BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS
POSGRADO EN CIENCIAS MATEMÁTICAS

“LÓGICAS DESCRIPTIVAS”

TESIS

QUE PARA OBTENER EL GRADO DE

DOCTOR EN CIENCIAS MATEMÁTICAS

PRESENTA:
JUAN PABLO MUÑOZ TORIZ

ASESOR(ES):
DR. JOSÉ RAMÓN ENRIQUE ARRAZOLA RAMÍREZ
DR. IVÁN MARTÍNEZ RUIZ

ENERO 2017

Benemérita Universidad Autónoma de Puebla

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS
POSGRADO EN CIENCIAS MATEMÁTICAS

LÓGICAS DESCRIPTIVAS

TESIS

QUE PARA OBTENER EL GRADO DE
DOCTOR EN CIENCIAS MATEMÁTICAS

PRESENTA
JUAN PABLO MUÑOZ TORIZ

DIRECTORES DE TESIS
DR. JOSÉ RAMÓN ENRIQUE ARRAZOLA RAMÍREZ
DR. IVÁN MARTÍNEZ RUIZ

PUEBLA, PUEBLA. 2017.

BUAP

DRA. LIDIA AURORA HERNÁNDEZ REBOLLAR
SECRETARIA DE INVESTIGACIÓN Y
ESTUDIOS DE POSTGRADO, FCFM-BUAP
P R E S E N T E:

Por este medio le informo que el(la) C:

JUAN PABLO MUÑOZ TORIZ

estudiante del Doctorado en Ciencias (Matemáticas), ha cumplido con las indicaciones que el Jurado le señaló en el Coloquio que se realizó el día 8 de diciembre de 2016, con la tesis titulada:

"LÓGICAS DESCRIPTIVAS"

Por lo que se le autoriza a proceder con los trámites y realizar el examen de grado en la fecha que se le asigne.

A T E N T A M E N T E.
H. Puebla de Z, a 11 de enero de 2017

DR. FERNANDO MACÍAS ROMERO
COORDINADOR DEL POSTGRADO
EN MATEMÁTICAS.

DR FMR/mtrv

60 AÑOS DE AUTONOMÍA UNIVERSITARIA

Facultad de Ciencias Físico Matemáticas
Av. San Claudio y 18 sur, edif. 111 A, Ciudad Universitaria, Col. San Manuel, Puebla, Pue. C.P. 72570
01 (222) 229 55 00 Ext. 7550 y 7552

Dedicatoria

A mi familia especialmente a mi esposa e hijos, que son una de mis principales motivaciones para mejorar.

A mis profesores (Jesús Lavalle, Everardo Bárcenas, Iván Martínez, José Arrazola y todos los que antecedieron) que han contribuido de manera muy significativa en mi formación académica.

A mis compañeros y amigos {Javier, Miguel, Luis Angel, Ruben, Reinaldo, Alejandro, Armando, Verónica,...} que son un conjunto numerable y con cada uno de ellos sé que puedo contar.

Finalmente a Dios, porque tengo que culpar y agradecer a alguien todas esas cosas que no he tenido la libertad de elegir.

Agradecimiento al CONACYT

Agradezco de manera especial al Consejo Nacional de Ciencia y Tecnología (CONACYT) por la beca (No. de apoyo 329035) que me fue otorgada para realizar mis estudios de Doctorado, sin la cual me hubiera sido más difícil tal logro.

Introducción

Motivación

En nuestro proyecto de Maestría "Demostración automática de Teoremas" se crearon demostradores automáticos de fórmulas para lógica clásica, lógica intuicionista y algunas otras lógicas no clásicas. En nuestro afán por extender nuestra automatización a la lógica de primer orden (POL), o al menos a la parte decidible de POL, nos introducimos al estudio de sistemas lógicos más expresivos. Lo anterior nos condujo a las lógicas descriptivas que en principio son variantes notacionales de extenciones de fragmentos decidibles de POL.

Contribución y contenido

Nuestros estudios estuvieron orientados a la caracterización de propiedades de las lógicas descriptivas para su implementación en algoritmos de decibilidad y la automatización de dichos algoritmos. En este sentido cubrimos los dos aspectos importantes de la investigación, la parte teórica y la parte práctica.

En nuestro trabajo caracterizamos propiedades de los roles descriptivos en termino de la satisfacción de conceptos, lo que permitió la creación de sistemas de secuentes e hipersecuentes para algunas lógicas descriptivas, dichos sistemas fueron implementados en ML. Aunado a lo anterior, se creo una interfaz gráfica en JAVA que permite la demostración automática de teoremas en las lógicas aquí expuestas y algunas otras lógicas importantes.

Estructura de la Tesis

El presente trabajo está estructurado de la siguiente manera:

En el primer capítulo se estudia el lenguaje conceptual básico $\mathcal{ALC}$ introducido por Schmidt-Schauβ and Smolka [18] así como el estado del arte del tema de investigación.

En el segundo capítulo se caracterizan propiedades de los roles y se da una restricción de $\mathcal{ALC}$ con reflexividad y transitividad en los roles, la cual llamamos $ALCS4$ dado que ésta corresponde a la lógica multimodal $S4$.

En el Tercer capítulo se estudia la noción de secuentes e hiperseceuntes y se dan caracterizaciones que permiten proponer los correspondientes sistemas para determinar la satisfacción de conceptos en las lógicas descriptivas estudiadas, de la misma forma se prueba que los sistemas son robustos y completos.

En las Conclusiones se presenta un resumen de los resultados de este trabajo y una idea de investigaciones futuras.

Finalmente el apéndice se presenta la automatización de los algoritmos en el lenguaje de programación ML.

Índice general

Introducción I

1. Preliminares 1
1.1. Estado del arte 1
1.2. Las Lógicas Descriptivas 2
1.2.1. Una axiomatización de $\mathcal{ALC}$ 4
1.3. El problema de la satisfacción 5

2. Una restricción de $\mathcal{ALC}$ 7
2.1. $ALCS4$ 10

3. Sistemas de Secuentes Lógico Descriptivos 13
3.1. Secuentes para ALCS4 13
3.1.1. Correctitud 15
3.1.2. Complejidad e implementación 19
3.2. Árboles de hipersecuentes para $\mathcal{ALC}$ 20
3.2.1. El cálculo 20
3.2.2. Correctitud 22
3.2.3. Complejidad e implementación 25

Conclusión 27

A. Implementación en ML 29
A.1. Implementación del sistema de secuentes para $ALCS4$ . . 30
A.2. Sistema de árboles de hipersecuentes para $\mathcal{ALC}$ 36

Bibliografía 46

Índice alfabético 48

Lógicas Descriptivas

Juan Pablo Muñoz Toriz

Enero de 2017

Capítulo 1

Preliminares

1.1. Estado del arte

El problema de formalizar el conocimiento se presenta en el campo de la inteligencia artificial, donde se trata de utilizar conocimiento en cierto contexto para hacer inferencia sobre dicho conocimiento. Este es el caso del proyecto de la Web semántica [6] propuesta por Berners Lee y otros en el 2001, que vendrá a ser una extensión de la Web actual, la cual se basa en la idea de añadir metadatos semánticos y ontológicos a la World Wide Web. Esa información adicional se debe proporcionar de manera formal, para que sea posible ser evaluada automáticamente por máquinas de procesamiento.

Para exportar la Web semántica se necesita un lenguaje común basado en web, con suficiente capacidad expresiva y de razonamiento para representar la semántica de las ontologías. De esta forma, la utilización de lenguajes tales como Web Ontology Language (OWL)y OWL Lite son una paso más en la obtención de la Web Semántica. Las semánticas de estos lenguajes están basadas en ciertas lógicas denominadas lógicas descriptivas (DL) [11].

Las lógicas descriptivas (DL) son una familia de lógicas que son fragmentos decidibles de la lógica de primer orden con propiedades computacionales atractivas. Estas lógicas surgieron como una extensión de los frames (marcos)[12] propuestos por Minsky en 1974 y redes semánticas [19] presentadas por Sowa en 1987, los cuales no estaban equipados con semánticas basadas en la lógica. El nombre de Description logics (DL) se les dio en los años 1980s. Antes de esto se llamaban (cronológicamente): sistemas terminológicos, y lenguajes conceptuales.

