; las Figuras 3.1b y 3.1c muestran dos de ellos. Es sencillo verificar que ambos satisfacen la condición (3.2).

Necesitamos tres funciones por que uno de los estados del modelo tiene tres sucesores. En general, necesitamos tantas funciones como máxima ramificación de salida (fan-out) pueda tener un estado del modelo. Esto complica la representación. Alternativamente, podemos usar una función binaria f, que tome un índice como argumento inicial. Esto se puede expresar fácilmente en un lenguaje con dos sorts ω y ι, el primero se referirá a los nodos del modelo propiamente, y el segundo a los indices de las funciones.

Un modelo funcional será entonces una estructura $\mathcal{I} = \langle W, I, \cdot^{\mathcal{I}}\rangle$ donde W y I son dominios no vacios para los sorts ω y ι, respectivamente; $p^{\mathcal{I}} \subseteq W$ para cada $p \in \mathsf{PROP}$; y, para cada $r \in \mathsf{REL}$, $f_r : I \times W \to W$ y $de_r \subseteq W$.



(a) (b) (c)

Figura 3.1: El modelos relacional (a) es expresado en (b) y (c) con las funciones f, g, h y el predicado de.

Claramente, todo modelo funcional induce un modelo relacional, tal que para toda relación r vale lo siguiente:

$$\forall x, y{:}\omega.r(x, y) \leftrightarrow (\neg de_r(x) \wedge \exists z{:}\iota.f(z, x) = y) \tag{3.3}$$

En [Areces and Gorín, 2011] se vio que una fórmula de la lógica modal básica es satisfacible si y sólo si existe un modelo funcional que la satisface. Luego, podemos ver a la *traducción funcional* simplemente como una traducción alternativa a la traducción estándar que mapea fórmulas modales a fórmulas de la lógica de primer orden con multi-sorts.

Definición 3.1 (Traducción Funcional)**.** Sea j una variable del tipo ω que ocurre en el término t del lenguaje funcional. La traducción funcional FT_t mapea fórmulas de $\mathcal{H}(@, \downarrow)$ en fórmulas de lógica de primer orden en el lenguaje funcional con una variable libre j del siguiente modo:

$$\begin{aligned}
FT_t(p) &\stackrel{def}{=} p(t) \\
FT_t(\downarrow i.\varphi) &\stackrel{def}{=} \exists i{:}\omega.(i = t \wedge FT_t(\varphi)) \\
FT_t([r]\varphi) &\stackrel{def}{=} \neg de_r(t) \longrightarrow \forall z{:}\iota.FT_{f_r(z,t)}(\varphi) \\
FT_t(i) &\stackrel{def}{=} i = t \\
FT_t(@_i\varphi) &\stackrel{def}{=} FT_i(\varphi)
\end{aligned}$$

(z es una variable nueva)

Teorema 1. *Sea φ una fórmula de $\mathcal{H}(@, \downarrow)$ y sea i un nominal que no ocurre en φ. Entonces se tiene que:*

(I) $\models \varphi$ *sii* $\models_{FO} \forall i{:}\omega.FT_i(\varphi)$

(II) φ *es satisfacible sii* $\exists i{:}\omega.FT_i(\varphi)$ *es satisfacible*

Consideremos ahora la siguiente fórmula modal:

$$[r](p \rightarrow \langle r \rangle p). \tag{3.4}$$

Por el Teorema 1, esta fórmula es satisfacible si y sólo si lo es su traducción funcional:

$$\exists i{:}\omega.(\neg de(i) \rightarrow \forall y{:}\iota.(p(f(y, i)) \rightarrow (\neg de(f(y, i)) \wedge \exists z{:}\iota.p(f(z, f(y, i)))))). \tag{3.5}$$

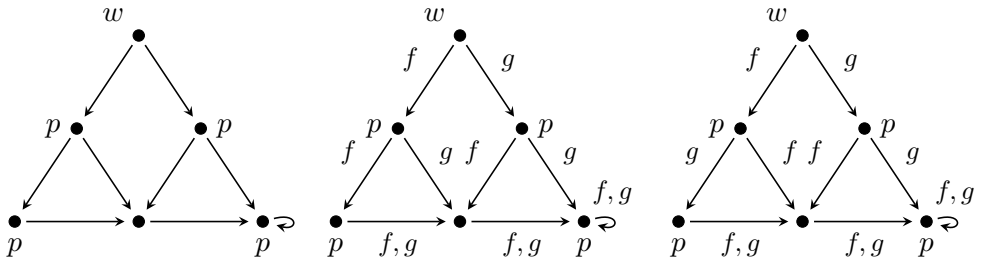


Figura 3.2: Un modelo (a) para la fórmula (3.4) y dos modelos (b) y (c) para su traducción funcional

Skolemizando i y z obtenemos la fórmula equisatisfacible:

$$\neg de(c) \rightarrow \forall y{:}\iota.(p(f(y,c)) \rightarrow (\neg de(f(y,c)) \wedge p(f(g(y),f(y,c))))). \tag{3.6}$$

Esta fórmula contiene dos símbolos de Skolem: una constante c y una función unaria g. La traducción funcional optimizada [Ohlbach and Schmidt, 1997] garantiza que sólo es necesario introducir constantes durante la skolemización. Debido a que las funciones de Skolem pueden causar que se construyan términos complejos durante la resolución la traducción funcional optimizada puede reducir drásticamente el proceso de saturación. Más aun, esta simplifica el desarrollo de estrategias de resolución que garantizan terminación, [Schmidt, 1999].

Para ilustrar la idea detrás de la traducción optimizada consideremos de nuevo la fórmula (3.4). La Figura 3.2a muestra un modelo que satisface (3.4) en el nodo w. La Figura 3.2b, por otro lado, muestra un modelo funcional para (3.6). Es fácil verificar que este modelo induce el de la Figura 3.2a

Observamos ahora que si i es interpretado como w (notación: $i \mapsto w$) hay dos posibles valores para y, f y g. Si $y \mapsto f$, entonces debemos tomar $z \mapsto f$, mientras que para $y \mapsto g$ debemos seleccionar $z \mapsto g$. Entonces el valor correcto para z es efectivamente una función de y, como lo atestigua la skolemización. Pero ahora viene la parte interesante, podemos "reacomodar" la asignación de funciones de modo que la elección de z sea independiente de y. Un ejemplo de esto es la Figura 3.2c; este modelo también induce (a) pero la elección correcta es $z \mapsto g$ independientemente del valor de y. En modelos maximales, donde todas los posibles formas de colocar las funciones son incluídas, es siempre posible hacer la intrepretación de cada variable independiente de las otras.

Ohlbach y Schmidt [Ohlbach and Schmidt, 1997] aprovecharon esta observación para probar que es correcto, en términos de satisfacibilidad, intercambiar dos cuantificadores consecutivos. Por lo tanto, se puede tomar una fórmula obtenida usando la traducción funcional básica, y mover todos los cuantificadores existenciales al principio, efectivamente evitando la introducción de funciones de Skolem. Esto es exactamente lo que hace la *traducción funcional optimizada*.

Definición 3.2. La traducción funcional optimizada a lógica de primer orden (OFT) se define como $OFT_j(\varphi) \stackrel{def}{=} \vartheta(FT_j(\varphi))$ donde $\vartheta(\gamma)$ transforma γ a forma normal prenexa y mueve todos los cuantificadores de tipo ι al frente.

Teorema 2. *Sea φ una fórmula de $\mathcal{H}(@,\downarrow)$ y sea i un nominal que no ocurre en φ. Entonces se tiene que:*

(I) $\models \varphi$ *sii* $\models_{FO} \forall i{:}\omega.OFT_i(\varphi)$

(II) φ *es satisfacible sii* $\exists i{:}\omega.OFT_i(\varphi)$ *es satisfacible*

3.3. Traducción Funcional sin Sorts

Como dijimos anteriormente, existen demostradores de teoremas que manejan sorts y otros que no, aunque en éste último caso siempre pueden simularse usando predicados unarios. También comentamos que, en ciertos casos, se preserva satisfacibilidad al remover las anotaciones de sorts. En los capítulos siguientes analizaremos y presentaremos algunos resultados concernientes a la eficiencia de ambos tipos de traducciones, con y sin anotaciones de sorts. Formalizaremos primero el concepto de "remover sorts" y presentaremos los resultados que aseguran la correctitud del procedimiento.

Definición 3.3 (Remover sorts)**.** La transformación $(.)^-$ toma una fórmula en lógica de primer orden con sorts y remueve los sorts de la siguiente manera:

$$\begin{aligned} a^- &\stackrel{def}{=} a \text{ , para } a \text{ una fórmula atómica} \\ (\neg a)^- &\stackrel{def}{=} \neg(a^-) \\ (\varphi \wedge \psi)^- &\stackrel{def}{=} \varphi^- \wedge \psi^- \\ (\forall i{:}\alpha.\varphi)^- &\stackrel{def}{=} \forall i.\varphi^- \\ (\exists i{:}\alpha.\varphi)^- &\stackrel{def}{=} \exists i.\varphi^- \end{aligned}$$

Un *modelo funcional sin sorts* es un modelo de la signatura resultante.

Para probar que remover sorts preserva satisfacibilidad, se necesita observar que cada fórmula satisfacible traducida funcionalmente es satisfecha por un modelo en cual las cardinalidades de W e I son iguales. Consideremos de nuevo el modelo de la Figura 3.1a. Este es representando en las Figuras 3.1b y 3.1c usando sólo tres funciones, pero puede ser representado por cualquier cantidad de funciones, ya que no nos importan las funciones duplicadas.

Por otro lado, dado que uno de los nodos de este modelo se relaciona con tres nodos diferentes, no puede ser representado con menos de tres funciones. Como la máxima cantidad de nodos con los que se puede relacionar un nodo cualquiera es W arribamos al siguiente teorema:

Teorema 3. *Sea φ una fórmula de $\mathcal{H}(@,\downarrow)$. Son equivalentes:*

(I) φ *es satisfacible*

(II) $\exists i{:}\omega.FT_i(\varphi)$ *es satisfacible*

(III) $\exists i.FT_i(\varphi)^-$ *es satisfacible*

El teorema análogo para la traducción optimizada necesita utilizar la unión disjunta de W^{W} copias de un modelo para garantizar que podemos elevar la cardinalidad de W cuando lo necesitemos.

Teorema 4. *Sea φ una fórmula de $\mathcal{H}(@,\downarrow)$. Son equivalentes:*

(I) φ *es satisfacible*

(II) $\exists i{:}\omega.OFT_i(\varphi)$ *es satisfacible*

(III) $\exists i.OFT_i(\varphi)^-$ *es satisfacible*

También se preserva satisfacibilidad al remover sorts para clases de frames definibles modalmente, como lo dice el Teorema 5. Aunque en realidad, la demostración sólo utiliza el hecho de que las clases de frames definibles modalmente son invariantes bajo unión disjunta (Teorema de Goldblatt y Thomason, ver [Blackburn *et al.*, 2001] o cualquier introducción a lógica modal).

Teorema 5. *Sea C una clase de modelos que puede definirse modalmente y sea φ una fórmula de $\mathcal{H}(@,\downarrow)$. Son equivalentes:*

(I) *φ es C-satisfacible.*

(II) *$\exists i{:}\omega.OFT_i(\varphi)$ es C-satisfacible.*

(III) *$\exists i.OFT_i(\varphi)^-$ es C-satisfacible.*

Capítulo 4

Implementación

Para implementar las traducciones utilizamos *haskell*, un lenguaje de programación funcional. Elegimos haskell por qué ya teniamos parte del parser de fórmulas híbridas implementado en él. Además haskell soporta *pattern matching*, el cual hace la tarea de traducir fórmulas mucho más simple.

El Algoritmo 1 transforma fórmulas de $\mathcal{H}(@,\downarrow)$ a forma normal negada (NNF), moviendo las negaciones hasta que sólo afecten símbolos proposicionales. Esta transformación se aplica a todas las fórmulas híbridas, y es necesaria para que las traducciones funcionales trabajen correctamente.

Algoritmo 1. ***Forma normal negada***

```
(1)  funct nnf(f) ≡
(2)    ⌈if f = ¬⟨r⟩(g) → [r](nnf(¬g))
(3)     □ f = ¬[r](g) → ⟨r⟩(nnf(¬g))
(4)     □ f = ¬¬(g) → nnf(g)
(5)     □ f = ¬@_n(g) → @_n(nnf(¬g))
(6)     □ f = ¬↓_n(g) → ↓_n(nnf(¬g))
(7)     □ f = ¬(g ∧ h) → nnf(¬g) ∨ nnf(¬h)
(8)     □ f = ¬(g ∨ h) → nnf(¬g) ∧ nnf(¬h)
(9)     □ f = ⟨r⟩(g) → ⟨r⟩(nnf(g))
(10)    □ f = [r](g) → [r](nnf(g))
(11)    □ f = ¬(g) → ¬ nnf(g)
(12)    □ f = @_n(g) → @_n(nnf(g)
(13)    □ f = ↓_n(g) → ↓_n(nnf(g))
(14)    □ f = g ∧ h → nnf(g) ∧ nnf(h)
(15)    □ f = g ∨ h → nnf(g) ∨ nnf(h)
(16)    □ true → f
(17)    fi⌋.
```

El Algoritmo 2 muestra pseudocódigo de la traducción funcional. En la linea 3 transformamos previamente la fórmula híbrida a NNF. La función $traduccion\_funcional'$ es muy similar a su definición (3.1), salvó algunas funciones auxiliares para el manejo de nuevas variables ($incrementar\_variable$ y $numero\_de\_cuantificadores$).

Algoritmo 2. ***Traducción funcional***

```
(1)  begin
(2)    funct traduccion_funcional(f) ≡
(3)      ⌈g := negation_normal_form(f);
(4)       r := traduccion_funcional'(x1, 1, g);
```

```
(5)        r ⌋.
(6)  where
(7)  funct traduccion_funcional'(t, x, f) ≡
(8)    ⌈if f = Nominal n → t = n
(9)     □ f = VarEstado sv → t = sv
(10)    □ f = Prop P → P(t)
(11)    □ f = [r](g) → y := incrementar_variable(x);
(12)                    h := traduccion_funcional'(R(y, t), y, g);
(13)                    ¬ de_r(t) → (∀y:ι.h)
(14)    □ f = ⟨r⟩(g) → y := incrementar_variable(x);
(15)                    h := traduccion_funcional'(R(y, t), y, g);
(16)                    ¬ de_r(t) ∧ (∃y:ι.h)
(17)    □ f = @_n(g) → traduccion_funcional'(n, x, f)
(18)    □ f = ↓_n(g) → h := traduccion_funcional'(t, x, g);
(19)                    ∃n:ω.(t = n) ∧ h
(20)    □ f = ¬g → ¬ traduccion_funcional'(t, x, g)
(21)    □ f = g ∧ h → g' := traduccion_funcional'(t, x, g);
(22)                   y := numero_de_cuantificadores(g);
(23)                   h' := traduccion_funcional'(t, y, h);
(24)                   g' ∧ h'
(25)    □ f = g ∨ h → g' := traduccion_funcional'(t, x, g);
(26)                   y := numero_de_cuantificadores(g);
(27)                   h' := traduccion_funcional'(t, y, h);
(28)                   g' ∨ h'
(29)    □ true → f
(30)    fi ⌋.
(31) end
```

Para remover sorts usamos el Algoritmo 3. Éste es muy similar a la definición (3.3), pero en el pseudocódigo se puede observar que utilizamos dos lenguajes diferentes, diferenciados por $'$, uno para FOL con multi-sorts y el otro para FOL sin sorts.

Luego, el Algoritmo 4 de la traducción funcional sin sorts, consiste simplemente en traducir funcionalmente la fórmula híbrida usando el Algoritmo 2 y luego removerle las anotaciones de sorts.

Algoritmo 3. ***Remover sorts***

```
(1)  funct remover_sorts(f) ≡
(2)    ⌈if f = s = t → s ='  t
(3)     □ f = Prop P → Prop' P
(4)     □ f = Rel s t → Rel' s t
(5)     □ f = ¬(g) → ¬' remover_sorts(g)
(6)     □ f = g ∧ h → remover_sorts(g) ∧' remover_sorts(h)
(7)     □ f = g ∨ h → remover_sorts(g) ∨' remover_sorts(h)
(8)     □ f = ∃n:ω.(g) → ∃'n.(remover_sorts(g))
(9)     □ f = ∀n:ω.(g) → ∀'n.(remover_sorts(g))
(10)    fi ⌋.
```

Algoritmo 4. ***Traducción funcional sin sorts***

```
(1)  funct traduccion_funcional_sin_sorts(f) ≡
(2)    ⌈g := traduccion_funcional(f);
(3)     remover_sorts(g) ⌋.
```

Algoritmo 5. ***Traducción funcional optimizada***

```
(1)  begin
(2)    funct traduccion_funcional_optimizada(f) ≡
(3)      ⌈g := traduccion_funcional(f);
(4)       h := forma_normal_prenexa(g);
(5)       mover_existenciales_afuera(h)⌋.
(6)  where
(7)  funct forma_normal_prenexa(f) ≡
(8)    ⌈if f = g ∧ h → g' := forma_normal_prenexa(g);
(9)                    h' := forma_normal_prenexa(h);
(10)                   mezclar(g' ∧ h')
(11)    □ f = g ∨ h → g' := forma_normal_prenexa(g);
(12)                   h' := forma_normal_prenexa(h);
(13)                   mezclar(g' ∨ h')
(14)    □ f = ∃i:s.g → ∃i:s.forma_normal_prenexa(g)
(15)    □ f = ∀i:s.g → ∀i:s.forma_normal_prenexa(g)
(16)    □ true → f
(17)    fi⌋.
(18) funct mezclar(f) ≡
(19)   ⌈if f = (∀i:s.g) ∧ h → ∀i:s.mezclar(g ∧ h)
(20)    □ f = (∃i:s.g) ∧ h → ∃i:s.mezclar(g ∧ h)
(21)    □ f = (∀i:s.g) ∨ h → ∀i:s.mezclar(g ∨ h)
(22)    □ f = (∃i:s.g) ∨ h → ∃i:s.mezclar(g ∨ h)
(23)    □ f = h ∧ (∀i:s.g) → ∀i:s.mezclar(h ∧ g)
(24)    □ f = h ∧ (∃i:s.g) → ∃i:s.mezclar(h ∧ g)
(25)    □ f = h ∨ (∀i:s.g) → ∀i:s.mezclar(h ∨ g)
(26)    □ f = h ∨ (∃i:s.g) → ∃i:s.mezclar(h ∨ g)
(27)    □ true → f
(28)    fi⌋.
(29) funct mover_existenciales_afuera(f) ≡
(30)   ⌈f' := borrar_existenciales(f);
(31)    existenciales := guardar_existenciales(f);
(32)    poner_existenciales(existenciales, f')⌋.
(33) funct borrar_existenciales(f) ≡
(34)   ⌈if f = ∃i:s.g → if s = ι then borrar_existenciales(g)
(35)                              else ∃i:s.borrar_existenciales(g)
(36)                    fi
(37)    □ f = ∀i:s.g → ∀i:s.borrar_existenciales(g)
(38)    □ true → f
(39)    fi⌋.
(40) funct guardar_existenciales(f) ≡
(41)   ⌈if f = ∃i:s.g → if s = ι then s : guardar_existenciales(g)
(42)                              else guardar_existenciales(g)
(43)                    fi
(44)    □ f = ∀i:s.g → guardar_existenciales(g)
(45)    □ true → lista_vacia fi⌋.
(46) funct poner_existenciales(f, existenciales) ≡
(47)   ⌈if existenciales = lista_vacia → f
(48)    □ existenciales = e : existenciales' →
(49)      poner_existenciales(∃e:ι.f, existenciales') fi⌋.
(50) end
```

El Algoritmo 5 muestra pseudocódigo de la traducción funcional optimizada. Éste consta de tres etapas secuenciales: traducir la fórmula híbrida a FOL con multi-sorts, transformar la fórmula resultante a forma normal prenexa y, por último, mover los existenciales que cuantifican variables del tipo ι (índices) al principio de la fórmula.