Los ingredientes esenciales de las DL son los individuos (elementos), conceptos (conjuntos) y los roles (relaciones entre elementos).

Las DL se caracterizan por su semántica basada en estructuras de tipo Kripke.

Un problema importante acerca de las DL es decidir cuándo un concepto se satisface o cuándo un concepto se subsume (es menos general o es sumergido) en otro, lo cual nos indica conocimiento de manera general. Es aquí donde los algoritmos de satisfacción de conceptos hacen su aparición.

Dado que algunas DL son variantes notacionales de lógicas modales , resulta que los sistemas de satisfacción de fórmulas en las lógicas modales pueden ser adecuados a sus variantes DL.

Muchos de los importantes sistemas en lógica proposicional modal tienen una formulación de tipo secuentes de Gentzen [8]. Sin embargo algunos de estos sistemas no son adecuados computacionalmente hablando, ya que no cumplen propiedades deseables para ser implementados (tal es el caso de la regla de corte). De tal forma que Arnon Avron en 1994 [1] introduce el marco de hipersecuentes, que es una generalización de los secuentes de Gentzen, para dotar a los secuentes de propiedades adecuadas para su implementación.

Cuando se habla de DL expresivas se introducen restricciones (de número, de número calificadas y generalizadas) respecto al número de hijos en los roles. Es aquí cuando los algoritmos tableau basados hacen su aparición. En 2001 Stephan Tobies [21] presenta un algoritmo tableau basado para la lógica modal graduada (variante notacional de la lógica $\mathcal{ALCQ}$) de tiempo PSpace que es la menor cota conocida.

En el 2009 F. Poggiolesi [14] presenta un calculo de hipersecuentes para lógicas modales, en particular para la lógica modal K.

En 2010 B. Hill [10] da un cálculo de hipersecuente libre de la regla de corte para la lógica proposicional dinámica.

En 2014 Sun y Sui [20], presenta un sistema de secuentes de Gentzen para la $\mathcal{ALC}$ el cual utiliza la regla de corte y contracción.

A diferencia de [14], nuestro cálculo de hipersecuentes permite multiples roles además de ser libre de reglas de corte y contracción en contraste con [20]. Otras contribuciones importantes son la adecuación semántica de los sistemas de secuentes e hipersecuentes a las DL y la implementación computacional de los algoritmos de satisfacción de conceptos, lo cual lo hacen un trabajo original.

Una fuente de referencia e información adicional sobre las lógicas descriptivas es [2].

1.2. Las Lógicas Descriptivas

Las Lógicas Descriptivas (DL por sus siglas en inglés) son un conjunto de sistemas de representación del conocimiento basados en sistemas lógi-

cos. Surgieron como sistemas puramente semánticos e incluso algunas de ellas son variantes notacionales de lógicas modales, lo cual permite que varias de estas lógicas admitan también una axiomatización adecuada.

Definición 1.1. *Sea N_C un conjunto de nombres de conceptos y sea N_R el conjunto de nombres de roles. El conjunto de conceptos de $\mathcal{ALC}$ es el conjunto más pequeño tal que*

- *Todo nombre de concepto es un concepto y*
- *Si C y D son conceptos y R es un nombre de rol, entonces $(C \sqcap D)$, $\neg C$ y $(\forall R.C)$ son conceptos.*

Los nombres de conceptos son interpretados en estructuras relacionales como conjuntos; el nombre A es utilizado como etiqueta; el nombre $\neg C$ es interpretado como el conjunto complemento; la semánticas para la conjunción $C \sqcap D$ corresponde al conjunto intersección; y $\forall R.C$ corresponde al conjunto de elementos tales que todos sus elementos accesibles mediante el rol R satisfacen C.

Consideramos también la siguiente notación: $C \sqcup D := \neg(\neg C \sqcap \neg D)$, $\bot := \neg A \sqcap A$, $\top := \neg\bot$, $C \sqsubseteq D := \neg C \sqcup D$, $C \equiv D := (C \sqsubseteq D) \sqcap (D \subseteq C)$ y $\exists R.C := \neg\forall R.(\neg C)$.

Considere por ejemplo el siguiente concepto descriptivo presentado en [2]:

$Man \sqcap \exists married.Doctor \sqcap \forall child.Happy$

Este concepto descriptivo corresponde a los hombres casados con un doctor y que todos sus hijos son felices.

Antes de dar una descripción formal de la semántica de ALC introduciremos la noción de interpretación, la cual es una pareja $I = (\Delta^I, \cdot^I)$, donde Δ^I es un conjunto no vacío llamado el dominio de I, y $\cdot^I$ es una función de interpretación, que asigna a cada nombre de concepto A un subconjunto del dominio $A^I \subseteq \Delta^I$ y a cada nombre de rol R una relación binaria en el dominio $R^I \subseteq \Delta^I \times \Delta^I$.

Definición 1.2 (Semantica)**.** *Dada una interpretación $I = (\Delta^I, \cdot^I)$, una semántica de conceptos descriptivos es definida como sigue:*

$$\begin{aligned}(\neg C)^I &= \Delta^I \setminus C^I \\ (C \sqcap D)^I &= C^I \cap D^I \\ (\forall R.C)^I &= \left\{ d \in \Delta^I \mid \forall e.(d,e) \in R^I \rightarrow e \in C^I \right\}\end{aligned}$$

Una interpretación I es un modelo de un concepto descriptivo C cuando $C^I \neq \emptyset$, lo cual se denota de la forma $I \models C$. En este caso decimos que C es satisfecho por I.

Si toda interpretación es un modelo de C, entonces decimos que C es válido y escribimos $\models C$.

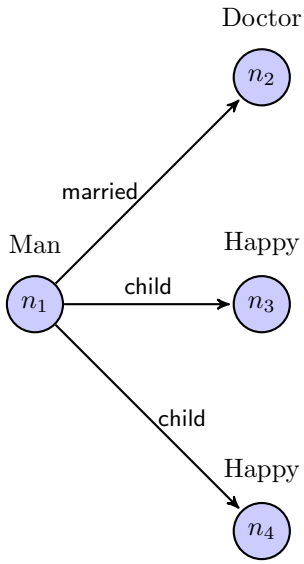

Figura 1.1: Un modelo relacional para Man $\sqcap$ $\exists$married.Doctor $\sqcap$ $\forall$child.Happy cumpliéndose para n_1.

1.2.1. Una axiomatización de $\mathcal{ALC}$

Es bien sabido que $\mathcal{ALC}$ es una variante notacional de la lógica multimodal K [17], por lo tanto una axiomatización para K es también una axiomatización para $\mathcal{ALC}$, la cual se compone por los axiomas de lógica clásica junto con el esquema de normalidad y las reglas de Modus Ponens y Necessitation.

Ahora daremos una axiomatización de $\mathcal{ALC}$, como la presentada en [15].

Definición 1.3 (Axiomatización de $\mathcal{ALC}$)**.** *Los axiomas están determinados de la siguiente forma:*

$A1$ $\neg(C \sqcap (D \sqcap \neg C))$

$A2$ $\neg((\neg(C \sqcap (D \sqcap (\neg E)))) \sqcap ((\neg(C \sqcap (\neg D))) \sqcap (C \sqcap (\neg E))))$

$A3$ $\neg((\neg((\neg C) \sqcap D)) \sqcap ((\neg((\neg C) \sqcap (\neg C))) \sqcap (\neg C)))$

$A4$ $\neg((\forall R.(\neg(C \sqcap (\neg D)))) \sqcap ((\forall R.C) \sqcap (\neg(\forall R.D))))$

Las reglas de inferencia son Modus Ponens (MP) y Necessitation (Nec).

$$\frac{\neg(C \sqcap \neg D) \qquad C}{D}\ MP \qquad\qquad \frac{C}{\forall R.C}\ Nec$$

Los axiomas $A1$-$A3$ pueden entenderse como una variante notacional de los correspondientes axiomas en Calculo Proposicional Clásico, mientras que el Axioma $A4$ corresponde a la normalidad en la lógica multi-modal K.