En la línea 7 se observa la función $forma\_normal\_prenexa$ que se encarga de mover todos los cuantificadores de la fórmula al principio de la misma. Para llevar a cabo esta tarea, primero se llama recursivamente en las subfórmulas inmediatas. Luego, utiliza la función auxiliar *mezclar* (linea 18) para mover los cuantificadores de cada subfórmula hacia afuera.

En la linea 29 se observa la función $mover\_existenciales\_afuera$, encargada de mover los cuantificadores existenciales, que cuantifican variables índices, al principio de la fórmula. Consta de tres funciones auxiliares: $borrar\_existenciales$, $guardar\_existenciales$ y $poner\_existenciales$.

La función $borrar\_existenciales$ devuelve como resultado una fórmula f' producto de removerle a su argumento los cuantificadores existencialesde variables índices.

La función $guardar\_existenciales$ retorna la lista de variables índices borrados en el paso anterior.

Finalmente, $poner\_existenciales$ es la función que junta las piezas, colocando las variables índice al principio de la fórmula f', cuantificadas existencialmente.

Por último, en el Algoritmo 6 se observa el pseudocódigo para la traducción funcional optimizada sin sorts que simplemente le remueve las anotaciones de sorts al resultado de la traducción funcional optimizada.

Algoritmo 6. ***Traducción funcional optimizada sin sorts***

```
(1) funct traduccion_optimizada_sin_sorts(f) ≡
(2)   ⌈g := traduccion_funcional_optimizada(f);
(3)    remover_sorts(g)⌋.
```

Capítulo 5

The Logics Workbench benchmark

5.1. Introducción

En este capítulo mostraremos los resultados obtenidos al utilizar el benchmark de The Logics Workbench (LWB, ver [Heuerding and Schwendimann, 1996]) sobre las distintas traducciones funcionales.

Dicho benchmark consiste en un conjunto de problemas, fórmulas, de las lógicas modales **K**, **KT** y **S4**. En esta tesis sólo utilizaremos las fórmulas del lenguaje modal básico **K**, para poner a prueba las distintas traducciones funcionales.

Las fórmulas del benchmark fueron seleccionadas siguiendo una serie de postulados, con el fin de asegurar la calidad de la comparación entre los distintos métodos de razonamiento. Estos postulados son los siguientes:

1. Debe haber fórmulas demostrables y no demostrables.
2. Fórmulas de diversas estructuras (profundidad modal, origen, número de variables).
3. Algunas fórmulas deben ser demasiado difíciles, incluso para los demostradores modernos.
4. Se debe conocer con anticipación el resultado para cada fórmula.
5. Los trucos simples no deben ayudar a resolverlas.
6. Aplicar el benchmark a los demostradores no debe llevar mucho tiempo.
7. Los resultados pueden ser sumarizados.

Satisfacer estos postulados implica el uso exclusivo de fórmulas escalables. Es decir, fórmulas capaces de hacerse arbitrariamente grandes y difíciles, sin perder sus características distintivas.

Coleccionar y construir dichas fórmulas, teniendo en cuenta los postulados, lleva mucho trabajo. Por este motivo el benchmark sólo consta de nueve clases de problemas, aunque cada uno con una estructura diferente. Esto constrasta con el enfoque elegido al usar generadores aleatorios de fórmulas, como *hGen*, en el cual el número de fórmulas no es un problema. Además hGen es un generador de fórmulas híbridas altamente parametrizado, por esta razón será usado para evaluar el desempeño de las traducciones en el capítulo siguiente.

En la sección 5.2 mostraremos algunas fórmulas del benchmark para obtener una idea de la dificultad a la que se enfrentarán los demostradores, y en la sección 5.3 presentaremos gráficos de los resultados obtenidos.

5.2. Fórmulas

Las fórmulas del benchmark estan divididas en nueve clases satisfacibles y nueve clases no satisfacibles. Cada fórmula esta parametrizada con un número $n \in [1, \ldots, 21]$, el cual indica la complejidad de la fórmula (mientras más grande es n, más compleja es la fórmula).

En la sección siguiente mostraremos los resultados que obtuvimos sólo para algunas de las clases de fórmulas del benchmark, por que son muy similares y, en varios casos, el problema no fue suficientemente difícil como para obtener conclusiones significativas. Por este motivo, en esta sección, sólo presentaremos las clases *poly*, *branch* y *t4p*.

Para cada fórmula listaremos las siguientes propiedades:

1. **Idea:** por qué la fórmula es satisfacible o no.
2. **Ocultamiento:** como está la fórmula escondida, para hacer el problema difícil de resolver.
3. **Características:** profundidad modal, número de variables.
4. **Definición** inductiva de la fórmula.

5.2.1. k_poly_p

Idea: la fórmula $(p_1 \leftrightarrow p_2) \vee (p_2 \leftrightarrow p_3) \vee \ldots \vee (p_{n-1} \leftrightarrow p_n) \vee (p_n \leftrightarrow p_1)$ codifica: si tenemos un poligono con n vértices, y todos los vértices son blancos o negros, entonces dos vértices adjacentes tienen el mismo color. Si n es impar, entonces esta fórmula es demostrable en lógica proposicional clásica CPC.

Ocultamiento: muchos $\Box$, $\Diamond$, y subfórmulas superfluas.

Características: esencialmente un problema de CPC. Tiene aproximadamente $4{,}5n$ variables y una profundidad modal de $3n$.

Definición:

$$k\_poly\_p(n) := \begin{cases} poly(3n+1) & n \bmod 2 = 0 \\ poly(3n) & n \bmod 2 = 1 \end{cases}$$

$$poly(n) := \Box^{n+1} \bigwedge_{i=1,\ldots,n+1}(p_i) \vee f(n,n) \vee \Box^{n+1} \bigwedge_{i=1,\ldots,n+1}(\neg p_{2i})$$

$$f(i,n) := \begin{cases} \text{false} & i = 0 \\ \Diamond(f(i-1,n) \vee \Diamond^i(p_n \leftrightarrow p_1)) \vee \Box p_{i+2} & i = n \\ \Diamond(f(i-1,n) \vee \Diamond^i(p_i \leftrightarrow p_{i+1})) \vee \Box p_{i+2} & \text{caso contrario} \end{cases}$$

5.2.2. k_poly_n

Idea: como en 5.2.1, pero para un número par de vértices.

Ocultamiento: muchos $\Box$, $\Diamond$, y subfórmulas superfluas. Las subfórmulas superfluas no influencian satisfacibilidad.

Características: esencialmente es un problema de CPC. Aproximadamente $4{,}5n$ variables, profundidad modal $3n$.

Definición:

$$k\_poly\_n(n) := \begin{cases} poly(3n) & n \bmod 2 = 0 \\ poly(3n+1) & n \bmod 2 = 1 \end{cases}$$

poly, f son como en 5.2.1

5.2.3. k_branch_n

Idea: p_{100+i} es cierta si estamos a profundidad $\geq i$ en el árbol.

bdepth intenta capturar la relación que tienen los p_{100+i}'s entre sí. Es decir, si estamos en una profundidad $\geq i$, entonces también estamos en una profundidad $\geq i-1$.

det establece una relación entre las proposiciones p_i y la profundidad i (p_{i+100}) en el árbol, de modo tal que si p_i es cierta (resp. falsa) en un nodo dado s en la profundidad j con $j \geq i$, entonces es cierta (resp. falsa) en todos los sucesores de s de profundidad al menos i.

Tomamos *branching* como la fórmula que intuitivamente dice que para todo nodo en la profundidad i, es posible encontrar dos nodos sucesores, a profundidad $i+1$ tal que p_{i+1} es cierta en uno y falsa en el otro.

Características: $2n+3$ variables, profundidad modal $n+1$.

Definición:

$$
\begin{aligned}
k\_branch\_n(n) &:= p_{100} \wedge \neg p_{101} \textstyle\bigwedge_{i=0,\ldots,n} (\Box^i (bdepth(n) \wedge det(n) \wedge branching(n))) \\
bdepth(n) &:= \textstyle\bigwedge_{i=1,\ldots,n+1} (p_{100+i} \rightarrow p_{99+i}) \\
det(n) &:= \textstyle\bigwedge_{i=0,\ldots,n} (p_{100+i} \rightarrow (p_i \rightarrow \Box(p_{100+i} \rightarrow p_i)) \wedge (\neg p_i \rightarrow \Box(p_{100+i} \rightarrow \neg p_i))) \\
branching(n) &:= \textstyle\bigwedge_{i=0,\ldots,n-1} (p_{100+i} \wedge \neg p_{101+i} \rightarrow \Diamond(p_{101+i} \wedge \neg p_{102+i} \wedge p_{i+1}) \\
&\quad \wedge \Diamond(p_{101+i} \wedge \neg p_{102+i} \wedge \neg p_{i+1}))
\end{aligned}
$$

Asumimos $n \leq 100$.

En [Halpern and Moses, 1992], se demostró que todo modelo de Kripke que satisface *k_branch_n* tiene al menos 2^n estados.

5.2.4. k_branch_p

Idea: la fórmula de ramificación como esta definida arriba, más la subfórmula $\neg\Box^n p_{n \text{ div } 3+1}$ para hacer la fórmula demostrable.

Características: $2n+3$ variables, profundidad modal $O(n)$.

Definición:

$$
\begin{aligned}
k\_branch\_p(n) &:= \neg(\neg(p_{100} \wedge \neg p_{101} \textstyle\bigwedge_{i=0,\ldots,n} (\Box^i (bdepth(n) \wedge det(n) \wedge branching(n)))) \\
&\quad \vee \neg\Box^n p_{n \text{ div } 3+1})
\end{aligned}
$$

5.2.5. k_t4p_n

Idea: $K \vdash T\{\Diamond p_0/p_0\} \wedge \Box T\{\neg\Box\Diamond p_0/p_0\} \wedge A4\{\Diamond p_0/p_0\} \wedge \Box(\Diamond\Box\Diamond p_0 \rightarrow (p_0 \rightarrow \Box p_0))$ $\rightarrow \Diamond\Box p_0 \vee \Diamond\Box\neg p_0$, donde: $T := \Box p_0 \rightarrow p_0$ y $A4 := \Box p_0 \rightarrow \Box\Box p_0$.

Ocultamiento: fórmulas superfluas ($\Diamond\neg p_3, \Diamond p_4$), instancias superfluas de A4 y A5, $\Box$ anidadas.

Características: 4 variables, profundidad modal $n+5$. Parcialmente en NNF.

Definición:

$$k\_t4p\_p(n) := E(n) \vee nnf(\neg C(n)) \vee \Diamond p_4$$

$$C(i) := \begin{cases} ((\Box\Diamond p_0 \to \Diamond p_1) \wedge \Box(\Box\neg\Box\Diamond p_1 \to \neg\Box\Diamond p_0) & \\ \wedge(\Box\Diamond p_0 \to \Box\Box\Diamond p_1) & \\ \wedge\Box(\Diamond\Box\Diamond p_0 \wedge p_0 \to \Box p_1) \wedge \Box\Diamond p_1)\{p_0 \wedge \Diamond\neg p_3/p_1\} & i = 0 \\ \Box A4\{p_1/p_0\} \wedge \Box C(i-1) \wedge A4\{\Diamond p_1/p_0\} & \text{caso contrario} \end{cases}$$

$$E(i) := \begin{cases} \Diamond\Box p_0 & i = 0 \\ \Diamond\neg A4\{\neg p_1/p_0\} \vee \Box E(i-1) \vee \Box A5\{p_1/p0\} & \text{caso contrario} \end{cases}$$

5.2.6. k_t4p_p

Idea: $\Diamond\Box p_0$ no es demostrable en **K** más cualquier instancia de **T**, **A4** y $\Box(\Diamond\Box\Diamond p_0 \to (p_0 \to \Box p_0))$

Ocultamiento: como en k_t4p_n.

Características: 4 variables, profundidad modal $2n + 4$. Parcialmente en forma normal negada.

Definición:

$$k\_t4p\_n(n) := E(2n-1) \vee nnf(\neg C(2n-1)) \vee \Diamond p_4$$

$$C(i) := \begin{cases} ((\Box\Diamond p_0 \to \Diamond p_1) \wedge \Box(\Box\neg\Box\Diamond p_1 \to \neg\Box\Diamond p_0) & \\ \wedge(\Box\Diamond p_0 \to \Box\Box\Diamond p_1) & \\ \wedge\Box(\Diamond\Box\Diamond p_0 \wedge p_0 \to \Box p_1))\{p_0 \wedge \Diamond\neg p_3/p_1\} & i = 0 \\ \Box A4\{p_1/p_0\} \wedge \Box C(i-1) \wedge A4\{\Diamond p_1/p_0\} & \text{caso contrario} \end{cases}$$

$E(i)$ como en k_t4p_n

5.3. Resultados

En esta sección mostraremos los resultados obtenidos para las tres clases de fórmulas que presentamos en la sección anterior.

Para evaluar el comportamiento de las traducciones utilizamos dos demostradores de teoremas: *SPASS* y *HTab*.

SPASS es un demostrador de teoremas para lógica de primer orden con equivalencia que puede manejar sorts. Además, implementa los refinamientos de resolución necesarios para asegurar los resultados de terminación de las traducciones que vimos anteriormente.

HTab es un demostrador basado en tableaux, para lógicas híbridas que implementa algoritmos que aseguran terminación para los fragmentos decidibles.

Con SPASS pondremos a prueba las distintas traduccioness funcionales implementadas, mientras que HTab recibirá solamente las fórmulas modales en forma normal negada (NNF) y nos servirá como punto de comparación entre un enfoque general (el que combina demostradores de lógica de primer orden como SPASS y traducciones) y uno más especializado y optimizado para las lógicas híbridas (el que representa HTab). Además, HTab nos permitira corroborar o, al menos, aumentar el grado de confianza en la corrección de las traducciones de las fórmulas aleatorias del capítulo siguiente (en donde no sabemos si la fórmula es satisfacible o no).

En cada gráfico de esta sección se muestran las siguientes curvas de tiempos de ejecución:

- **htab:** HTab con las fórmulas modales en NNF.
- **fun_spass:** SPASS con la traducción funcional de las fórmulas modales en NNF.

- **fun_sin_sorts_spass:** idem anterior, pero para la traducción funcional sin sorts.
- **opt_spass:** idem anterior, pero para la traducción funcional optimizada.
- **opt_sin_sorts_spass:** idem anterior, pero para la traducción funcional optimizada sin sorts.
- **fun_mspass:** idem anterior, pero para la traducción funcional implementada por SPASS (MSPASS).
- **poly_fun_mspass:** idem anterior, pero para la traducción funcional poliádica implementada por SPASS (MSPASS).

Las traducciones *fun_mspass* y *poly_fun_mspass* no fueron implementadas por nosotros, se encuentran incluidas en SPASS. La primera es muy similar a nuestra implementación de la traducción funcional (*fun_spass*) pero codifica la información de sorts utilizando predicados unarios. Por otro lado, *poly_fun_mspass* es la traducción funcional poliádica, traducción que codifica la información de sorts en el nombre de los predicados, que en este caso son k-arios. Por lo tanto es una traducción funcional sin sorts, similar a las nuestras, pero que preserva satisfacibilidad sólo en el lenguaje modal básico $\mathbf{K}_n$, o la extensión de $\mathbf{K}_n$ que usa el axioma de serialidad **D** [Hustadt and Schmidt, 1999]. Además, SPASS utiliza parámetros diferentes a los nuestros (los parámetros por defecto) al trabajar sobre estas traducciones.

En el eje vertical de los gráficos colocamos el tiempo, en segundos, de las ejecuciones de los demostradores. Para cada demostrador usamos un límite de tiempo de 180 segundos, es decir, si el demostrador no obtenia un resultado en 180 segundos, o menos, se terminaba su ejecución.

En el eje horizontal colocamos el parámetro n de las fórmulas del benchmark. Mientras más grande es n, más grande y dificíl para el demostrador es la fórmula.

Dada ya la información para entender los gráficos, estamos casi en condiciones de pasar a observar los resultados obtenidos.

En esta tesis nos interesa comparar los tiempos entre las traducciones que usan sorts y las que no. Por este motivo, compararemos *fun_spass* vs *fun_sin_sorts_spass*, *opt_spass* vs *opt_sin_sorts_spass* y *fun_mspass* vs *pol_fun_mspass*. Además será interesante comparar el desempeño de un demostrador especializado y optimizado para lógica híbrida como lo es HTab contra el método más general y flexible que se obtiene al utilizar las traducciones funcionales y SPASS.