Si un concepto descriptivo C es derivable con la axiomatización de $\mathcal{ALC}$, entonces escribimos $\vdash_{\mathcal{ALC}} C$.

Escribiremos $\vdash C$ si no existe confusión al respecto.

1.3. El problema de la satisfacción

Uno de los principales problemas que se presenta en DL es la satisfacción tanto de subsunciones como de conceptos, la cual permite inferir conocimiento de manera general.

Definición 1.4 (Subsunción de conceptos)**.**
Un concepto C se encuentra subsumido por un concepto D si para toda interpretación I, $I \models C \sqsubseteq D$. En tal caso se escribirá $\models C \sqsubseteq D$.

Definición 1.5 (Tautología)**.** *Un concepto C se llamará tautología si cumple que $\models \top \sqsubseteq C$.*

Proposición 1.6 (Reducción a la insatisfacción)**.** *$\models C \sqsubseteq D$ si y sólo si $C \sqcap \neg D$ no se satisface.*

Demostración.

Demostraremos equivalentemente la siguiente afirmación.

$\models C \sqsubseteq D$ si y sólo si $D \sqcup \neg C$ es tautología.

Se cumple que $\models C \sqsubseteq D$ si y sólo si $C^I \subseteq D^I$ para toda interpretación I si y sólo si $C^I \cup (\triangle^I \setminus C^I) \subseteq D^I \cup (\triangle^I \setminus C^I) = D^I \cup (\neg C)^I$ si y sólo si $\top^I \subseteq (D \sqcup \neg C)^I$ si y sólo si $\models \top \sqsubseteq D \sqcup \neg C$.

□

El resultado anterior garantiza la subsunción de un concepto en términos de la insatisfacción.

La correctitud de la axiomatización de $\mathcal{ALC}$ se prueba de manera inmediata.

Teorema 1.7 ([7, 15])**.** *La axiomatización de $\mathcal{ALC}$ es robusta y completa, esto es, para cualquier concepto C de $\mathcal{ALC}$, se tiene que*

$$\vdash C \text{ si y sólo si } \models C$$

Capítulo 2

Una restricción de $\mathcal{ALC}$

En esta sección formalizaremos una lógica basada en $\mathcal{ALC}$ que se obtiene al restringir las propiedades que cumplen sus roles.

Dada una relación binaria r sobre un conjunto S, se dice que r es:

- Simétrica: si para todo $s_1, s_2 \in S$, si $(s_1, s_2) \in r$ entonces $(s_2, s_1) \in r$.
- Serial: si para todo $s_1 \in S$, existe $s_2 \in S$ tal que $(s_1, s_2) \in r$.
- Reflexiva: si para todo $s \in S$ se tiene que $(s, s) \in r$.
- Transitiva: si para cualesquiera $s_1, s_2, s_3 \in S$, si $(s_1, s_2) \in r$ y $(s_2, s_3) \in r$ entonces $(s_1, s_3) \in r$.
- Euclidiana: si para cualesquiera $s_1, s_2, s_3 \in S$, si $(s_1, s_2) \in r$ y $(s_1, s_3) \in r$ entonces $(s_2, s_3) \in r$.

Diremos que un rol R es simétrico, serial, reflexivo, transitivo o euclidiano sobre I si la relación R^I es simétrica, serial, reflexiva, transitiva o euclidiana sobre $\triangle^I$ respectivamente.

Si sobre cualquier interpretación I se cumple que R es simétrico , serial, reflexivo, transitivo o euclidiano, entonces diremos simplemente que R es simétrico , serial, reflexivo, transitivo o euclidiano respectivamente.

En la configuración de conceptos descriptivos de $\mathcal{ALC}$, decimos que una interpretación $I = (\triangle^I, \cdot^I)$ es simétrica, serial, reflexiva, transitiva o euclidiana si y sólo si todos sus roles cumplen la propiedad respectiva.

Las propiedades que cumplen los roles en las lógicas descriptivas repercuten de manera significativa en el comportamiento de sus conceptos.

Los siguientes teoremas relacionan las propiedades de los roles con la satisfacción de conceptos y son demostrables en cualquier lógica descriptiva con expresividad mayor o igual que $\mathcal{ALC}$ y cualquier interpretación I.

Teorema 2.1 (Simetría)**.** *Un rol R es simétrico si y sólo si para cualquier concepto C se cumple que $\models \neg C \sqsubseteq \forall R.(\neg \forall R.C)$.*

Demostración.

Probamos la suficiencia como sigue:

Sea $x \in (\neg C)^I$ y sea y tal que $(x, y) \in R^I$. Entonces $(y, x) \in R^I$. Luego existe un z a saber $z = x$ tal que $z \in (\neg C)^I$. Por lo tanto $x \in (\forall R.(\neg \forall R.C))^I$.

Probemos la necesidad de la siguiente manera:

Sean $x, y \in \triangle^I$ tales que $(x, y) \in R^I$ y sea C un concepto tal que $C^I = \triangle^I \setminus \{x\}$ entonces $x \in (\neg C)^I$. Luego existe z tal que $(y, z) \in R^I$ y también se cumple que $z \in (\neg C)^I$. Por lo que $z = x$ y $(y, z) \in R^I$.

□

Teorema 2.2 (Serialidad)**.** $\models \neg \forall R.\bot$ *si y sólo si el rol R es serial.*

Demostración.

Sea $x \in \triangle^I$ entonces existe y en $\triangle^I$ tal que $(x, y) \in R^I$. Luego $x \in (\neg \forall R.\bot)^I$, lo cual prueba la necesidad.

Si $\neg \forall R.\bot$ es teorema entonces para cada x en $\triangle^I$ se tiene que $x \in (\neg \forall R.\bot)^I$. Por lo tanto para cada x en $\triangle^I$ existe un y en $\triangle^I$ tal que $(x, y) \in R^I$, con lo que la suficiencia queda probada.

□

Teorema 2.3 (Reflexividad)**.** *Un rol R es reflexivo si y sólo si para cualquier concepto C se cumple que $\models \forall R.C \sqsubseteq C$.*

Demostración.

Para probar la la suficiencia. Sea $x \in (\forall R.C)^I$, si y es tal que $(x, y)R^I$ entonces $y \in (C)^I$ y por tanto $x \in C^I$.

Para probar la necesidad. Sea x en $\triangle^I$ y supongamos que $(x, x) \notin R^I$. Sea C tal que $C^I = \triangle^I \setminus \{x\}$ entonces $x \in (\forall R.C)^I$ luego $x \in C$ lo cual es una contradicción. Por lo tanto $(x, x) \in R^I$.

□

Teorema 2.4 (Transitividad)**.** *Un rol R es transitivo si y sólo si para cualquier concepto C se cumple que $\models \forall R.C \sqsubseteq \forall R.\forall R.C$.*

Demostración.

La suficiencia la probamos de la siguiente forma:

Sea $x \in (\forall R.C)^I$, entonces para cada y se cumple que si $(x, y)R^I$, entonces se tiene que $y \in (C)^I$. Luego si z es tal que $(y, z) \in R^I$ se tiene que $(x, z) \in R^I$ por lo cual $z \in C^I$ y por tanto $x \in (\forall R.(\forall R.C))^I$.

La necesidad la probamos como siguen:

Sean $x, y, z \in \triangle^I$ tales que $(x, y) \in R^I$ y $(y, z) \in R^I$. Supongamos que $(x, z) \notin R^I$ Sea C tal que $C^I = \triangle^I \setminus \{z\}$ entonces $x \in (\forall R.C)^I$ y por

tanto $x \in (\forall R.(\forall R.C))^I$. Dado que $(x, y) \in R^I$ y $(y, z) \in R^I$, se cumple que $z \in C^I$ lo cual es una contradicción. Se concluye que $(x, z) \in R^I$.