La Figura 5.1 muestra los tiempos de ejecución de los demostradores con las distintas traducciones para el problema *Poly* satisfacible.

Si nos enfocamos en la instancia $n = 8$ del problema se observan los siguientes tiempos de ejecución (en segundos):

Traducción	Tiempo (segundos)
htab	0,12
fun_spass	5,91
fun_sin_sorts_spass	0,72
opt_spass	179,75
opt_sin_sorts_spass	0,9
fun_mspass	4,99
pol_fun_mspass	0,51

Claramente se observa una ventaja de aproximadamente 5 segundos de las traducciones funcionales sin sorts sobre las que usan sorts. La traducción funcional optimizada con sorts se desempeña pobremente, SPASS demoró cerca de 3 minutos en obtener un resultado, mientras que las otras traducciones lo hacen en decimas de segundo. El simple hecho de removerle los sorts a esta traducción optimizada hizo que SPASS demore 3 minutos menos, obteniendo una aceleración muy significativa. Por otro lado, y como se esperaba, HTab es mucho más rápido

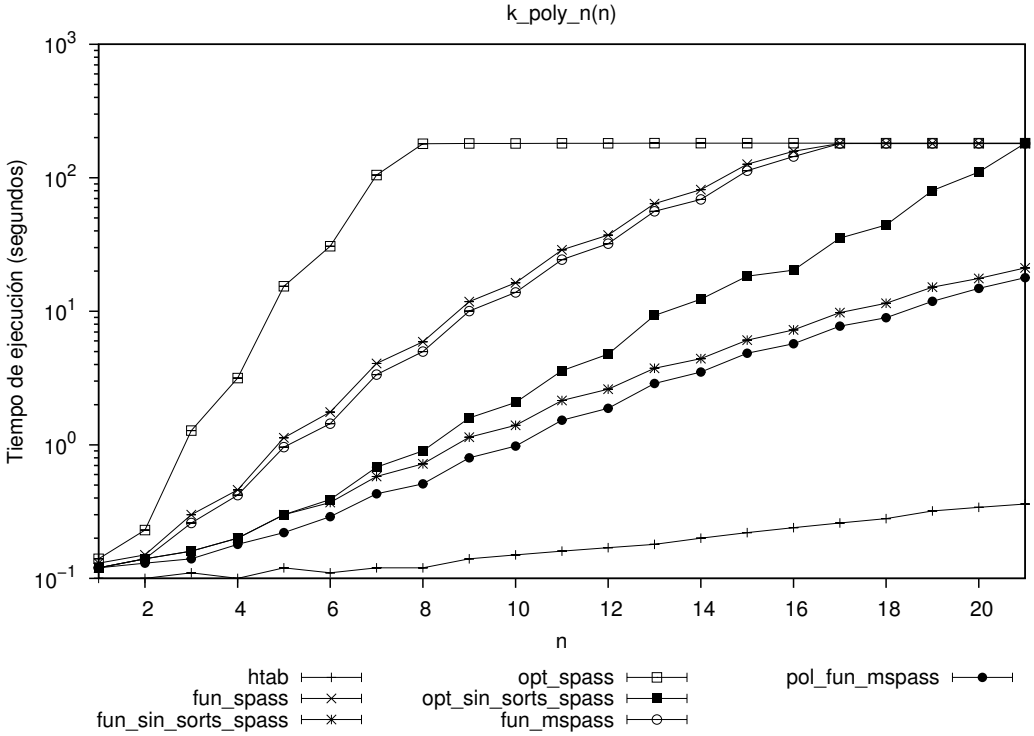


Figura 5.1: Tiempos de ejecución para el problema Poly satisfacible

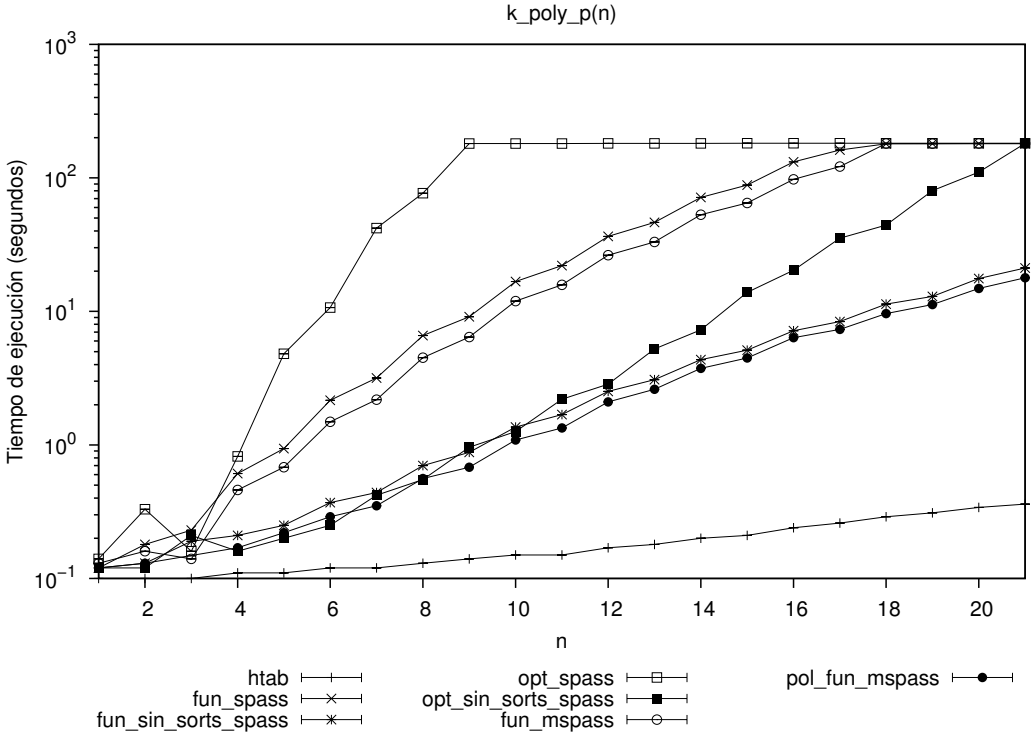


Figura 5.2: Tiempos de ejecución para el problema Poly no satisfacible

sobre las fórmulas modales que SPASS con las traducciones, con un desempeño del orden de la décima de segundo.

Ahora observaremos los tiempos de ejecución para una instancia mucho más difícil del pro-

blema. En la instancia $n = 14$, del problema Poly satisfacible, se observan los siguientes tiempos de ejecución:

Traducción	Tiempo (segundos)
htab	0,2
fun_spass	81,59
fun_sin_sorts_spass	4,42
opt_spass	>180
opt_sin_sorts_spass	12,33
fun_mspass	68,84
pol_fun_mspass	3,51

Se puede notar que los demostradores demoraron más en obtener el resultado. Esto se debe a que en esta instancia tenemos $14/8 = 1{,}75$ veces más proposiciones y profundidad modal que en la instancia $n = 8$. Sin embargo, las traducciones funcionales sin sorts siguen aventajando a sus respectivas traducciones con sorts. Por ejemplo la traducción funcional sin sorts demora solo el 5 % del tiempo de ejecución de la traducción funcional (con sorts). Es decir, las diferencias de los tiempos de ejecución del demostrador para las traducciones con sorts y sin sorts tienden a crecer a medida que aumenta n, y lo hacen muy rápido.

Por otro lado, a medida que crece n, HTab demuestra claramente que es muy superior a la combinación de SPASS con las traducciones funcionales. Por ejemplo, para $n = 21$ HTab demora $0,36$ segundos; mientras que la traducción poliádica, la mejor traducción funcional, demora 17 segundos.

Es oportuno destacar que, si bien los enfoques que usan traducciones a lógica de primer orden son en general más lentos que los métodos específicos para lógica modal, los primeros poseen la ventaja de poder tratar cualquier lógica modal capaz de ser embebida en lógica de primer orden. El método de traducciones es genérico: puede manejar lógicas modales de primer orden, lógicas modales indecidibles, combinaciones de lógicas modales y no modales. Además, las traducciones son bastante sencillas y pueden realizarse en $O(n * log(n))$.

El comportamiento de las distintas traducciones en los casos no satisfacibles es muy similar, por lo cuál sólo analizaremos en detalle las clases de fórmulas satisfacibles. Los tests muestran en estos casos que el comportamiento de las distintas traducciones está correlacionado con la estructura de las fórmulas y no con el hecho de que la instancia sea satisfacible o no.

Ahora pasaremos a analizar los gráficos del problema Branch. En la Figura 5.3 se observan los resultados para la versión satisfacible del problema.

Como hicimos con Poly, nos posicionamos en la instancia $n = 8$. En ella se pueden observar los siguientes tiempos:

Traducción	Tiempo (segundos)
htab	60,64
fun_spass	21,87
fun_sin_sorts_spass	7,31
opt_spass	>180
opt_sin_sorts_spass	1,92
fun_mspass	14,94
pol_fun_mspass	6,69

Se puede observar un patrón muy parecido al problema Poly. Claramente remover sorts afectó positivamente los tiempos de ejecución de SPASS, a tal punto de que la traducción funcional optimizada pasó de registrar el peor tiempo, demorando más de 3 minutos; a registrar el mejor tiempo, con 1,92 segundos.

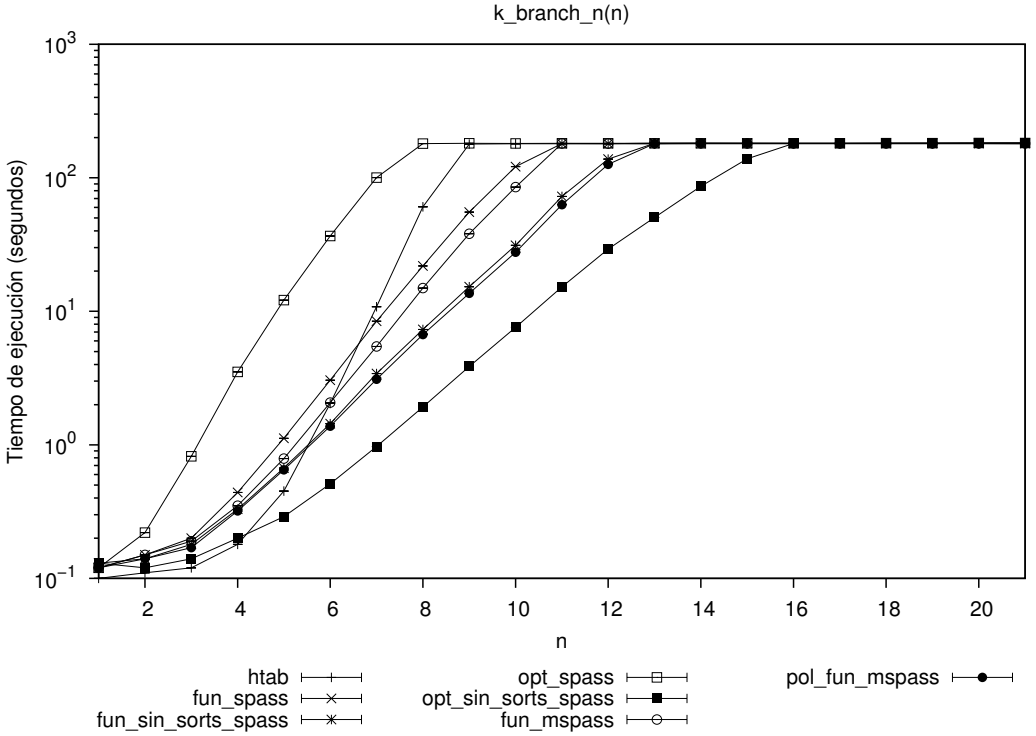


Figura 5.3: Tiempos de ejecución para el problema Branch satisfacible

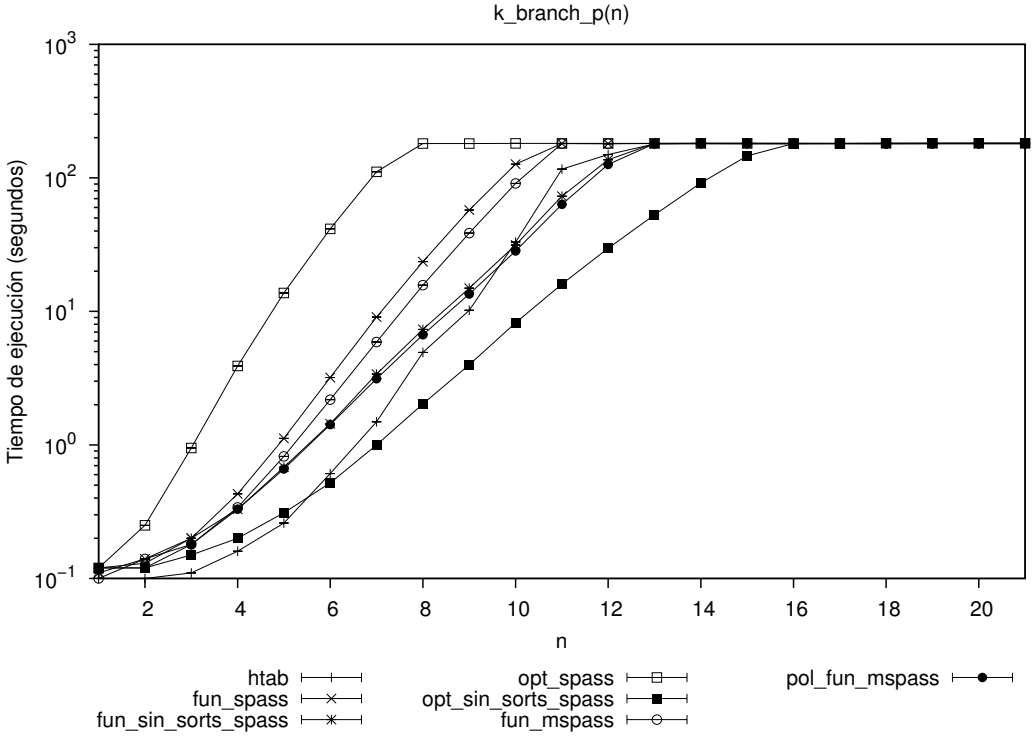


Figura 5.4: Tiempos de ejecución para el problema Branch no satisfacible

A diferencia del problema Poly, aquí HTab se comportó deficientemente comparado con la mayoria de las traducciones. Esto puede deberse a que Branch es un problema bastante especial, pues en el caso satisfacible se necesita generar un modelo de 2^n estados.

Ya en la instancia $n = 14$, la única traducción que no supera el tiempo límite de 180 segundos es la traducción funcional optimizada sin sorts, demorando $50,41$ segundos.

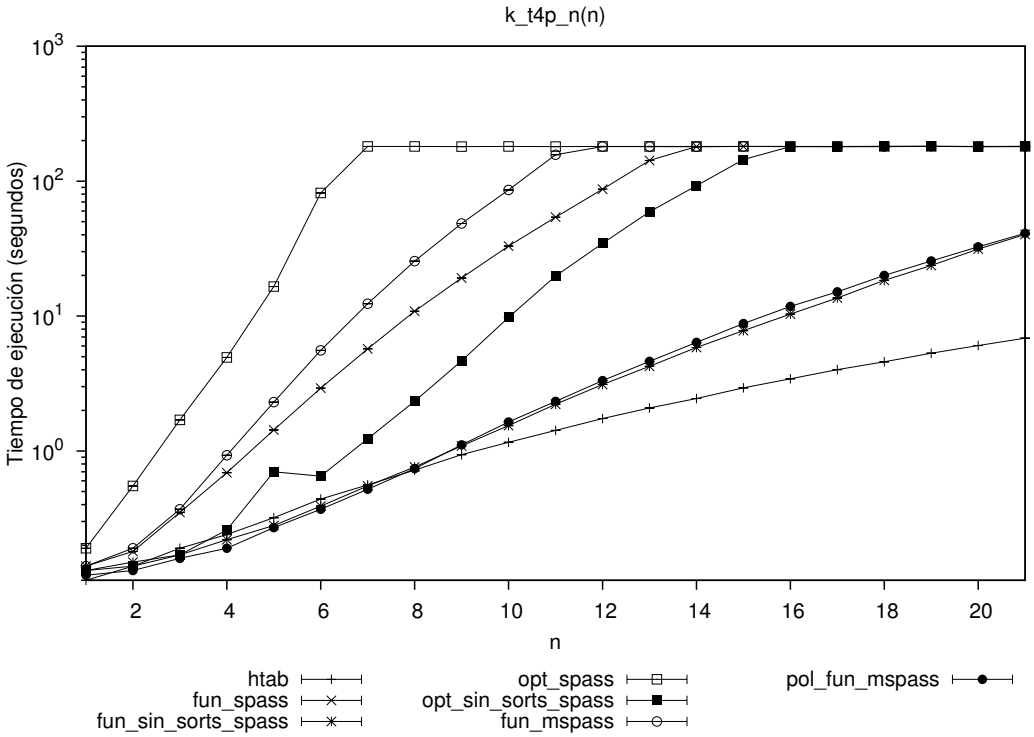


Figura 5.5: Tiempos de ejecución para el problema T4p satisfacible

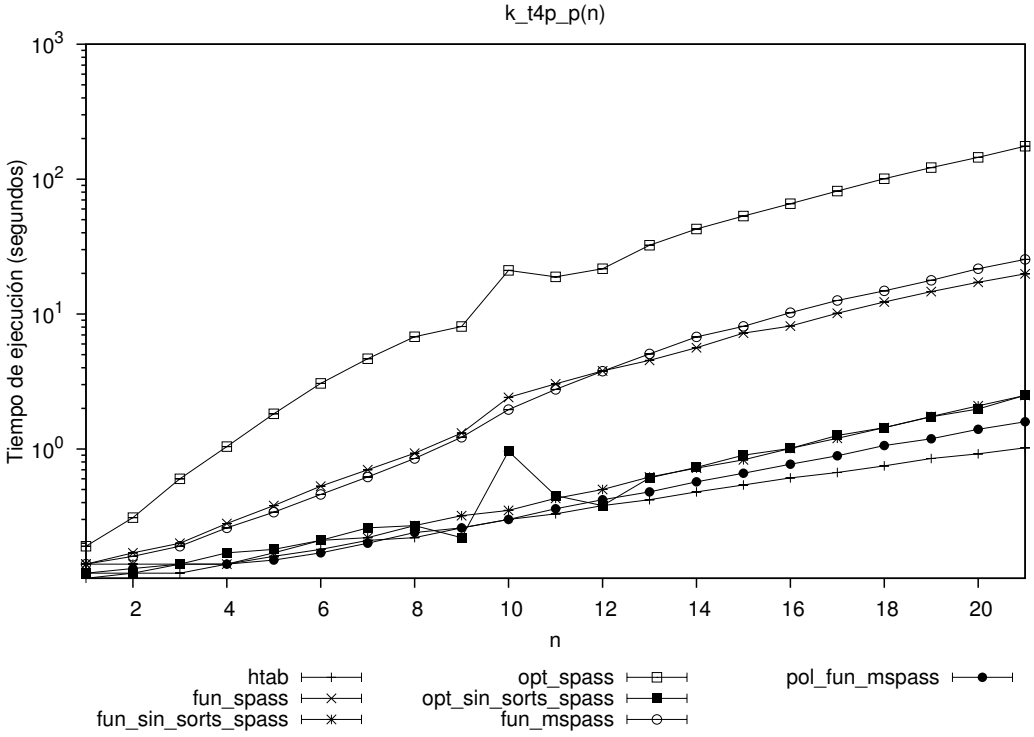


Figura 5.6: Tiempos de ejecución para el problema T4p no satisfacible

Por último, en las Figuras 5.5 y 5.6 podemos ver los resultados para el problema T4p. En la instancia $n = 8$ tenemos los siguientes resultados:

Traducción	Tiempo (segundos)
htab	0,72
fun_spass	10,85
fun_sin_sorts_spass	0,76
opt_spass	>180
opt_sin_sorts_spass	2,34
fun_mspass	25,53
pol_fun_mspass	0,74

Para $n = 14$ tenemos los siguientes resultados:

Traducción	Tiempo (segundos)
htab	2,45
fun_spass	>180
fun_sin_sorts_spass	5,84
opt_spass	>180
opt_sin_sorts_spass	92,42
fun_mspass	>180
pol_fun_mspass	6,38

De nuevo, se observa que remover sorts acelera el proceso de demostración. Una diferencia que aparece en este gráfico con respecto a los anteriores es que: la traducción funcional optimizada sin sorts demora más que las otras traducciones sin sorts (funcional y poliádica), contrastando con los resultados del problema Branch en el cuál ésta traducción era la mejor.