□

Teorema 2.5 (Euclidianidad)**.** *Un rol R es euclidiano si y sólo si para cualquier concepto C se cumple que $\models \neg(\forall R.C) \sqsubseteq \forall R.(\neg\forall R.C)$.*

Demostración.

Procedemos de la siguiente forma para probar la suficiencia:

Sea $x \in (\neg\forall R.C)^I$ entonces existe z tal que $(x, z)R^I$ y $z \in (\neg C)^I$. Luego sea $y \in \triangle^I$ tal que $(x, y) \in R^I$ entonces $(y, z) \in R^I$ y $z \in (\neg C)^I$. Así se concluye que $x \in (\forall R.(\neg\forall R.C))^I$.

La prueba de la necesidad la realizamos como sigue:

Sean $x, y, z \in \triangle^I$ tales que $(x, y) \in R^I$ y $(x, z) \in R^I$. Sea C un concepto tal que $C^I = \triangle^I \setminus \{z\}$ entonces $x \in (\neg\forall R.C)^I$ y por tanto $x \in (\forall R.(\neg\forall R.C))^I$. Por lo tanto para y existe $w \in (\neg C)^I$ tal que $(y, z) \in R^I$. Así, como $z = w$, se tiene que $(y, z) \in R^I$.

□

Observemos que las reglas que caracterizan a las lógicas proposicionales modales normales son validas en lógicas descriptivas con grado de expresividad mayor que $\mathcal{ALC}$.

Teorema 2.6 (MP)**.** *Si $\models C \sqsubseteq B$ y C es tautología entonces B también lo es.*

La prueba se sigue inmediatamente.

Teorema 2.7 (Necesitación)**.** *Si C es una tautología entonces $\forall R.C$ también lo es.*

Demostración.

C es tautología si y sólo si $\top^I \subseteq C^I$ para cada interpretación I.

Sea $x \in \triangle^I$ y supongamos que y es tal que $(x, y) \in R^I$ entonces $y \in \triangle^I \subseteq C^I$ y por tanto se concluye que $x \in \forall R.C$. □

Teorema 2.8 (Normalidad)**.** $\neg(\forall R(D \sqcup \neg C)) \sqcup \neg(\forall R.C) \sqcup \forall R.D$ *es tautología.*

Demostración.

Sea $x \in \triangle^I$. Supongamos que $x \notin (\neg(\forall R(D \sqcup \neg C)))^I$ entonces $x \in (\forall R.(D \sqcup \neg C))^I$. Supongamos además que $x \notin (\neg(\forall R.C))^I$ entonces $x \in (\forall R.C)^I$. Sea $y \in \triangle^I$ tal que $(x, y) \in R^I$ entonces $y \in (D \sqcup \neg C)^I$ y $y \in C^I$ y por ende $y \in ((D \sqcup \neg C) \sqcap C)^I$, de donde $y \in (D \sqcap C)^I$. Así tenemos que $y \in C^I$ y $y \in D^I$ que es lo que se requería. □

Las siguientes proposiciones muestran algunas propiedades que cumplen los conceptos lógico descriptivos de $\mathcal{ALC}$.

Proposición 2.9. *Si* $\models \forall R.C \sqsubseteq D$ *entonces* $\models \forall R.(\forall R.C) \sqsubseteq \forall R.D$.

Demostración.

Sea $x \in (\forall R.(\forall R.C))^I$ y sea y tal que $(x, y) \in R^I$. Notemos que, por hipótesis, z es tal que $(y, z) \in R^I$ entonces $z \in C^I$.

Supongamos que $y \notin D^I$ entonces $y \in (\neg D)^I \subseteq (\neg(\forall R.C))^I$, por ende existe $z \in (\neg C)^I$ tal que $(y, z) \in R^I$ lo cual contradice que $x \in (\forall R.(\forall R.C))^I$. Por lo tanto $y \notin D^I$ y así $x \in (\forall R.D)^I$. □

En lo sucesivo se empleará la expresión $\models C \equiv D$ como una abreviación para $\models C \sqsubseteq D$ y $\models D \sqsubseteq C$.

Proposición 2.10. $\models \forall R.(C \sqcap D) \equiv \forall R.C \sqcap \forall R.D$.

Demostración.

Sea $x \in (\forall R.(C \sqcap D))^I$, si y es tal que $(x, y) \in R^I$ entonces $y \in (C \sqcap D)^I = C^I \cap D^I$. Luego $x \in (\forall R.C)^I$ y $x \in (\forall R.D)^I$. Por lo tanto $x \in (\forall R.C \sqcap \forall R.D)^I$

Sea $x \in (\forall R.C \sqcap \forall R.D)^I$ entonces $x \in (\forall R.C)^I$ y $x \in (\forall R.D)^I$. Si y es tal que $(x, y) \in R^I$ entonces $y \in (C \sqcap D)^I$. Por lo cual $x \in (\forall R.(C \sqcap D))^I$ □

2.1. *ALCS*4

A continuación presentamos una extensión de la lógica $\mathcal{ALC}$ que, como se puede constatar, corresponde a la la lógica multi-modal S4.

Definición 2.11. *Llamaremos ALCS*4 *a la extensión axiomática de la lógica descriptiva* $\mathcal{ALC}$*, la cual se obtiene restringiendo la semántica a roles reflexivo transitivos.*

Una manera usual de estudiar una lógica formal consiste en definir un conjunto de axiomas y reglas de inferencia y entonces, si es posible, obtener una semántica adecuada para dicha lógica. Sin embargo para estudiar las lógicas descriptivas es más natural proceder de manera inversa, es decir, empezando con una interpretación semántica analizamos las consecuencias validas en la lógica correspondiente y si es posible, definir un conjunto de axiomas y reglas de inferencia.

Definición 2.12. *Axiomatización de ALCS*4

*La axiomatización de ALCS*4 *esta formada por los axiomas y reglas de inferencia de* $\mathcal{ALC}$ *(Definición 1.3) junto con los siguientes axiomas.*

A5 $\neg(\neg C \sqcap \forall R.C)$

A6 $\neg(\neg(\forall R.(\forall R.C)) \sqcap \forall R.C)$

Por abuso de notación, cuando se desprenda del contexto, también escribimos $\vdash C$ cuando un concepto descriptivo C en $ALCS4$ es derivable en sus correspondiente axiomatización.

Observe que $A5$ y $A6$ corresponden a las propiedades de reflexividad y transitividad relativas a los roles, respectivamente.

Teorema 2.13. *La axiomatización de* $ALCS4$ *es sólida y completa, esto es, para cualquier concepto descriptivo* C *y cualquier interpretación* I *reflexivo transitiva, se tiene que*

$$\vdash C \text{ si y sólo si } I \models C$$

Demostración.

Para la suficiencia procedemos como en la prueba del Teorema 1.7([7, 16]), con la condición adicional que los roles son reflexivos y transitivos, y por el Teorema 2.3 y el Teorema 2.4 para cualquier axioma C de $ALCS4$ se tiene que $I \models C$. Y por Teorema 2.6, Teorema 2.7 y Teorema 2.8 se cumple que todo teorema es tautología.

La necesidad se sigue inmediatamente del Teorema 1.7, Teorema 2.3 y el Teorema 2.4.

□

Capítulo 3

Sistemas de Secuentes Lógico Descriptivos

En este capítulo introducimos un cálculo de secuentes para $ALCS4$, el cual es libre de corte y contracción . Probamos también su correctitud y se presenta un análisis sobre la complejidad del sistema de secuentes. Finalmente, se realiza una implementación de dicho cálculo de secuentes en el lenguaje de programación ML.

3.1. Secuentes para ALCS4

Un secuente es una pareja (Γ, Λ), escrito como $\Gamma \Rightarrow \Lambda$, donde Γ y Λ son conjuntos, posiblemente vacíos, de conceptos descriptivos. Intuitivamente, el símbolo $\Rightarrow$ denota que la conjunción de conceptos en Γ implica la disyunción de conceptos en Λ. Generalmente se suele escribir $\Gamma, C_1, ..., C_k$ en lugar de $\Gamma \cup \{C_1, ..., C_k\}$. Asimismo escribiremos $\bigsqcap \Gamma$ y $\bigsqcup \Lambda$ para referirnos a $\bigsqcap_{C \in \Gamma} C$ y $\bigsqcup_{C \in \Lambda} C$ respectivamente.

Una regla de secuente es una tupla de secuentes $(S_0, S_1, S_2, ..., S_n)$ no vacía, la cual se representa de la siguiente manera:

$$\frac{S_1, S_2, ..., S_n}{S_0}$$

Si una regla de secuente sólo tiene un secuente, entonces es llamada un secuente inicial. El significado intuitivo de una regla secuente es que el primer secuente S_0 se deriva de de los demás secuentes S_i $(i > 0)$. Si la regla es un secuente inicial, entonces ese secuente puede interpretarse como un axioma porque no hay necesidad de suposiciones para derivar la

misma. Finalmente un conjunto no vacío de reglas de secuentes se llama un sistema consecuente.

Antes de definir un sistema de secuentes para $ALCS4$, establecemos las siguientes convenciones para ciertas reglas de secuentes.

Si $\Gamma = \{C_1, ..., C_k\}$ entonces denotamos mediante $\forall R.\Gamma$ al conjunto $\{\forall R.C_1, ..., \forall R.C_k\}$.

Definición 3.1 (Sistema de secuentes de $ALCS4$)**.** *Definimos un sistema de secuentes para $ALCS4$ con las siguientes reglas.*

Secuente inicial

$$\frac{}{C, \Gamma \Rightarrow C, \Lambda}\ Ax$$

Reglas de conceptos

$$\frac{\Gamma \Rightarrow C, \Lambda}{\neg C, \Gamma \Rightarrow \Lambda}\ \neg A \qquad \frac{C, \Gamma \Rightarrow \Lambda}{\Gamma \Rightarrow \neg C, \Lambda}\ \neg K$$

$$\frac{C, D, \Gamma \Rightarrow \Lambda}{C \sqcap D, \Gamma \Rightarrow \Lambda}\ \sqcap A \qquad \frac{\Gamma \Rightarrow C, \Lambda \quad \Gamma \Rightarrow D, \Lambda}{\Gamma \Rightarrow C \sqcap D, \Lambda}\ \sqcap K$$

Reglas de cuantificación

$$\frac{C, \Gamma \Rightarrow \Lambda}{\forall R.C, \Gamma \Rightarrow \Lambda}\ \forall A \qquad \frac{\forall R.\Gamma \Rightarrow C}{\forall R.\Gamma \Rightarrow \forall R.C}\ \forall K$$

Intuitivamente, una prueba es definida como un árbol de secuentes donde cada nivel se obtiene por la aplicación de una regla de un sistema de secuentes dado, cada rama es finita y comienza con un secuente inicial.

Considere por ejemplo la siguiente prueba del axioma $A1$ en el sistema de secuentes de $ALCS4$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{C, D \Rightarrow C}\ Ax}{(\neg C), D, C \Rightarrow}\ \neg A}{(D \sqcap (\neg C)), C \Rightarrow}\ \sqcap A}{(C \sqcap (D \sqcap (\neg C))) \Rightarrow}\ \sqcap A}{\Rightarrow (\neg (C \sqcap (D \sqcap (\neg C))))}\ \neg K$$

Una regla de secuentes instanciada es la regla que resulta por el

intercambio, en el esquema de la regla, de las apariciones de un concepto por un concepto especifico.

Definición 3.2 (Árbol de prueba)**.** *Dado un sistema de secuentes, un árbol de derivación se define de manera inductiva como sigue:*

- *cualquier regla del sistema de secuentes instanciada es un árbol derivación;*
- *la siguiente expresión es un árbol de derivación*

$$\frac{T_1 \quad \dots \quad T_n}{S_0}$$

con la condición que T_i $(i = 1, \dots, n)$ es un árbol de derivación, y

$$\frac{S_1 \quad \dots \quad S_n}{S_0}$$

es una regla instanciada, tal que S_i es la raíz correspondiente (secuente más bajo) de T_i.

Un árbol de derivación se denomina ábol de prueba, o simplemente una prueba, si todas sus ramas son finitas y tienen como raíz secuentes iniciales.

Si existe una prueba para un secuente $\Rightarrow \Gamma$, entonces escribimos $\vdash_G \Gamma$ (G debido a Gentzen).

3.1.1. Correctitud

Con el fin de mostrar que $ALCS4$ es correcta, a continuación, mostramos robustez y completitud.

Ahora planteamos el siguiente lema requerida para demostrar robustez.

Lema 3.3. *Si $\forall R.C \sqsubseteq D$ es válido, entonces $\forall R.\forall R.C \sqsubseteq \forall R.D$ también lo es.*

Demostración. Supongamos que $x \in (\forall R.\forall R.C)^I$, si ocurre que y es tal que $(x, y) \in R^I$, entonces $y \in (\forall R.C)^I$ y por hipótesis $y \in (D)^I$ así que $x \in (\forall R.D)^I$. □

Ahora estamos listos para probar la robustez.

Teorema 3.4 (Solidez)**.** *Si existe una prueba del secuente $\Gamma \Rightarrow \Lambda$ en el sistema de secuentes para $ALCS4$, entonces la formula $\bigsqcap \Gamma \sqsubseteq \bigsqcup \Lambda$ es válida en $ALCS4$, esto es, toda interpretación reflexiva y transitiva es un modelo de la fórmula.*

Demostración. Procedemos por inducción sobre la altura del árbol de prueba de $\Gamma \Rightarrow \Lambda$.

El caso base para $C, \Gamma \Rightarrow C, \Lambda$ es trivial.

Si ahora asumimos que tenemos una prueba

$$\frac{T}{\Gamma' \Rightarrow \Lambda'}$$

entonces tenemos que probar que si existe también la prueba

$$\frac{\dfrac{T}{\Gamma' \Rightarrow \Lambda'}}{\Gamma \Rightarrow \Lambda}$$

entonces $\bigsqcap \Gamma \sqsubseteq \bigsqcup \Lambda$ es válida.

Considere el caso cuando el último paso en la prueba es el siguiente:

$$\frac{\Gamma \Rightarrow C, \Lambda}{\neg C, \Gamma \Rightarrow \Lambda}$$

Entonces por inducción tenemos que $\bigsqcap \Gamma \sqsubseteq C \sqcup \bigsqcup \Lambda$ es válida, esto es, para para cualquier interpretación reflexiva y transitiva I, existe un $x \in \Delta^I$, tal que si $x \in (\bigsqcap \Gamma)^I$, entonces $x \in (C)^I$ o $x \in (\bigsqcup \Lambda)^I$. No es difícil entonces concluir que si $x \in (\bigsqcap \Gamma)^I$ y $x \notin (C)^I$, entonces $x \in (\bigsqcup \Lambda)^I$, es decir, $\bigsqcap \Gamma \sqcap \neg C \sqsubseteq \bigsqcup \Lambda$ es también válida.

Procedemos de manera análoga para el caso cuando el último paso de la prueba es una regla con operador booleano.

Considere ahora que el último paso de la prueba es el siguiente:

$$\frac{C, \Gamma \Rightarrow \Lambda}{\forall R.C, \Gamma \Rightarrow \Lambda}$$

Por inducción sabemos que para cualquier interpretación I reflexiva y transitiva, se cumple lo siguiente:

$$\left(C \sqcap \bigsqcap \Gamma\right)^I \subseteq \bigsqcup \Lambda^I$$

También se sabe (por el Teorema 2.3), debido a la reflexividad, que $(\forall R.C)^I \subseteq (C)^I$.

Es entonces evidente que $(\forall R.C \sqcap \bigsqcap \Gamma)^I \subseteq (\bigsqcup \Lambda)^I$.