Podemos notar también que el problema no satisfacible es ligeramente más fácil que el problema satisfacible (en los gráficos anteriores esta diferencia era menos notoria). Esto es lógico: SPASS, y en general, todos los demostradores basados en resolución, se comportan mejor en los problemas no satisfacibles por qué sólo tienen que derivar la cláusula vacía.

5.4. Conclusiones

En este capítulo presentamos los resultados del desempeño de los demostradores, al usar traducciones de fórmulas del benchmark del LWB.

Observamos que el desempeño de las traducciones sin sorts es claramente superior al de las traducciones con sorts.

También notamos que en algunos problemas SPASS supera en desempeño a HTab, un demostrador especializado en lógicas híbridas (aunque no el más optimizado).

Por lo tanto, podemos concluir que remover sorts de las traducciones funcionales, de fórmulas del lenguaje modal básico **K**, es una forma efectiva de optimizar el desempeño de demostradores de lógica de primer orden como SPASS.

Capítulo 6

Benchmarks usando fórmulas aleatorias

6.1. Introducción

El benchmark del capítulo anterior provee un conjunto interesante de fórmulas. Pero, a pesar de que las clases son diferentes, no cubren todos los tipos de entradas. Además, los sistemas de prueba actuales, incorporan etapas de preprocesamiento que reducen muchas de las clases a fórmulas triviales mucho antes de que la busqueda comience.

Como observamos en el capítulo anterior, varias clases de fórmulas eran resueltas fácilmente por los probadores, de modo que no pudimos usarlas para obtener conclusiones. Es el caso de las clases: lin, dum y path.

Se podría agregar más clases de fórmulas al benchmark para resolver estos problemas, pero es difícil crear manualmente fórmulas resistentes a estas normalizaciones de entrada y resulta imposible imaginar fórmulas que resistan futuras optimizaciones.

Por estos motivos, en los últimos años, el esfuerzo se concentró en el diseño de generadores de fórmulas aleatorias, apropiados para la lógica modal básica ([Hustadt and Schmidt, 1997],[Horrocks, 2000], [Patel-Schneider *et al.*, 2003]).

En la sección siguiente introduciremos el testeo aleatorio usando fórmulas en CNF, y daremos detalles de $CNF_{\Box_m}$, el nombre dado usualmente a las fórmulas multi-modales en forma normal conjuntiva. Luego en la sección 6.3 presentaremos *hGen* un generador CNF para lenguajes híbridos. Por último, mostraremos los resultados obtenidos para distintos subconjuntos del lenguaje híbrido $H(@,\downarrow)$.

6.2. Generadores CNF proposicionales y modales

El problema de satisfacibilidad para la lógica proposicional ha sido ampliamente investigado dado que tiene muchas aplicaciones en timetabling, optimización de código o criptografia. Es sabido que las clausulas CNF aleatorias de tres literales (3CNF) capturan la complejidad del problema de satisfacibilidad para esa lógica. Por lo tanto, aunque haya muchos problemas extraidos de aplicaciones reales y conjuntos de tests para esta lógica, uno de los más conocidos y ampliamente utilizados es Random 3SAT: una conjunción de L claúsulas de 3 literales proposicionales cada una, elegidos de un conjunto de N variables proposicionales diferentes. Dado que 3SAT se ha convertido en el test aleatorio estandar para el testeo de satisfacibilidad en lógica proposicional, desarrollar una versión modal de este test ha recibido naturalmente mucho atención.

Una fórmulas modal en CNF, una fórmula $CNF_{\Box_m}$, es una conjunción de claúsulas $CNF_{\Box_m}$, donde cada claúsula es una disjunción de un cierto número de literales proposicionales o modales.

Un literal, es un átomo o su negación, los átomos modales son fórmulas de la forma $\Box_i C$, donde C está en $CNF_{\Box_m}$. Una fórmula $3CNF_{\Box_m}$ es una fórmula $CNF_{\Box_m}$ donde todas las claúsulas tienen tres literales.

Un número de generadores de fórmulas $CNF_{\Box_m}$ ha sido propuesto en la literatura. La última versión acepta cinco parámetros principales: la máxima profundidad modal D, el número de variables proposicionales N, el número de modalidades m, el número de claúsulas L, y la probabilidad p de que un átomo puramente proposicional ocurra a profundidad menor a d. A pesar de que el número usual de literales por claúsula es tres, el generador brinda un alto grado de control sobre el tamaño de la claúsula. De hecho, el balance modal/proposicional y la distribución de probabilidad del tamaño de la claúsula pueden ser especificada como constantes o como una función de la profundidad modal.

Dados estos parámetros, una fórmula $CNF_{\Box_m}$ de profundidad D es un conjunto de L claúsulas, cada una hecha de un número de distintas disjunciones CNF, consistentes en una proposición del conjunto $\{P_1, \ldots, P_n\}$ o, si $D > 0$, una fórmula $\Box_r C$, donde $\Box_r \in \{\Box_1, \ldots, \Box m\}$, y C es una cláusula $CNF_{\Box_m}$ de profundidad $(D-1)$.

Una corrida estandar del test usando el generador $CNF_{\Box_m}$ es así: todos los parámetros salvo L se mantienen fijos, luego se selecciona un rango de L que cubra la transición desde "sólo fórmulas satisfacibles son generadas" a "sólo fórmulas no satisfacibles son generadas". Un número fijo de fórmulas son generadas para cada una de las configuraciones de los parámetros, y son pasadas como entrada al probador bajo evaluación, generalmente con un tiempo límite. La mediana del tiempo de CPU transcurrido, la proporción de fórmulas satisfacibles y otros posibles indicadores son graficados contra L o L/N.

6.3. El generador híbrido hGen

El generador híbrido *hGen* [Areces and Heguiabehere, 2003] extiende el generador $CNF_{\Box_m}$ descripto en [Patel-Schneider and Sebastiani, 2003]. *hGen* esta implementado en haskell y acepta los siguientes parámetros:

- El máximo anidamiento de operadores D;
- El número de variables proposicionales, nominales y variables de estado: N_p, N_n y N_x;
- El número de modalidades N_m;
- El número de claúsulas L;
- La distribución de probabilidades del tamaño de la claúsula (una lista $[f_1, \ldots, f_n]$ con f_i la frecuencia relativa de las claúsulas de tamaño i)
- La probabilidad de un disjunto de ser no atómico p_{op}
- Las frecuencias relativas de los operadores modales (@, $\downarrow$, universal): p_{mod}, p_{down}, p_{at} y p_{univ}.
- Las frecuencias relativas de las proposiciones, nominales y variables de estado en los disjuntos atómicos: p_{prop}, p_{nom} y p_{svar};
- La probabilidad de que cualquier literal aparezca negado p_{neg};
- El número de instancias a generar *numinst*.

Dados estos parámetros, una fórmula híbrida en CNF de profundidad D es un conjunto de L claúsulas, cada una hecha de (un número elegido de $[f_1, \ldots, f_n]$) disjuntos híbridos en CNF que consisten en:

- una proposición del conjunto $\{P_1, \ldots, P_{N_p}\}$, ó
- un nominal del conjunto $\{n_1, \ldots, n_{N_n}\}$, ó
- una variable de estado del conjunto $\{x_1, \ldots, x_{N_x}\}$, ó
- si $D > 0$
  - un disjunto $\Box_r C$, donde $\Box_r \in \{\Box_1, \ldots, \Box_{N_m}\}$ y C es una claúsula híbrida en CNF de profundidad $(D-1)$, ó
  - un disjunto $@_n C$, donde $n \in \{n_1, \ldots, n_{N_n}\}$ y C es una claúsula híbrida en CNF de profundidad $(D-1)$, ó
  - un disjunto $\downarrow x_r \; op \; C$, donde $x_r \in \{x_1, \ldots, x_{N_x}\}$, $op \in \{@, \Box, \mathbf{A}\}$ (que puede aparecer aún), C es una claúsula híbrida en CNF con profundidad $(D-1)$, ó
  - un disjunto $\mathbf{A}$C, donde C es una claúsula híbrida en CNF con profundidad $(D-1)$.

Algunos de los parámetros de hGen se mantendrán fijos en todos los tests. Estos parámetros son:

- el número de modalidades es 1 en todos los casos: $N_m = 1$,
- la probabilidad de que un disjunto sea modal (no sea un átomo) es: $p_{op} = 0{,}5$,
- la probabilidad de que un literal este negado: $p_{neg} = 0{,}5$

6.4. Benchmarks para K

En esta sección presentaremos los resultados obtenidos al testear las traducciones utilizando el generador aleatorio *hGen* para obtener fórmulas del lenguaje modal básico **K**.

En la figura 6.2 podemos observar el gráfico de los tiempos de ejecución de hTab y SPASS al resolver las distintas traducciones de las fórmulas generadas por hGen utilizando los siguientes parámetros para generar el lenguaje modal básico: $N_p = 3$ (número de proposiciones distintas), $Nm = 1$ (número de modalidades), $D = 2$ (máximo anidamiento de $\Box$), distribución de tamaños de claúsulas: 50 % con dos literales y 50 % con tres literales ($[0, 1, 1]$).

Antes de pasar a comentar el gráfico de la figura 6.2 explicaremos el gráfico de la figura 6.1 que muestra lo que se conoce como fenómeno de transición de fase [Gent and Walsh, 1994]. Este es un punto, relacionado al radio N_p/L (número de proposiciones diferentes/número de claúsulas), en el que se pasa abruptamente de una mayoria de fórmulas satisfacibles a tener una mayoria de fórmulas no satisfacibles. Esto coincide en muchos casos con un patrón de fórmulas fáciles-difíciles-fáciles, es decir, las fórmulas más difíciles se encuentran en el punto de transición de fase, mientras que las fórmulas con gran probabilidad de ser satisfacibles y las fórmulas con gran probabilidad de ser no satisfacibles son más fáciles de detectar. Sin embargo, por la forma en que se generan las fórmulas en hGen, las fórmulas con gran probabilidad de ser no satisfacibles son mucho más grandes (ya que contienen más claúsulas). El mayor tamaño resulta en un incremento de tiempo de procesamiento que se suma al tiempo necesario para detectar que son inconsistentes. El resultado es un patrón mayormente monónotono donde la complejidad aumenta gradualmente. Lo importante, en todo caso, es asegurar que los test atraviesen la zona de cambio de fase, para tener cierta seguridad de que no se encontraran "sorpresas" (instancias de mucha mayor complejidad) fuera del benchmark. En la figura 6.1 se observa la transición de fase alrededor de las $L = 70$ claúsulas.

Ahora sí pasamos al gráfico de la figura 6.2, observamos los tiempos de ejecución de los probadores. Para $L = 70$ tenemos los siguientes tiempos:

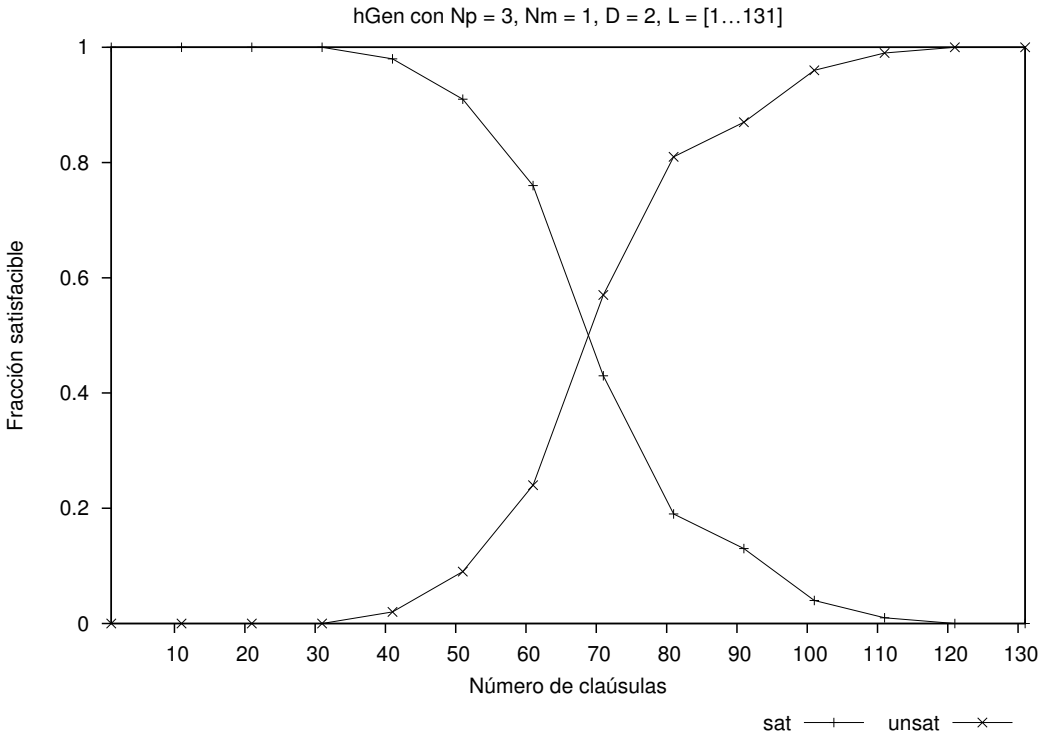


Figura 6.1: Transición de fase satisfacible/no satisfacible

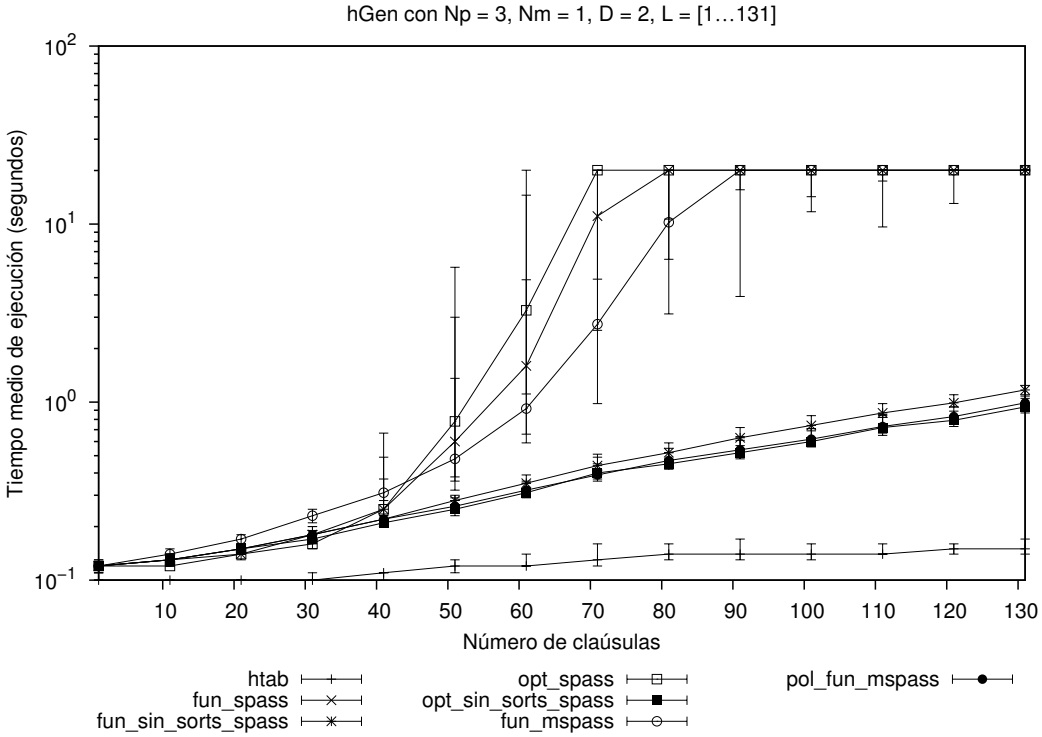


Figura 6.2: Tiempos de ejecución para fórmulas aleatorias de **K** con $D = 2$

Traducción	Tiempo (segundos)
htab	0,13
fun_spass	11,09
fun_sin_sorts_spass	0,44
opt_spass	>20
opt_sin_sorts_spass	0,4
fun_mspass	2,74
pol_fun_mspass	0,39

Podemos ver que para fórmulas generadas aleatoriamente se sigue obteniendo el mismo resultado del capítulo anterior, es decir, las traducciones funcionales sin sorts se comportan mucho mejor que sus respectivas traducciones con sorts.