Consideremos ahora el caso final:

$$\frac{\forall R.\Gamma \Rightarrow C}{\forall R.\Gamma \Rightarrow \forall R.C}$$

Por inducción $\forall R.\Gamma \sqsubseteq C$ es válido. Ahora, sabemos que $\forall R.\forall R.\Gamma \sqsubseteq \forall R.C$ es válido, por el Lema 3.3 y también $\forall R.C \sqcap D \equiv \forall R.C \sqcap \forall R.D$ es válido para cualesquiera C, D y r por la Proposición 2.10.

Ya que toda interpretación es transitiva, entonces también sabemos que $\forall R.\Gamma \sqsubseteq \forall R.\forall R.\Gamma$ (por el Teorema 2.4) es válido. Por consiguiente, $\forall R.\Gamma \sqsubseteq \forall R.C$ es válido. □

La manera de verificar completud será a partir de la completud de la axiomatización de $ALCS4$.

Teorema 3.5. *Si un concepto descriptivo C es derivable en la axiomatización de $ALCS4$, entonces existe una prueba en el sistema de secuentes de $ALCS4$, es decir,*

$$si \ \vdash C, \ entonces \ \vdash_G C.$$

Demostración. Procedemos por inducción sobre la derivación en la axiomatización de $ALCS4$.

El caso base son los axiomas y modus ponens. A1 ya se ha demostrado en un ejemplo anterior. Vamos a demostrar los axiomas restantes y modus ponens. Observe que A4, A5 y A6 son los axiomas representativos de $S4$ y por tanto merecen un poco más de atención. Aun así todos son igual de importantes y dignos de ser demostrados.

Para $A2$ tenemos:

$$\frac{}{C \Rightarrow (C \sqcap (\neg D)), C, E}\ Ax$$

1

$$\dfrac{\dfrac{}{C \Rightarrow E, D, C}\ Ax \quad \dfrac{\dfrac{}{C, D \Rightarrow E, D}\ Ax}{C \Rightarrow (\neg D), D, E}\ \neg K}{C \Rightarrow (C \sqcap (\neg D)), D, E}\ \sqcap K \quad \dfrac{\dfrac{}{C, E \Rightarrow (C \sqcap (\neg D)), E}\ Ax}{C \Rightarrow (\neg E), (C \sqcap (\neg D)), E}\ \neg K}{C \Rightarrow (D \sqcap (\neg E)), (C \sqcap (\neg D)), E}\ \sqcap K$$

2

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{1 \quad 2}{C \Rightarrow (C \sqcap (D \sqcap (\neg E))), (C \sqcap (\neg D)), E}\ \sqcap K}{(\neg E), C \Rightarrow (C \sqcap (\neg D)), (C \sqcap (D \sqcap (\neg E)))}\ \neg A}{(C \sqcap (\neg E)) \Rightarrow (C \sqcap (D \sqcap (\neg E))), (C \sqcap (\neg D))}\ \sqcap A}{(\neg(C \sqcap (\neg D))), (C \sqcap (\neg E)) \Rightarrow (C \sqcap (D \sqcap (\neg E)))}\ \neg A}{(\neg(C \sqcap (D \sqcap (\neg E)))), (C \sqcap (\neg E)), (\neg(C \sqcap (\neg D))) \Rightarrow}\ \neg A}{((\neg(C \sqcap (\neg D))) \sqcap (C \sqcap (\neg E))), (\neg(C \sqcap (D \sqcap (\neg E)))) \Rightarrow}\ \sqcap A}{((\neg(C \sqcap (D \sqcap (\neg E)))) \sqcap ((\neg(C \sqcap (\neg D))) \sqcap (C \sqcap (\neg E)))) \Rightarrow}\ \sqcap A}{\Rightarrow (\neg((\neg(C \sqcap (D \sqcap (\neg E)))) \sqcap ((\neg(C \sqcap (\neg D))) \sqcap (C \sqcap (\neg E)))))}\ \neg K$$

Para $A3$ tenemos:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{C \Rightarrow (\neg C), C}\,Ax}{\Rightarrow (\neg C), (\neg C), C}\,\neg K \quad \dfrac{\dfrac{}{C \Rightarrow C, D}\,Ax}{\Rightarrow (\neg C), D, C}\,\neg K}{\Rightarrow ((\neg C) \sqcap D), (\neg C), C}\,\sqcap K \quad \dfrac{\dfrac{}{C \Rightarrow ((\neg C) \sqcap D), C}\,Ax}{\Rightarrow (\neg C), ((\neg C) \sqcap D), C}\,\neg K}{\Rightarrow ((\neg C) \sqcap (\neg C)), ((\neg C) \sqcap D), C}\,\sqcap K}{(\neg C) \Rightarrow ((\neg C) \sqcap D), ((\neg C) \sqcap (\neg C))}\,\neg A}{(\neg((\neg C) \sqcap (\neg C))), (\neg C) \Rightarrow ((\neg C) \sqcap D)}\,\neg A}{(\neg((\neg C) \sqcap D)), (\neg C), (\neg((\neg C) \sqcap (\neg C))) \Rightarrow}\,\neg A}{((\neg((\neg C) \sqcap (\neg C))) \sqcap (\neg C)), (\neg((\neg C) \sqcap D)) \Rightarrow}\,\sqcap A}{((\neg((\neg C) \sqcap D)) \sqcap ((\neg((\neg C) \sqcap (\neg C))) \sqcap (\neg C))) \Rightarrow}\,\sqcap A}{\Rightarrow (\neg((\neg((\neg C) \sqcap D)) \sqcap ((\neg((\neg C) \sqcap (\neg C))) \sqcap (\neg C))))}\,\neg K$$

Para $A4$ tenemos:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{C \Rightarrow D, C}\,Ax \quad \dfrac{\dfrac{}{C, D \Rightarrow D}\,Ax}{C \Rightarrow (\neg D), D}\,\neg K}{C \Rightarrow (C \sqcap (\neg D)), D}\,\sqcap K}{(\neg(C \sqcap (\neg D))), C \Rightarrow D}\,\neg A}{(\forall R.(\neg(C \sqcap (\neg D)))), C \Rightarrow D}\,\forall A}{(\forall R.C), (\forall R.(\neg(C \sqcap (\neg D)))) \Rightarrow D}\,\forall A}{(\forall R.C), (\forall R.(\neg(C \sqcap (\neg D)))) \Rightarrow (\forall R.D)}\,\forall K}{(\neg(\forall R.D)), (\forall R.C), (\forall R.(\neg(C \sqcap (\neg D)))) \Rightarrow}\,\neg A}{((\forall R.C) \sqcap (\neg(\forall R.D))), (\forall R.(\neg(C \sqcap (\neg D)))) \Rightarrow}\,\sqcap A}{((\forall R.(\neg(C \sqcap (\neg D)))) \sqcap ((\forall R.C) \sqcap (\neg(\forall R.D)))) \Rightarrow}\,\sqcap A}{\Rightarrow (\neg((\forall R.(\neg(C \sqcap (\neg D)))) \sqcap ((\forall R.C) \sqcap (\neg(\forall R.D)))))}\,\neg K$$

Para $A5$ tenemos:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\forall R.C) \Rightarrow (\forall R.C)}\,Ax}{(\forall R.C) \Rightarrow (\forall R.(\forall R.C))}\,\forall K}{(\neg(\forall R.(\forall R.C))), (\forall R.C) \Rightarrow}\,\neg A}{((\forall R.C) \sqcap (\neg(\forall R.(\forall R.C)))) \Rightarrow}\,\sqcap A}{\Rightarrow (\neg((\forall R.C) \sqcap (\neg(\forall R.(\forall R.C)))))}\,\neg K$$

Para $A6$ tenemos:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{C \Rightarrow C}\,Ax}{(\forall R.C) \Rightarrow C}\,\forall A}{(\neg C), (\forall R.C) \Rightarrow}\,\neg A}{((\forall R.C) \sqcap (\neg C)) \Rightarrow}\,\sqcap A}{\Rightarrow (\neg((\forall R.C) \sqcap (\neg C)))}\,\neg K$$

Para MP tenemos:

$$\dfrac{\dfrac{\dfrac{}{C \Rightarrow D, C}\,Ax \quad \dfrac{\dfrac{}{C, D \Rightarrow D}\,Ax}{C \Rightarrow (\neg D), D}\,\neg K}{C \Rightarrow (C \sqcap (\neg D)), D}\,\sqcap K}{(\neg(C \sqcap (\neg D))), C \Rightarrow D}\,\neg A$$

Aquí $(\neg(C \sqcap (\neg D)))$ corresponde a $C \sqsubseteq D$ traduciendo subsunción y disyunción.