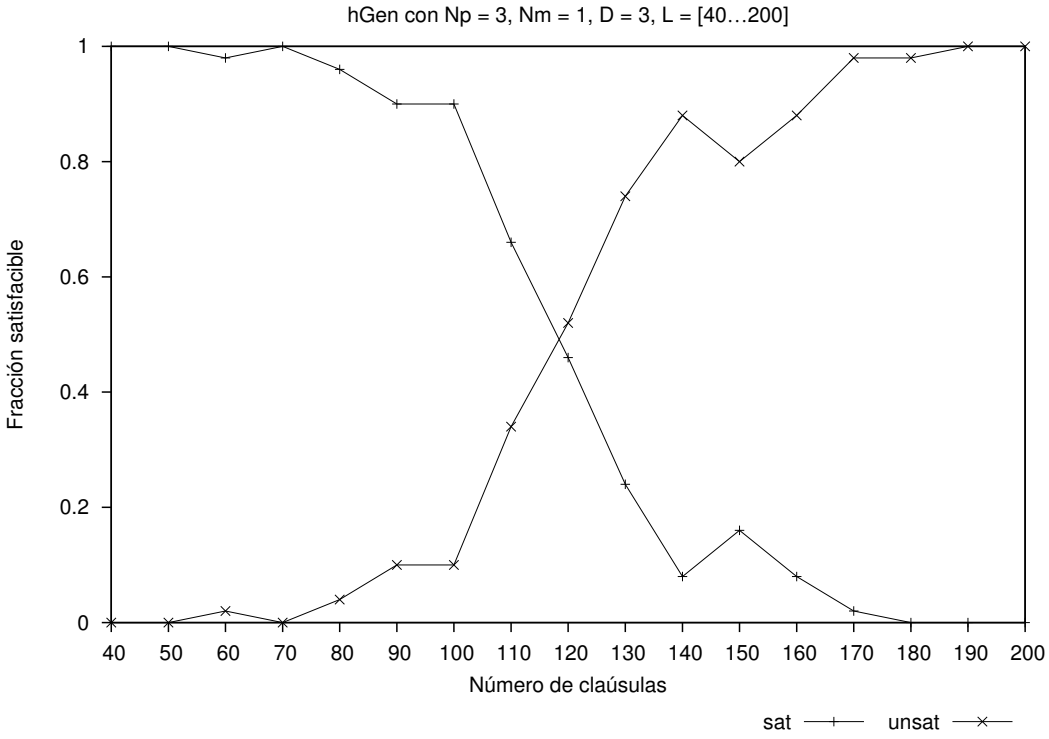


Figura 6.3: Transición de fase satisfacible/no satisfacible

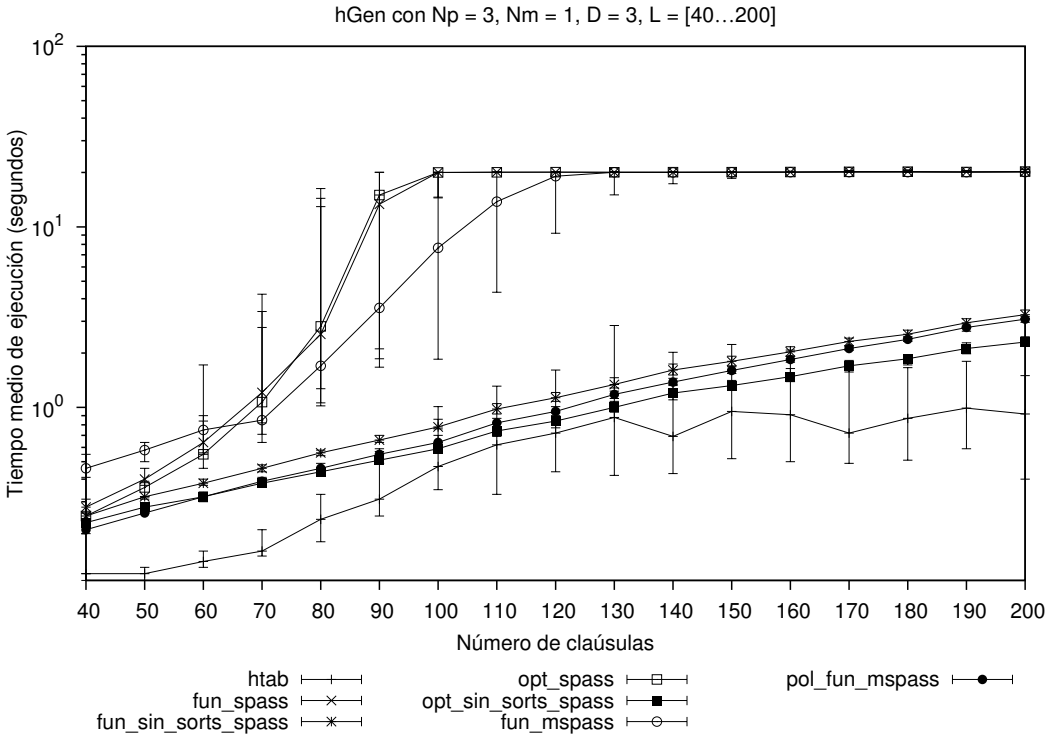


Figura 6.4: Tiempos de ejecución para fórmulas aleatorias de **K** con $D = 3$

En la figura 6.4 observamos los tiempos para fórmulas similares a las anteriores pero con anidamiento máximo $D = 3$ y todas las claúsulas tienen dos literales. Para $L = 120$, donde se encuentra la transición de fase, se registran estos tiempos de ejecución:

Traducción	Tiempo (segundos)
htab	0,72
fun_spass	>20
fun_sin_sorts_spass	1,13
opt_spass	>20
opt_sin_sorts_spass	0,84
fun_mspass	19,08
pol_fun_mspass	0,95

Si comparamos con el gráfico anterior, notaremos que aquí las curvas se encuentran un poco más arriba, es decir, los probadores demoraron más tiempo y por ende estas fórmulas aleatorias son un poco más difíciles que las anteriores. Sin embargo, se observa que las traducciones sin sorts siguen siendo mejores que las que mantienen los sorts.

Con estos resultados, damos por concluido el testeo de fórmulas del lenguaje modal básico **K**. Hemos mostrado que tanto para fórmulas creadas manualmente como para fórmulas generadas aleatoriamente, las traducciones funcionales sin sorts tienen un desempeño mayor al de las traducciones sin sorts.

6.5. Benchmarks para $\mathcal{H}$

En esta sección testearemos las traducciones funcionales usando fórmulas del lenguaje híbrido básico $\mathcal{H}$ generadas aleatoriamente con hGen.

Recordemos que el lenguaje híbrido $\mathcal{H}$ es el lenguaje **K** extendido con nominales.

Para que hGen genere fórmulas en $\mathcal{H}$ utilizaremos los parámetros p_n (proporción de nominales) y N_n (cantidad de nominales). Fijaremos $N_n = 3$ y veremos que ocurre para $p_n = 50\,\%, 33\,\%$ y $20\,\%$; es decir, reduciremos progresivamente la cantidad de nominales en relación a la cantidad de proposiciones.

En las figuras 6.5 y 6.6 observamos los gráficos de la transición de fase y de los tiempos de ejecución para $p_n = 50\,\%$, respectivamente. Aproximadamente en $L = 80$ claúsulas tenemos un $50\,\%$ de fórmulas satisfacibles y un $50\,\%$ de fórmulas no satisfacibles. Los tiempos para $L = 80$ son los siguientes:

Traducción	Tiempo (segundos)
htab	0,16
fun_spass	0,68
fun_sin_sorts_spass	0,63
opt_spass	0,68
opt_sin_sorts_spass	0,58

Las fórmulas generadas resultaron ser demasiado fáciles y los probadores las resolvieron en menos de un segundo a la mayoria.

Ahora veremos que ocurre si empezamos a reemplazar nominales por proposiciones.

En la figura 6.8 se observan los resultados al disminuir la cantidad de nominales a un $33\,\%$ ($67\,\%$ proposiciones). Para $L = 80$ se registran los tiempos:

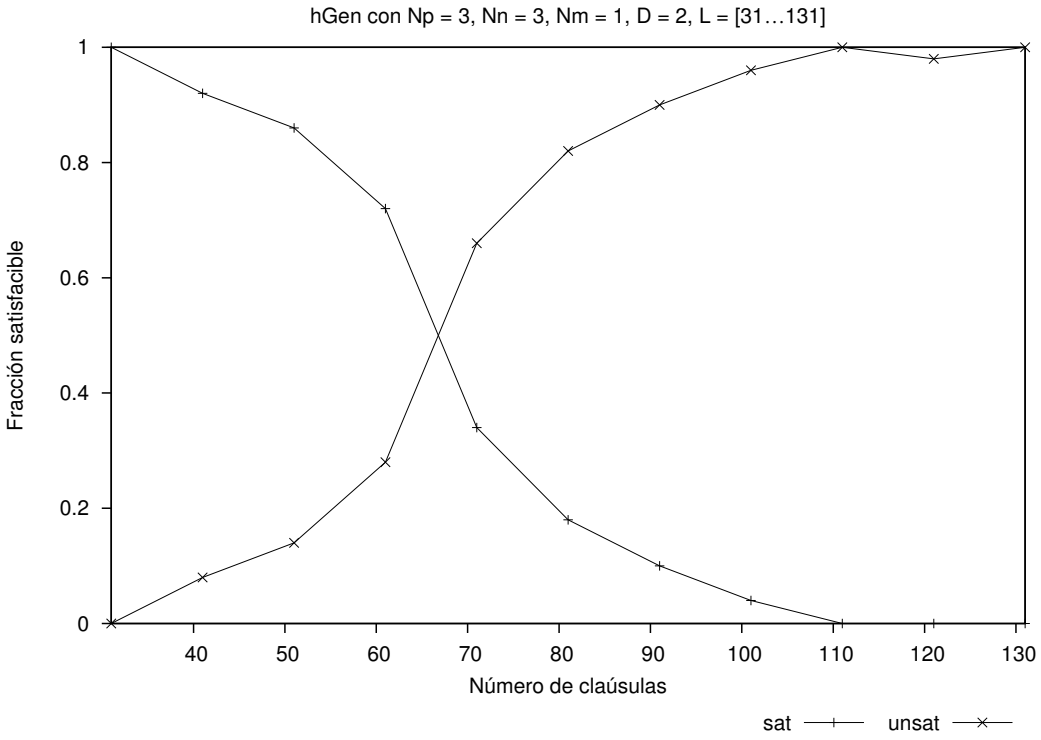


Figura 6.5: Transición de fase satisfacible/no satisfacible

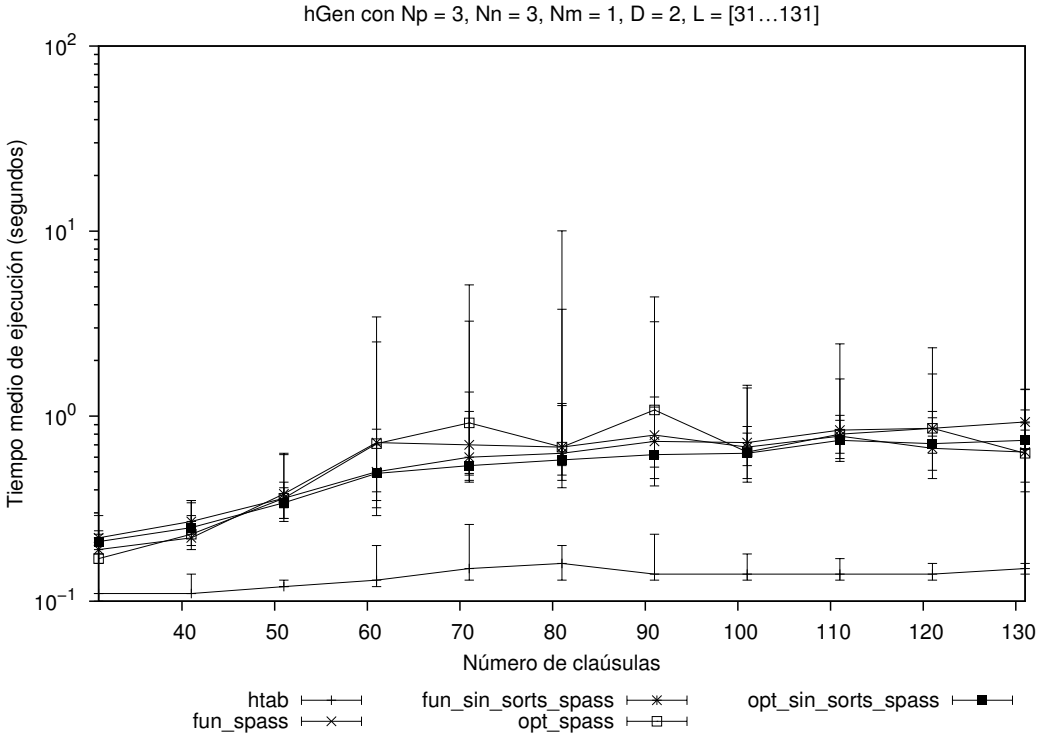


Figura 6.6: Tiempos de ejecución para fórmulas de $\mathcal{H}$ (50 % nominales)

Traduccion	Tiempo (segundos)
htab	0,18
fun_spass	3,31
fun_sin_sorts_spass	0,86
opt_spass	2,79
opt_sin_sorts_spass	0,77

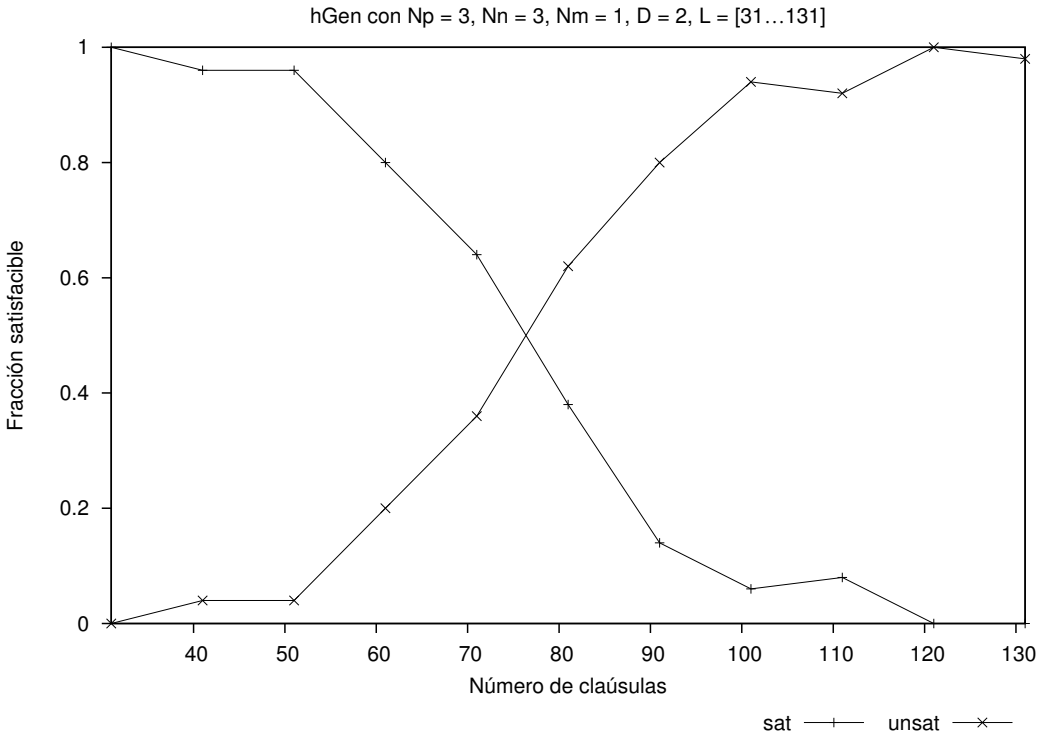


Figura 6.7: Transición de fase satisfacible/no satisfacible

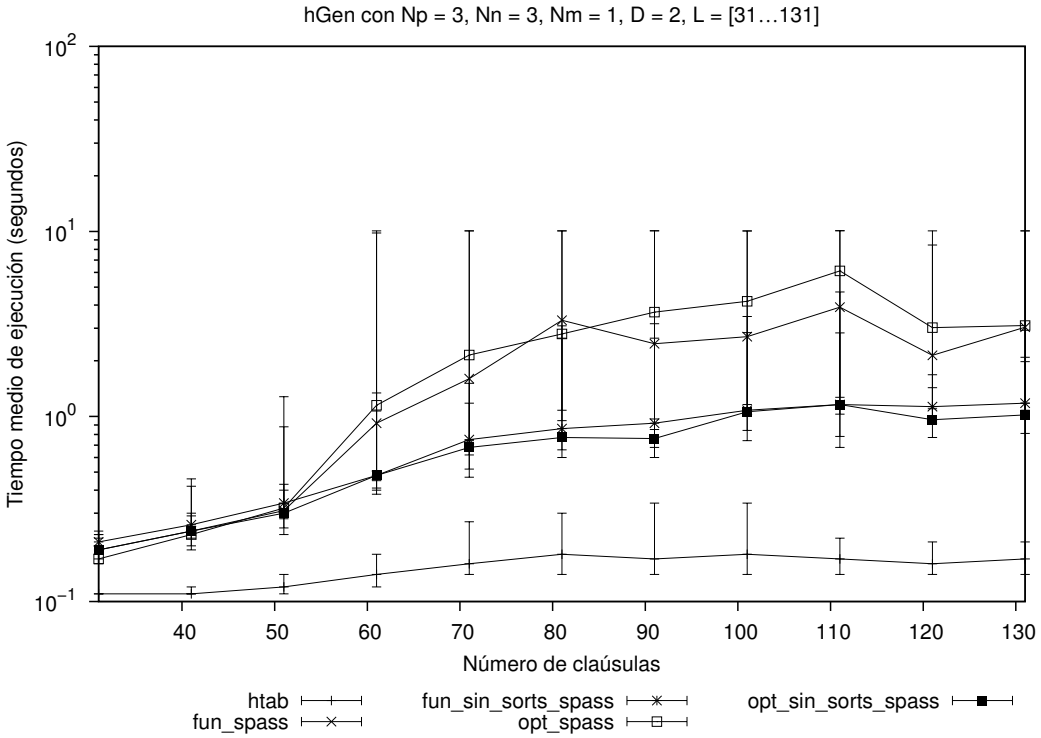


Figura 6.8: Tiempos de ejecución para fórmulas de $\mathcal{H}$ (33 % nominales)

Podemos ver que se produjo un incremento en los tiempos de ejecución, sobresaliendo los tiempos de las traducciones con sorts. Reemplazar nominales por proposiciones incrementó la dificultad de las fórmulas, a tal punto que se empieza a notar una pequeña diferencia entre las

traducciones con sorts y las traducciones sin sorts en favor de estas últimas.

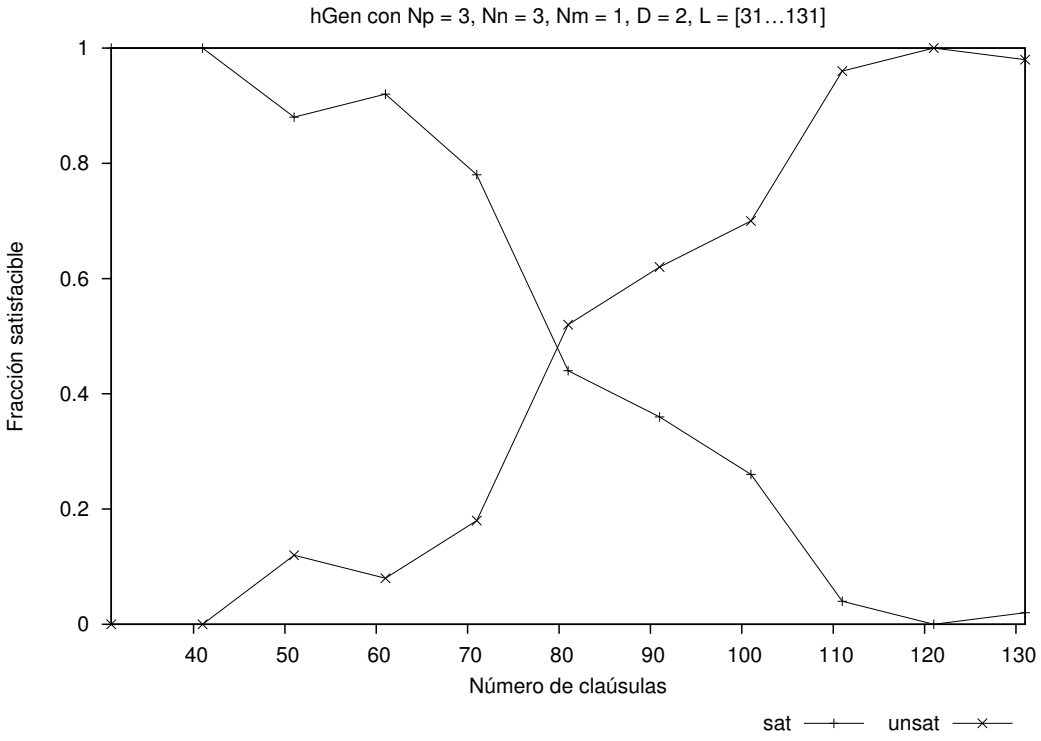


Figura 6.9: Transición de fase satisfacible/no satisfacible

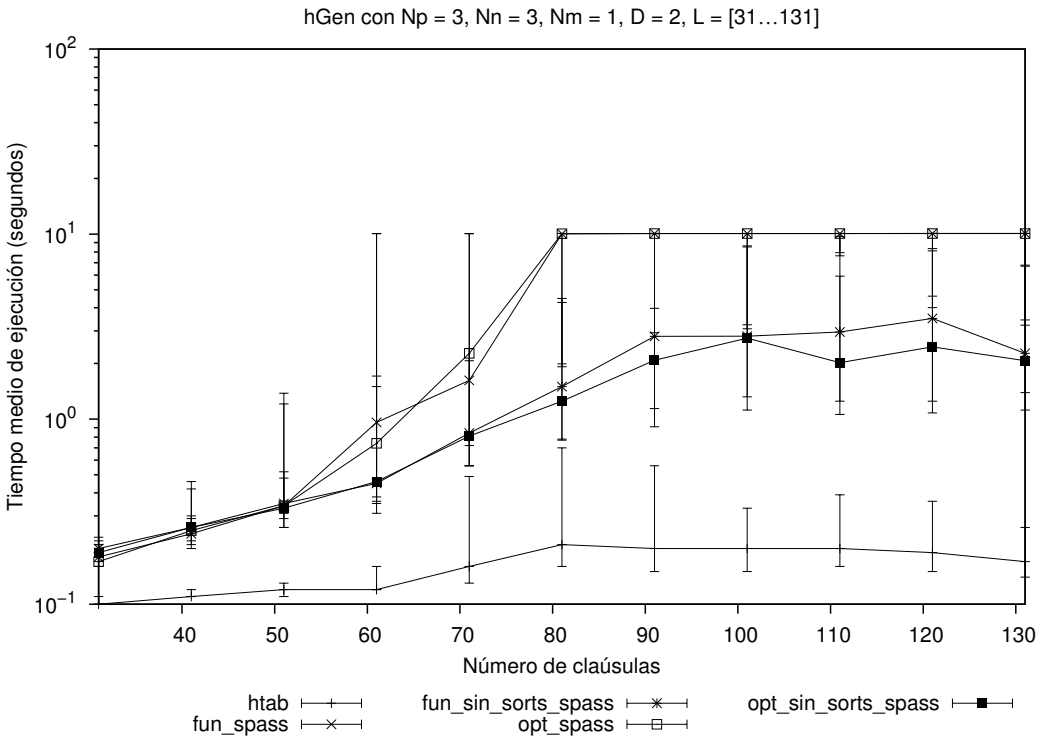


Figura 6.10: Tiempos de ejecución para fórmulas de $\mathcal{H}$ (20 % nominales)

Finalmente, en la figura 6.10 se observan los resultados al disminuir la cantidad de nominales a un 25 % (75 % proposiciones). Para $L = 80$ se registran los tiempos:

Traducción	Tiempo (segundos)
htab	0,21
fun_spass	>10
fun_sin_sorts_spass	1,5
opt_spass	>10
opt_sin_sorts_spass	1,25

Ahora se puede observar claramente que las traducciones funcionales sin sorts se comportan mejor que las que tienen sorts.

Por lo tanto, podemos concluir que: remover sorts en fórmulas de lenguaje $\mathcal{H}$ mejora el desempeño de las traducciones funcionales.

Sin embargo, agregar nominales, parece disminuir la dificultad de las fórmulas generadas aleatoriamente (con respecto a fórmulas de **K**). Más adelante daremos algunas explicaciones sobre las posibles causas de este fenómeno, que contradice lo que se esperaria de fórmulas pertenecientes a un lenguaje más expresivo.

6.6. Benchmarks para $\mathcal{H}(@)$

Ahora pasaremos a evaluar las fórmulas del lenguaje $\mathcal{H}(@)$. Para ello, configuramos hGen de modo que remplace, en un porcentaje de casos, el operador $\Box$ por el operador @. Probaremos para $p_@ = 25\,\%, 50\,\%$ y $75\,\%$, y cuando sea necesario, cambiaremos la profundidad modal para obtener tiempos más significativos ($D = 4, 6, 7$).

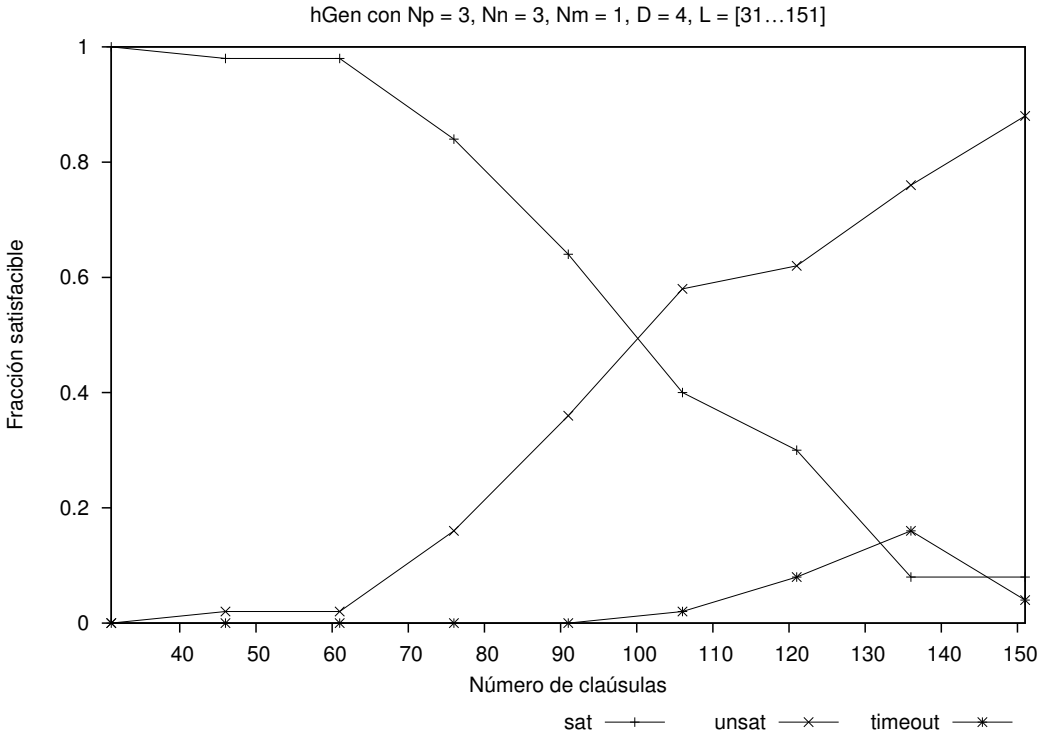


Figura 6.11: Transición de fase satisfacible/no satisfacible

En la figura 6.11 podemos observar que la transición de fase, utilizando los parámetros $p_@ = 25\,\%$ y $D = 4$, ocurre en $L = 91$ claúsulas aproximadamente.

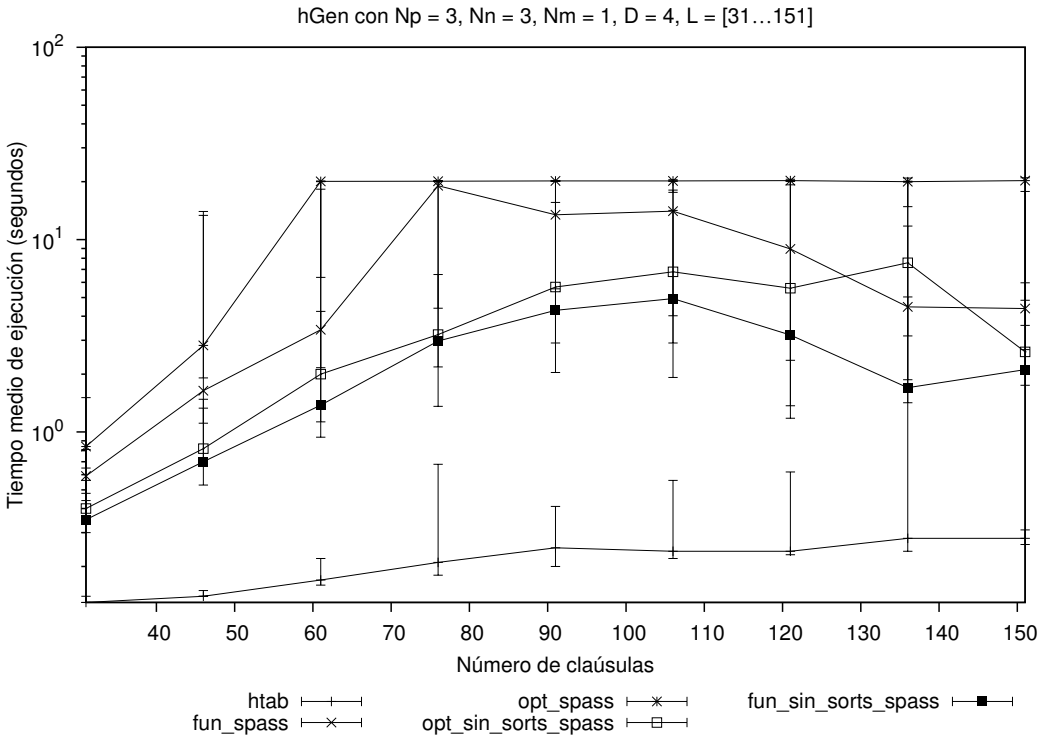


Figura 6.12: Tiempos de ejecución para fórmulas de $\mathcal{H}(@)$ (25 % @ y 12, 5 % nominales)

En la figura 6.12 se observan los tiempos de ejecución para dicha configuración. En $L = 91$ se registra:

Traducción	Tiempo (segundos)
htab	0,25
fun_spass	13,46
fun_sin_sorts_spass	4,29
opt_spass	>20
opt_sin_sorts_spass	5,67

Por lo cuál, aun con 25 % de operadores @ y 75 % de operadores $\Box$, tenemos una gran diferencia a favor de las traducciones funcionales sin sorts.

Si aumentamos la cantidad de operadores @ a un 50 % obtenemos las figuras 6.13 y 6.14. En la figura 6.13 podemos observar que la transición de fase ocurre en las $L = 71$ claúsulas aproximadamente. Los tiempos de ejecución para $L = 71$ son:

Traducción	Tiempo (segundos)
htab	0,18
fun_spass	0,82
fun_sin_sorts_spass	0,48
opt_spass	18,11
opt_sin_sorts_spass	0,92

Podemos ver que la diferencia de tiempos entre la traducción funcional y la traducción funcional sin sorts se volvió insignificante (0,34 segundos). Pero se mantiene una gran diferencia entre la traducción optimizada y la traducción optimizada sin sorts. Tanto las traducciones funcionales como HTab con las fórmulas modales mejoraron su desempeño con respecto al gráfico anterior. Por lo tanto, agregar operadores @ parece reducir la dificultad de la fórmula.

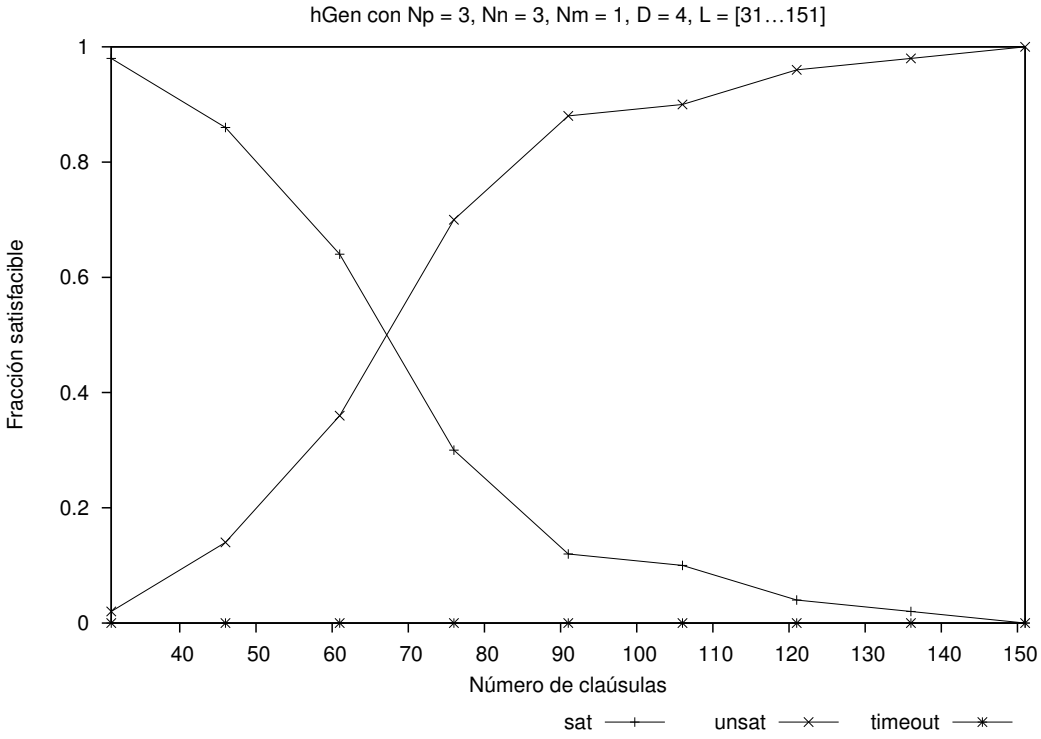


Figura 6.13: Transición de fase satisfacible/no satisfacible

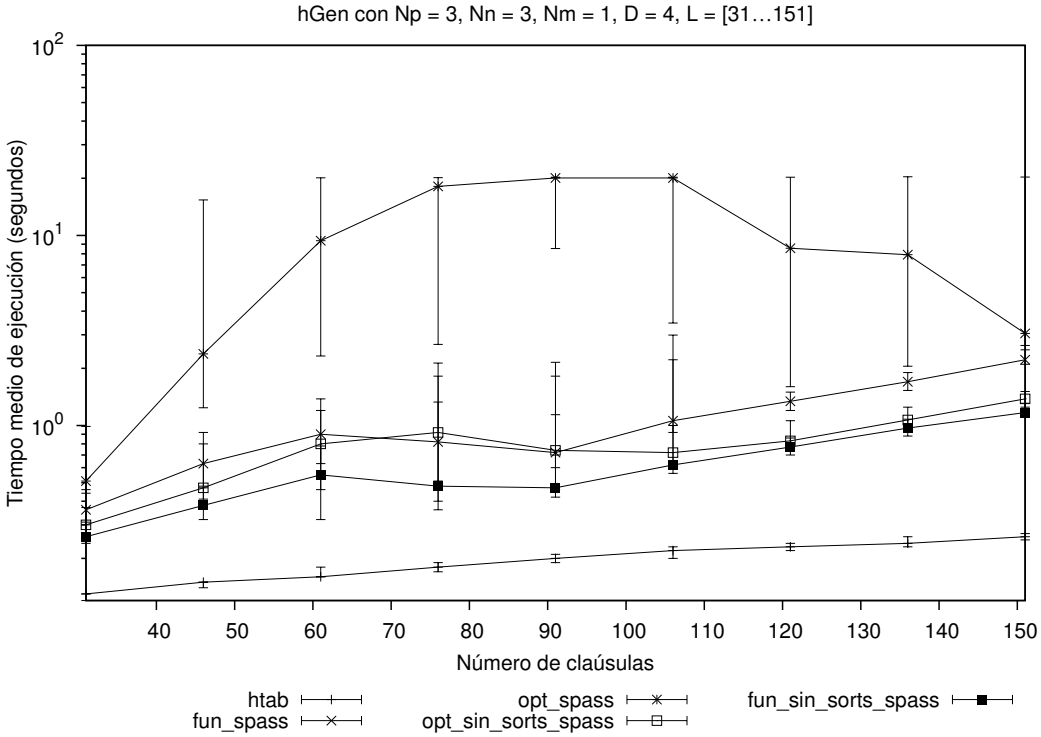


Figura 6.14: Tiempos de ejecución para fórmulas de $\mathcal{H}(@)$ (50 % @ y 12, 5 % nominales)