Para el paso inductivo, asumimos que tenemos una prueba para un concepto descriptivo C, entonces mostramos la regla de necesitación:

$$\dfrac{\dfrac{}{\Rightarrow C}}{\Rightarrow (\forall R.C)}\,\forall K$$

□

Corolario 3.6 (Completitud)**.** *Para cualquier interpretación reflexiva y transitiva I, si I es un modelo para un concepto descriptivo C, entonces existe una prueba de C en el sistema de secuentes de $ALCS4$, es decir, $\vdash_G C$.*

Es consecuencia inmediata de los Teoremas 3.5 y 2.13.

Es importante indicar la existencia de una jerarquía para la aplicación de las reglas. Primero aplicamos reglas lógicas, seguido de la regla de cuantificación derecha (si es posible) y concluyendo con la regla de cuantificación izquierda.

3.1.2. Complejidad e implementación

Vamos a demostrar que el sistema secuente para $ALCS4$ está en EXPTIME. Esto se debe principalmente al hecho de que las pruebas tienen forma de árboles binarios y existe una cota exponencial sobre el número de árboles binarios.

Teorema 3.7 (Complejidad)**.** *El sistema de secuentes para $ALCS4$ está en EXPTIME.*

Demostración. Note primero que el sistema satisface la propiedad de la subfórmula, es decir, para cada regla (S_0, S_1) o (S_0, S_1, S_2), las fórmulas en S_1 o S_2 son todas subfórmulas de las que ocurren en S_0. Entonces es evidente que la altura de las pruebas (número de pasos de prueba) es lineal con respecto al tamaño de los secuentes de entrada. Note ahora que debido a la regla conjunción a la derecha, las pruebas son en forma de árbol binario. La cota exponencial proviene del hecho de que el número de nodos en árboles binarios es exponencialmente acotado respecto a la altura del árbol. □

En cuanto a la implementación, hemos seguido un enfoque funcional como en [13]. En particular, el sistema $ALCS4$ se implementó en ML [9].Para ello consideramos tres tipos de datos fundamentales: descripciones de conceptos cuyo alfabeto está formado por nombres de conceptos y roles, secuentes definidos como un par de listas de concepto y reglas, que pueden ser axiomas y reglas con una o dos hipótesis.

En el Apéndice A.1 se presenta el código para la implementación, la cual fue verificada en una PC con procesador Intel Core i5 de 1.8 GHz, Windows 8.1, en Moscow ML versión 2.01. Por ejemplo, la verificación de la validez del Axioma 5, presente en la demostración del Teorema 3.5, requirió en nuestra implementación de 16.4844 milisegundos.

3.2. Árboles de hipersecuentes para $\mathcal{ALC}$

En esta sección, introducimos primero la noción de árbol de hipersecuentes, el cual es una generalización de los secuentes. Despues se introduce un sistema de árboles de hypersecuentes para $\mathcal{ALC}$, sin ningún tipo de restricción en los roles. También se dan las pruebas para la correctitud y complejidad (2EXPTIME). Además se describe la implementación correspondiente.

3.2.1. El cálculo

Los árboles de hipersecuentes como su nombre sugiere son estructuras de tipo árbol, donde cada uno de sus nodos es un secuente. La siguiente es una definición más precisa.

Definición 3.8 (Árbol de hipersecuentes)**.** *Un árbol de hipersecuentes* (THS) *por sus sigla en inglés se define de manera inductiva por la siguiente gramática:*

$$T := S/MT$$

$$MT := \emptyset T; MT$$

donde S *es un secuente.*

En cuanto a la notación, escribimos S, en lugar de $S/\emptyset$.
Como en [10], se define el zoom para árboles de hipersecuentes.

Definición 3.9. *El conjunto de zoom de árbol de hipersecuentes $ZTHS$ es definido inductivamente de la siguiente manera:*

- $[-] \in ZTHS$.
- *Si $T_1, ..., T_n \in THS$, $r_1, ..., r_n$ son nombres de roles, entonces $[-]/r_1 : T_1; ...; r_n : T_n \in ZTHS$*
- *Si S es un secuente, $T_2, ..., T_n \in THS$, $r_1, ..., r_n$ son nombres de roles y $T_1[-] \in ZTHS$, entonces $S/r_1 : T_1[-]; r_2 : T_2; ...; r_n : T_n \in ZTHS$*

$T[T']$ denota la substitución de T' en $T[-]$ la cual es definida de la siguiente manera:

- Si $T[-] = [-]$, entonces $T[T'] = T'$.
- Si $T[-] = [-]/r_1 : T_1; ...; r_n : T_n$ y $T' = S'/w_1 : T'_1; ...; w_k : T'_k$, entonces $T[T'] = S'/r_1 : T_1; ...; r_n : T_n; w_1 : T'_1; ...; w_k : T'_k$.
- Si $T[-] = S/r_1 : T_1[-]; r_2 : T_2; ...; r_n : T_n$, entonces $T[T'] = S/r_1 : T_1[T']; r_2 : T_2; ...; r_n : T_n$.

Definición 3.10 (Sistema de árboles de hipersecuentes para $\mathcal{ALC}$)**.** *Definimos un sistema de árboles de hipersecuentes para $\mathcal{ALC}$ mediante las siguientes reglas.*

Árbol de hipersecuentes inicial

$$\frac{}{T[C, \Gamma \Rightarrow C, \Lambda]}\ Ax$$

Reglas de conceptos

$$\frac{T[\Gamma \Rightarrow C, \Lambda]}{T[\neg C, \Gamma \Rightarrow \Lambda]}\ \neg A \qquad \frac{T[C, \Gamma \Rightarrow \Lambda]}{T[\Gamma \Rightarrow \neg C, \Lambda]}\ \neg K$$

$$\frac{T[C, D, \Gamma \Rightarrow \Lambda]}{T[C \sqcap D, \Gamma \Rightarrow \Lambda]}\ \sqcap A \qquad \frac{T[\Gamma \Rightarrow C, \Lambda] \quad T[\Gamma \Rightarrow D, \Lambda]}{T[\Gamma \Rightarrow C \sqcap D, \Lambda]}\ \sqcap K$$

Reglas de cuantificación

$$\frac{T\,[(\forall R.C, \Gamma \Rightarrow \Lambda)/R : (C, \Gamma' \Rightarrow \Lambda'/MT'); MT]}{T\,[(\forall R.C, \Gamma \Rightarrow \Lambda)/R : (\Gamma' \Rightarrow \Lambda'/MT'); MT]}\ \forall RA$$

$$\frac{T[(\Gamma \Rightarrow \Lambda)/R : (\Rightarrow C)]}{T[\Gamma \Rightarrow \forall R.C, \Lambda]}\ \forall RK$$

Observe que la regla $\forall RA$ se puede aplicar a un hipersecuente de la forma $T[(\forall R.C, \Gamma \Rightarrow \Lambda)/MT]$ siempre que exista un hipersecuente en MT que esté etiquetado por el rol R. Por lo tanto, si cada hipersecuente en MT etiquetado por R contiene el concepto C en el antecedente de la raíz, entonces la regla $\forall RA$ no se puede aplicar y por lo tanto podemos eliminar al concepto $\forall R.C$ del hipersecuente T.