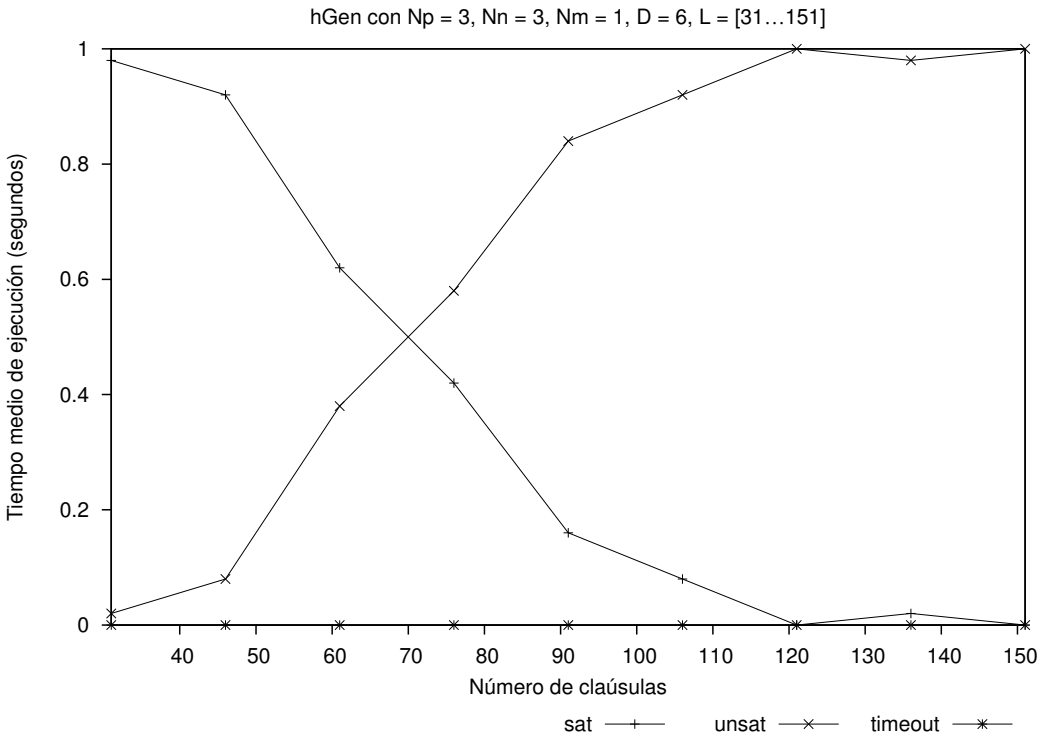


Figura 6.15: Transición de fase satisfacible/no satisfacible

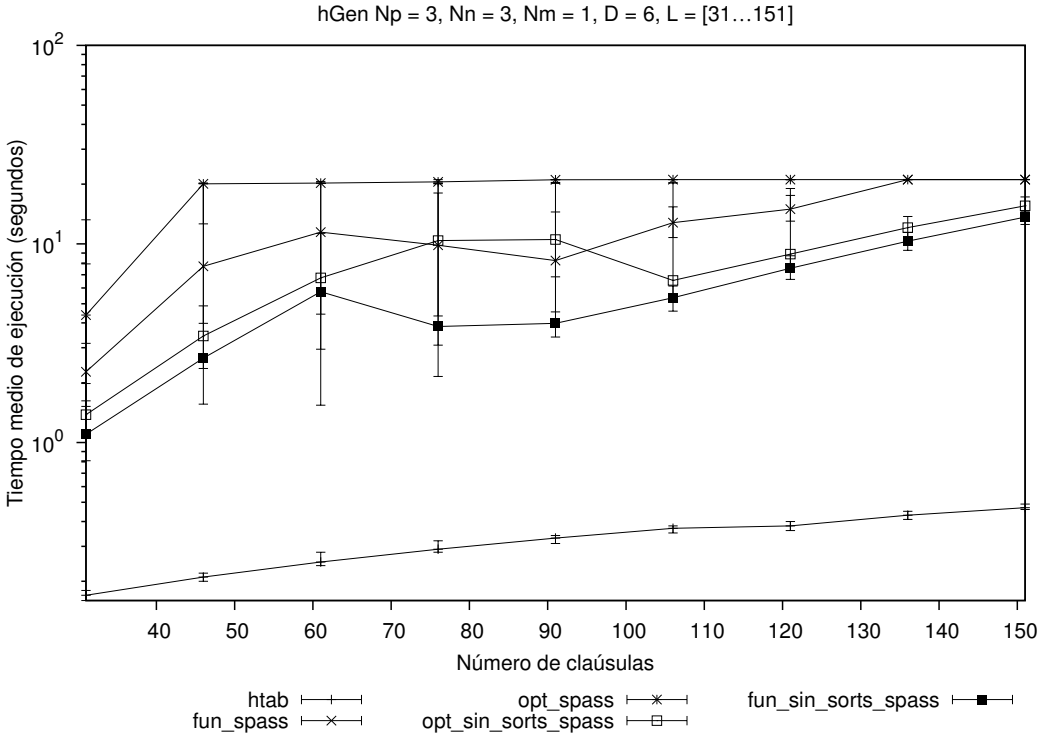


Figura 6.16: Tiempos de ejecución para fórmulas de $\mathcal{H}(@)$ (50 % @ y 12,5 % nominales)

Para saber si sigue existiendo una ventaja de la traducción funcional sin sorts sobre la traducción funcional cuando $p_@ = 50\,\%$, generamos el gráfico de la figura 6.16 que muestra los tiempos de ejecución para $D = 6$. En el se observan, para $L = 71$, los tiempos:

Traducción	Tiempo (segundos)
htab	0,29
fun_spass	9,85
fun_sin_sorts_spass	3,84
opt_spass	>20
opt_sin_sorts_spass	10,4

Al aumentar la profundidad modal, se incremento la dificultad del problema, y también se incremento la cantidad de operadores $\Box$ y @ de la fórmula (en la misma medida). Por lo que, al haber más $\Box$, se hizo más notorio el efecto de remover sorts; obteniendo una ventaja de 6 segundos.

Habiendo notado que remover sorts funciona positivamente en $\mathcal{H}(@)$, podemos hablar ahora sobre las razones por las cuales agregar nominales y/o operadores @ reduce la dificultad de las fórmulas. Como mencionamos en la sección anterior, esto es contrario a lo que debería ocurrir, ya que estamos incrementando la expresividad del lenguaje, y por lo tanto las fórmulas pueden ser más complejas.

En [Gorın, 2006] se ofrecen posibles causas para este fenómeno:

1. Si tomamos un conjunto de fórmulas modales **K** y reemplazamos símbolos proposicionales por nominales o reemplazamos operadores $\Box$ por operadores @, la cantidad de fórmulas no satisfacibles puede incrementarse, pero no decrecer. Por lo tanto, estariamos aumentando la cantidad de fórmulas no satisfacibles y es sabido que los probadores basados en resolución como SPASS se desempeñan mejor en ellas.

2. La presencia de operadores @, en reemplazo de operadores $\Box$, reduce la cantidad de unificaciones redundantes.

3. La profundidad modal es aceptada como medida de complejidad en los lenguajes modales. Sin embargo, en el caso híbrido, es menos fiable. Por ejemplo, es posible construir fórmulas semánticamente equivalentes, pero cuyas profundidades modales difieran tanto como queramos.

6.7. Benchmarks para $\mathcal{H}(\downarrow)$

Para $L = 61$ tenemos los tiempos:

Traducción	Tiempo (segundos)
htab	0,15
fun_spass	>20
fun_sin_sorts_spass	8,2
opt_spass	>20
opt_sin_sorts_spass	>20

Podemos observar, como lo venimos haciendo, una gran ventaja por parte de las traducciones sin sorts.

Sin embargo, se observa una gran varianza en todo el gráfico. Esto puede deberse a un bug que encontramos en hGen, por el cuál se generan muchas subfórmulas triviales, por ejemplo $\downarrow N1\ .\ (N1 \vee P1 \vee P2)$. Los desarrolladores de hGen se encuentran en el momento de la escritura de este trabajo resolviendo los problemas con la generación para el caso $\mathcal{H}(\downarrow)$ por lo que no se han realizado más experimentos en esta lógica por el momento.

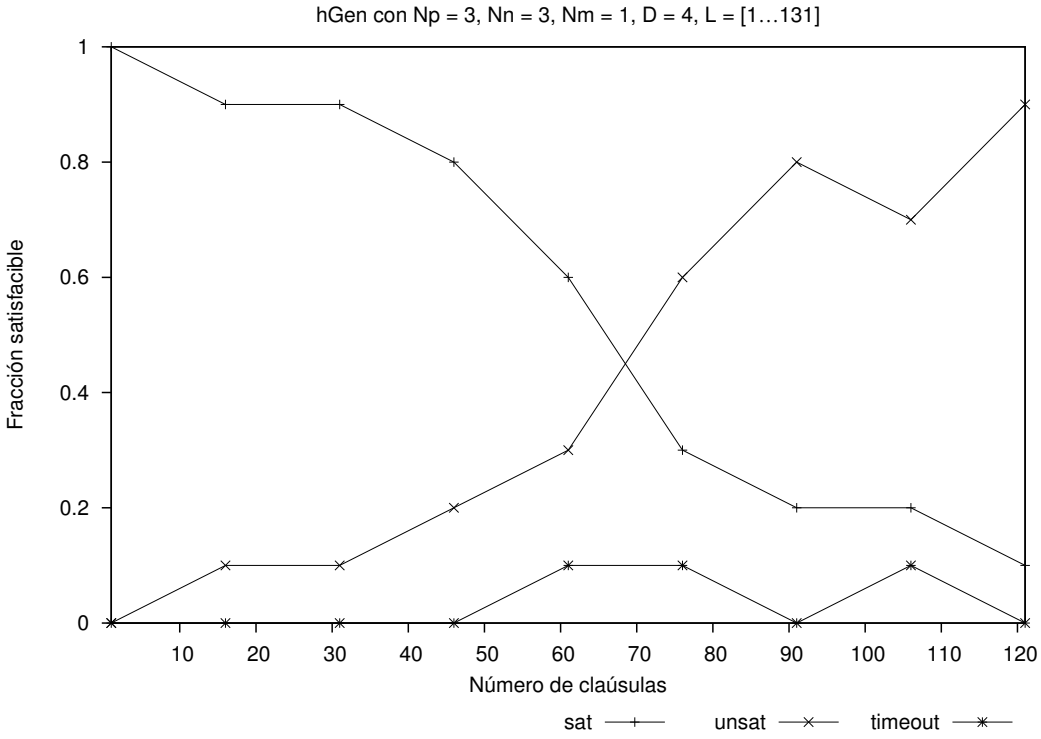


Figura 6.17: Transición de fase satisfacible/no satisfacible

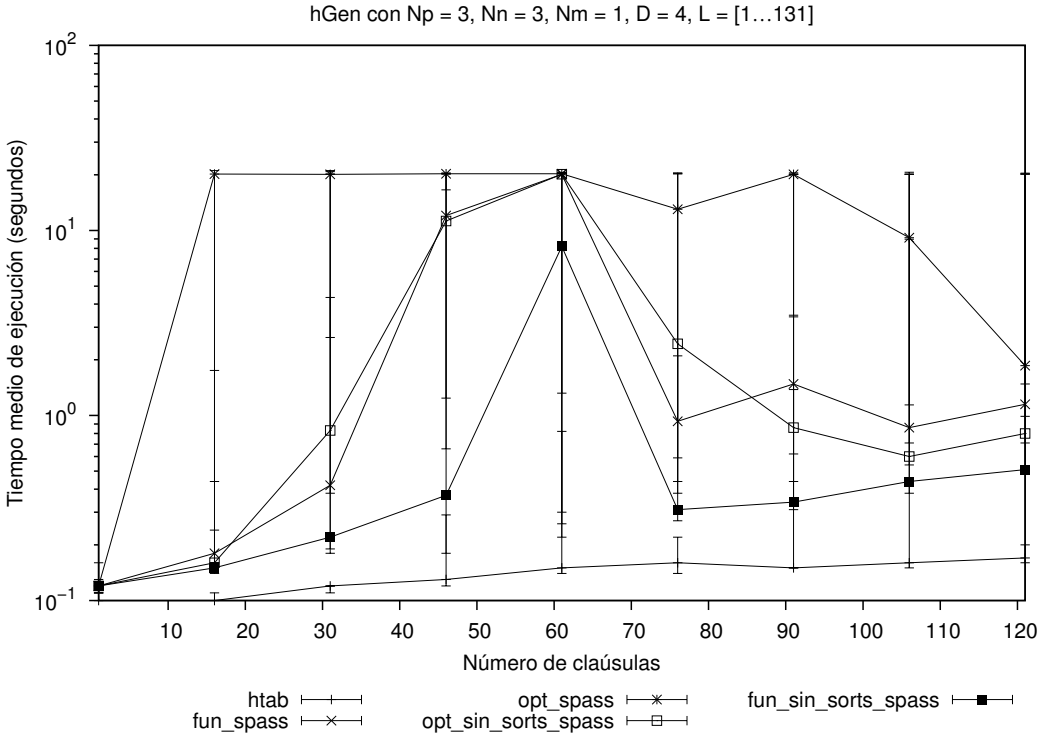


Figura 6.18: Tiempos de ejecución para fórmulas de $\mathcal{H}(\downarrow)$ (10 % $\downarrow$, 10 % nominales)

6.8. Conclusión

En éste capítulo hemos investigado el comportamiento de las traducciones en distintos fragmentos del lenguaje híbrido $\mathcal{H}(@, \downarrow)$.

Observamos que las traducciones sin sorts tienen un desempeño mejor que las traducciones con sorts. Además notamos que agregar nominales y operadores @ a la fórmula la hace más fácil, lo cual sugeriria que el método de testeo, usando fórmulas aleatorias, no es muy adecuado para la lógicas híbridas.

Capítulo 7

Evaluando las traducciones funcionales en KT y K4

7.1. Introducción

En capítulos anteriores evaluamos los demostradores al resolver el problema de validez de una fórmula sobre la clase de todos los frames. Sin embargo, remover sorts de las traducciones funcionales también es correcto para otras clases de frames.

En el Capítulo 3 enunciamos el Teorema 5, el cuál nos asegura que remover sorts también preserva satisfacibilidad en frames definidos modalmente.

Este resultado es muy útil, ya que diferentes aplicaciones de la lógica modal típicamente validan diferentes axiomas modales, aparte de los del sistema mínimo **K**. Por ejemplo, si miramos modelos como flujos de tiempo, es natural asumir que la relación de accesibilidad es transitiva; y, cualquier instancia del esquema $\Box\varphi \rightarrow \Box\Box\varphi$ es válida en la clase de los frames transitivos. Sin embargo, ninguna instancia de éste esquema (llamado **4**, por razones históricas) es demostrable en **K**. Por lo tanto, si queremos una lógica para trabajar con flujos temporales, debemos agregar todas sus instancias como axiomas extra, obteniendo como resultado la lógica conocida como **K4**. Más aún, una fórmula es demostrable en la lógica **K4** si y sólo si es satisfecha en todos los modelos basados en frames cuya relación de accesibilidad es transitiva.

Lo mismo ocurre con otros axiomas como $\Box\varphi \rightarrow \varphi$ (axioma **T**) que, al agregarlo a **K** (lógica **KT**), caracteriza los frames con relación de accesibilidad reflexiva.

El hecho de que los axiomas modales reflejen propiedades de las relaciones de accesibilidad es una de las características más atractivas de la lógica modal.

En este capítulo investigaremos los efectos de la eliminación de sorts en el desempeño de SPASS para modelos con relaciones reflexivas o transitivas. Para ello, tomaremos la conjunción de las fórmulas traducidas funcionalmente y una fórmula φ en lógica de primer orden (de la signatura adecuada), que obligará a la relación a ser reflexiva o transitiva, según sea el caso.

7.2. Experimento en KT

Para evaluar el desempeño de las traducciones funcionales sobre frames reflexivos, utilizamos la fórmula $\forall x.\ r(x,x)$. Pero primero usamos la ecuación 3.3 para reescribir la relación al lenguaje funcional, obteniendo: $\forall x{:}\omega.\neg de_r(x) \wedge \exists z{:}\iota.f_r(z,x) = x$, fórmula que añadiremos al conjunto de claúsulas. Además, como se menciona en [Ohlbach and Schmidt, 1997], debemos añadir la declaración de sorts siguiente: $f : \iota \times \omega \rightarrow \omega$. De otro modo, el demostrador no reconocerá a $f(x_2, x_1)$ como elemento de ω, y $\neg de_r(f(x_2, x_1))$ no será válida al no poder aplicarse el axioma de reflexividad.

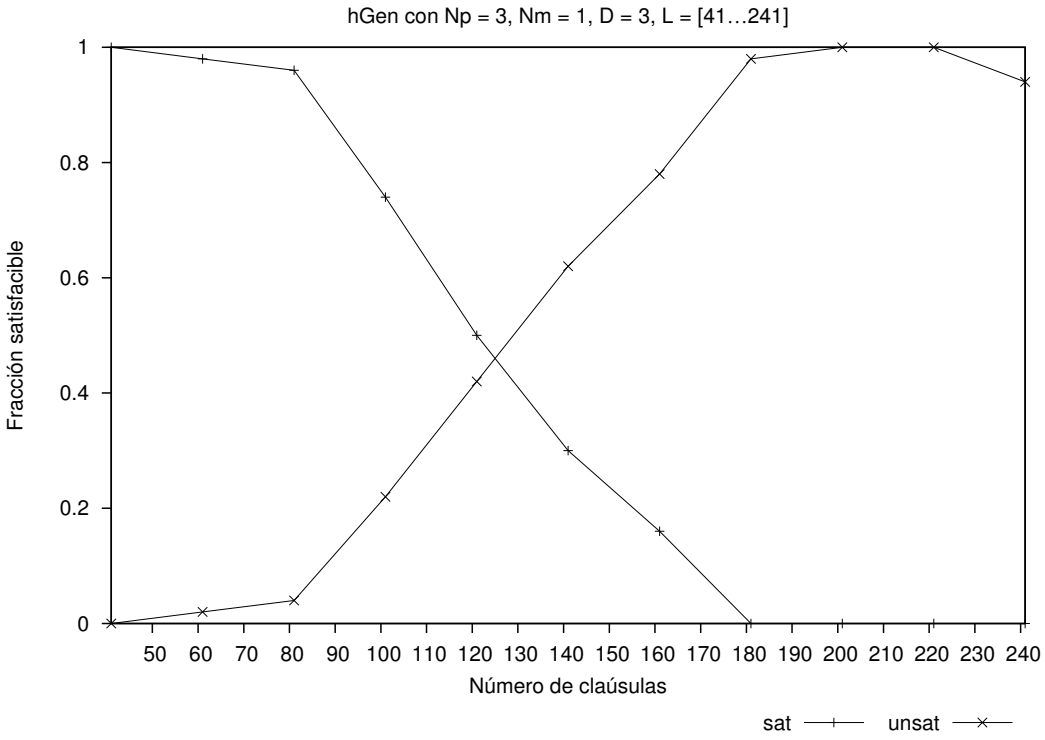


Figura 7.1: Transición de fase satisfacible/no satisfacible

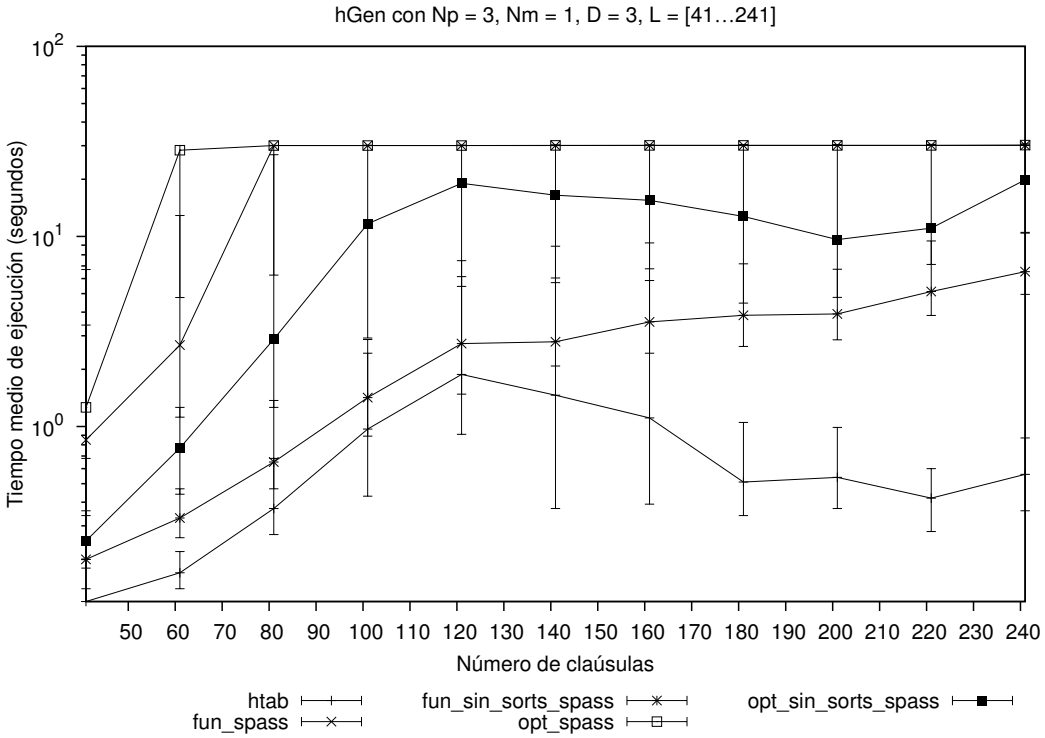


Figura 7.2: Tiempos de ejecución para fórmulas aleatorias de **KT** con $D = 3$

En la Figura 7.2 se observan los tiempos de ejecución de SPASS y HTab en la lógica **KT**, para fórmulas generadas aleatoriamente. El objetivo de este test, al igual que los anteriores, es observar el efecto que tiene el borrado de sorts de las traducciones funcionales en el desempeño

de SPASS, pero esta vez sobre frames reflexivos.

En el gráfico se observan para $L = 121$ los siguientes tiempos medios:

Traducción	Tiempo (segundos)
htab	1,88
fun_spass	>30
fun_sin_sorts_spass	2,73
opt_spass	>30
opt_sin_sorts_spass	19,04

Las traducciones funcionales con sorts hacen que SPASS exceda el tiempo máximo de 30 segundos, pero no esto no sucede para las traducciones sin sorts. Más aún, podemos notar que la traducción funcional sin sorts demora menos del 10 % que la traducción funcional con sorts.

Por lo tanto, podemos concluir que remover sorts mejora el desempeño de SPASS incluso cuando tratamos con frames reflexivos.

7.3. Experimento en K4

Ahora realizaremos el mismo experimento para frames transitivos. La relación debe satisfacer $\forall xyz.\ (r(x,y) \wedge r(y,z) \rightarrow r(x,z))$, por lo cual agregamos la fórmula $\forall x{:}\omega.(\neg de_r(x) \rightarrow \forall ab{:}\iota\exists c{:}\iota.f_r(b, f_r(a,x)) = f_r(c,x))$ a las claúsulas.

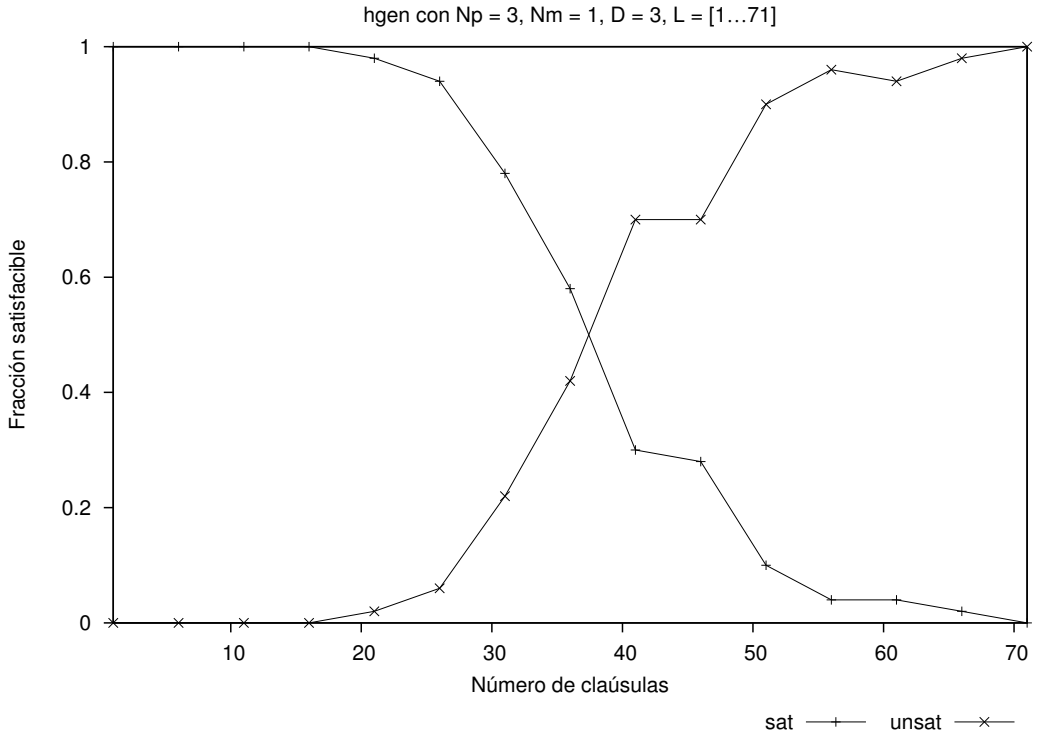


Figura 7.3: Transición de fase satisfacible/no satisfacible

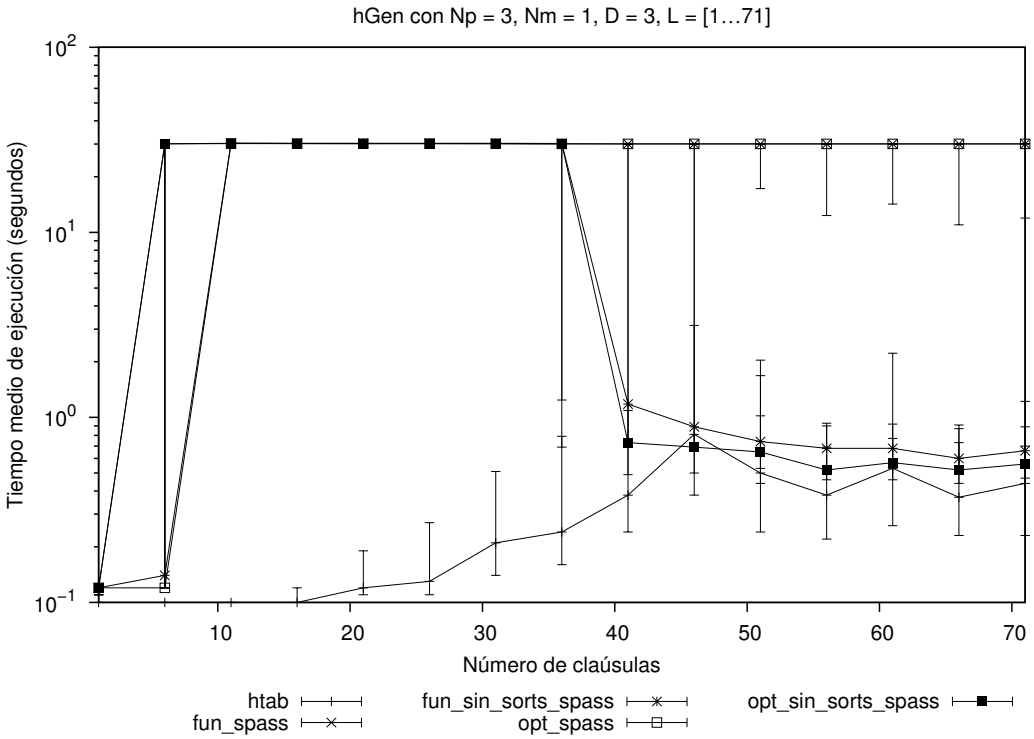


Figura 7.4: Tiempos de ejecución para fórmulas aleatorias de **K4** con $D = 3$

En la Figura 7.4 se observan los tiempos de ejecución de los demostradores para la lógica **K4**. Para $L = 41$ tenemos:

Traducción	Tiempo (segundos)
htab	0,38
fun_spass	>30
fun_sin_sorts_spass	1,18
opt_spass	>30
opt_sin_sorts_spass	0,73

Podemos notar que remover sorts de las traducciones produce una gran aceleración en SPASS para las fórmulas no satisfacibles, acercandose al desempeño de HTab. Seria muy útil obtener una imagen completa del desempeño de las traducciones funcionales con sorts para saber con precisión cuál es la ganancia al removerlos. Pero decidir satisfacibilidad en frames transitivos es difícil: el test de la Figura 7.4 demoró un día entero en realizarse. Por este motivo dejamos para otra ocasión una evaluación más exhaustiva en **K4**.

Capítulo 8

Conclusión

En esta tesis hemos investigado el desempeño del demostrador de lógica de primer orden SPASS sobre las traducciones funcionales para lógicas modales.

Implementamos en haskell varias traducciones funcionales: traducción funcional, traducción funcional sin sorts, traducción funcional optimizada y traducción funcional optimizada sin sorts (junto a otras transformaciones necesarias).

Luego testeamos nuestras implementaciones, junto a otras traducciones funcionales ya implementadas en SPASS, sobre el lenguaje modal **K**, usando clases de fórmulas creadas manualmente. También testeamos las traducciones mediante fórmulas generadas aleatoriamente, pertenecientes a las lógicas: **K**, $\mathcal{H}$, $\mathcal{H}(@)$ y $\mathcal{H}(\downarrow)$. Finalmente en el Capítulo 7 realizamos test sobre fórmulas en **K4** y **KT**.

Al analizar todos los resultados obtenidos, pudimos concluir que remover las anotaciones de sorts (de las traducciones) produce efectivamente una mejora significativa en el desempeño de SPASS.

Además, comprendimos la dificultad del problema de evaluar métodos de decisión para lógica modal: por un lado existen pocos tests derivados de aplicaciones reales y, por el otro, utilizando generadores aleatorios resulta difícil abarcar el amplio espectro de fórmulas híbridas. Por lo tanto sugerimos, como trabajo futuro, la creación de fórmulas híbridas que cumplan las propiedades del benchmark del Capítulo 5.

También queda como trabajo futuro la realización de más tests en la lógica $\mathcal{H}(\downarrow)$ y en lógicas modales con distintos frames. En el primer caso, no se pudo obtener resultados significativos debido a un bug en hGen. Este y otros bugs encontrados en HTab y el parser de lógica híbrida fueron reportados a sus respectivos desarrolladores.

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 Gorín, 2011] C. Areces and D Gorín. Unsorted functional translations. *Electronic Notes in Theoretical Computer Science*, 278(0):3–16, 2011. Proceedings of the 7th Workshop on Methods for Modalities (M4M 2011).

[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.*, 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, 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 and Enjalbert, 1989] Y. Auffray and P. Enjalbert. Modal theorem proving: An equational viewpoint. In *Proc. of the 11th IJCAI*, pages 441–445. Morgan Kaufmann Pub., 1989.

[Auffray and Enjalbert, 1992] Y. Auffray and P. Enjalbert. Modal theorem proving: An equational viewpoint. *J. of Logic and Computation*, 2(3):247–295, 1992.

[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.

[Blackburn *et al.*, 2001] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Scie*. Cambridge University Press, Cambridge, 2001.

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

[Fariñas del Cerro and Herzig, 1988] L. Fariñas del Cerro and A. Herzig. Linear modal deductions. In *Proc. of CADE-9*, volume 310 of *LNCS*, pages 487–499. Springer, 1988.

[Gent and Walsh, 1994] Ian P. Gent and Toby Walsh. The sat phase transition. pages 105–109. John Wiley and Sons, 1994.

[Gorın, 2006] Daniel Gorın. Hybrid layering. In *ESSLLI Student Session*, page 99. Citeseer, 2006.

[Halpern and Moses, 1992] Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319 – 379, 1992.

[Harel *et al.*, 1984] David Harel, Dexter Kozen, and Jerzy Tiuryn. Dynamic logic. In *Handbook of Philosophical Logic*, pages 497–604. MIT Press, 1984.

[Heuerding and Schwendimann, 1996] Alain Heuerding and Stefan Schwendimann. A benchmark method for the propositional modal logics k, kt, s4, 1996.

[Horrocks *et al.*, 2006] I. Horrocks, U. Hustadt, U. Sattler, and R. Schmidt. Computational modal logic. In *Handbook of Modal Logics*, pages 181–245. Elsevier, 2006.

[Horrocks, 2000] Ian Horrocks. R.: An analysis of empirical testing for modal decision procedures. *Logic Journal of the IGPL*, pages 293–323, 2000.

[Hustadt and Schmidt, 1997] Ullrich Hustadt and Renate A. Schmidt. On evaluating decision procedures for modal logic. pages 202–207. Morgan Kaufmann, 1997.

[Hustadt and Schmidt, 1999] U. Hustadt and R. A. Schmidt. An empirical analysis of modal theorem provers. *J. of Applied Non-Classical Logics*, 9(4):479–522, 1999.

[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.

[Ohlbach and Schmidt, 1997] H. Ohlbach and R. Schmidt. Functional translation and second-order frame properties of modal logics. *J. of Logic and Computation*, 7(5):581–603, 1997.

[Ohlbach, 1988a] H. Ohlbach. *A Resolution Calculus for Modal Logics*. PhD thesis, Universität Kaiserslautern, 1988.

[Ohlbach, 1988b] H. Ohlbach. A resolution calculus for modal logics. In *Proc. of CADE-9*, volume 310 of *LNCS*, pages 500–516. Springer, 1988.

[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.

[Patel-Schneider *et al.*, 2003] Peter Patel-Schneider, Peter F. Patel-schneider, Roberto Sebastiani, and Roberto Sebastiani. A new general method to generate random modal formulae for testing decision procedures. *Journal of Artificial Intelligence Research*, 18:2003, 2003.

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

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

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

[Schmidt, 1997] R. Schmidt. *Optimised Modal Translation and Resolution.* PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 1997.

[Schmidt, 1999] R. Schmidt. Decidability by resolution for propositional modal logics. *J. of Automated Reasoning*, 22(4):379–396, 1999.

[Walther, 1989] C. Walther. Many-sorted inferences in automated theorem proving. In *Sorts and Types in Artificial Intelligence*, volume 418 of *LNCS*, pages 18–48. Springer, 1989.

[Weidenbach *et al.*, 2007] C. Weidenbach, R. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. System description: Spass version 3.0. In *Proc. of CADE-21*, number 4603 in LNAI, pages 514–520. Springer-Verlag, 2007.

[Weidenbach, 2001] C. Weidenbach. Combining superposition, sorts and splitting. In *Handbook of Automated Reasoning*, volume 2, chapter 27, pages 1965–2014. Elsevier and MIT Press, 2001.

[Zamov, 1989] N. Zamov. Modal resolutions. *Soviet Math*, 33(9):23–29, 1989.