Como ejemplo de la utilización del sistema de árboles de hipersecuentes, consideremos la siguiente prueba de $A4$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Rightarrow /R : (C \Rightarrow D, C)}\,Ax \quad \dfrac{\dfrac{}{\Rightarrow /R : (D, C \Rightarrow D)}\,Ax}{\Rightarrow /R : (C \Rightarrow (\neg D), D)}\,\neg K}{\Rightarrow /R : (C \Rightarrow (C \sqcap (\neg D)), D)}\,\sqcap A}{\Rightarrow /R : ((\neg(C \sqcap (\neg D))), C \Rightarrow D)}\,\neg A}{(\forall R.(\neg(C \sqcap (\neg D)))) \Rightarrow /R : (C \Rightarrow D)}\,\forall RA}{(\forall R.C), (\forall R.(\neg(C \sqcap (\neg D)))) \Rightarrow /R : (\Rightarrow D)}\,\forall RA}{(\forall R.C), (\forall R.(\neg(C \sqcap (\neg D)))) \Rightarrow (\forall R.D)}\,\forall RK}{(\neg(\forall R.D)), (\forall R.C), (\forall R.(\neg(C \sqcap (\neg D)))) \Rightarrow}\,\neg A}{((\forall R.C) \sqcap (\neg(\forall R.D))), (\forall R.(\neg(C \sqcap (\neg D)))) \Rightarrow}\,\sqcap A}{((\forall R.(\neg(C \sqcap (\neg D)))) \sqcap ((\forall R.C) \sqcap (\neg(\forall R.D)))) \Rightarrow}\,\sqcap A}{\Rightarrow (\neg((\forall R.(\neg(C \sqcap (\neg D)))) \sqcap ((\forall R.C) \sqcap (\neg(\forall R.D)))))}\,\neg K$$

3.2.2. Correctitud

Ahora mostraremos que el sistema de árboles de hipersecuentes para $\mathcal{ALC}$ es correcto.

Los árboles de hipersecuentes se interpretan como una disyunción compuesta por sus ramas. Más precisamente, definimos inductivamente la siguiente función de interpretación:

- $(C_1, \ldots, C_n \Rightarrow D_1, \ldots, D_m)^H = \prod_{i=1}^{n} C_i \sqsubseteq \bigsqcup_{i=1}^{m} D_i$, y

- $(S/R_1 : T_1, \ldots, R_k : T_k)^H = S^H \sqcup \bigsqcup_{i=1}^{k} \forall R_i.({T_i}^H)$.

Teorema 3.11 (Solidez)**.** *Si existe una prueba del árbol de hipersecuentes T en el sistema de árboles de hipersecuentes para $\mathcal{ALC}$, entonces T^H es válida.*

Demostración. Procedemos por inducción sobre la altura de la prueba de T.

En el caso base, distinguimos dos casos: uno cuando $C, \Gamma \Rightarrow C, \Lambda$ ocurre en la raíz de T, y el otro cuando esto no pasa.

En el primer caso esta claro que $C \sqcap \bigsqcap \Gamma \sqsubseteq C \sqcup \bigsqcup \Lambda$ es válido. En el segundo caso, es fácil ver que $\forall R_1 \ldots . \forall R_n . (C \sqcap \bigsqcap \Gamma \sqsubseteq C \sqcup \bigsqcup \Lambda)$ también es válido, ya que si un concepto D se satisface, entonces $\forall RD$ también se satisface para cualquier rol R, debido a que $\Delta^I \subseteq D^I$ se cumple.

En el paso inductivo asumimos una prueba de un árbol T', y entonces mostramos por casos que la regla $\frac{T'}{T}$ puede ser aplicada. Considere ahora el caso cuando el último paso de la prueba es el siguiente.

$$\frac{T[\Gamma \Rightarrow C, \Lambda]}{T[\neg C, \Gamma \Rightarrow \Lambda]}$$

En este caso, también distinguimos dos subcasos: cuando $\neg C, \Gamma \Rightarrow \Lambda$ ocurre en la raíz de T, y cuando esto no sucede. En el primer caso, sabemos que $\Gamma \Rightarrow C, \Lambda$ también ocurre en la raíz de T y éste es también válido (por hipótesis inductiva). Entonces para cualquier interpretación I, sabemos que $(\bigsqcap \Gamma)^I \subseteq C^I \cup (\bigsqcup \Lambda)^I$. De esto, implicamos que $(\bigsqcap \Gamma)^I \cap \left(C^I\right)^c \subseteq (\bigsqcup \Lambda)^I \cap (C^I)^c \subseteq (\bigsqcup \Lambda)^I$, y por esto $\neg C, \Gamma \Rightarrow \Lambda$ es válido. Los otros casos cuando el último paso de la prueba involucra operadores booleanos se prueba de manera análoga.

Nos enfocaremos ahora al caso de la cuantificación izquierda.

$$\frac{T\left[(\forall R.C, \Gamma \Rightarrow \Lambda)/R : (C, \Gamma' \Rightarrow \Lambda'/MT')\right]}{T\left[(\forall R.C, \Gamma \Rightarrow \Lambda)/R : (\Gamma' \Rightarrow \Lambda'/MT')\right]}$$

Sólo probaremos el caso básico y cuando $MT' = \emptyset$, el argumento para los otros casos es similar. Entonces, por hipótesis inductiva, obtenemos que el siguiente concepto es válido.

$$\neg \left(\forall R.C \sqcap \bigsqcap \Gamma\right) \sqcup \bigsqcup \Lambda \sqcup \forall R. \left(\neg \left(C \sqcap \bigsqcap \Gamma'\right) \sqcup \bigsqcup \Lambda'\right)$$

Entonces por las leyes de De Morgan:

$$\neg \forall R.C \sqcup \neg \bigsqcap \Gamma \sqcup \bigsqcup \Lambda \sqcup \forall R. \left(\neg C \sqcup \neg \bigsqcap \Gamma' \sqcup \bigsqcup \Lambda'\right)$$

y por normalidad:

$$\neg\forall R.C \sqcup \neg\bigsqcap\Gamma \sqcup \bigsqcup\Lambda \sqcup \forall R.\neg C \sqcup \forall R.\left(\neg\bigsqcap\Gamma' \sqcup \bigsqcup\Lambda'\right)$$

y puesto que $\forall R.\neg C \sqsubseteq \neg\forall R.C$ se satisface, entonces claramente se implica que $((\forall R.C, \Gamma \Rightarrow \Lambda)/R : (\Gamma' \Rightarrow \Lambda'))^H$ es válido.

Cuantificación a la derecha:

$$\frac{T[(\Gamma \Rightarrow \Lambda)/R : (\Rightarrow C)]}{T[\Gamma \Rightarrow \forall R.C, \Lambda]}$$

Una vez más sólo probaremos el caso base cuando $T[-] = [-]$. Por inducción, sabemos que el siguiente concepto es válido $(\neg\bigsqcap\Gamma) \sqcup \bigsqcup\Lambda \sqcup \forall R.(\neg\top \sqcup C)$. Entonces $(\neg\bigsqcap\Gamma) \sqcup \bigsqcup(\forall R.C, \Lambda)$ es válido. Lo cual claramente implica que la raíz del hipersecuente es es válida bajo la función H.

□

La completitud también se demuestra con respecto a la completitud de la axiomatización de $\mathcal{ALC}$. Probaremos primero que el sistema de árboles de hipersecuentes para $\mathcal{ALC}$ es completo con respecto a la axiomatización de $\mathcal{ALC}$.

Teorema 3.12. *Si un concepto descriptivo C es derivable en la axiomatización de $\mathcal{ALC}$, entonces hay una prueba en el sistema de árboles de hipersecuentes de $\mathcal{ALC}$, es decir,*

$$\textit{si } \vdash C, \textit{ entonces } \vdash_H C.$$

Demostración. La prueba se hará por inducción sobre la derivación en la axiomatización de $\mathcal{ALC}$. En el caso base, el único subcaso interesante es la prueba del axioma $A4$, la cual se dio como un ejemplo anterior. Para el paso inductivo, asumimos que ya tenemos una prueba de un concepto C, entonces probemos la regla de necesitación.

$$\frac{\dfrac{}{\Rightarrow /R : (\Rightarrow C)}}{\Rightarrow (\forall R.C)}\ \forall RK$$

□

La completud es ahora clara del Teorema 3.12 y del Teorema 1.7.

Corolario 3.13 (Completud)**.** *Para cualquier interpretación I, si I es un modelo para un concepto descriptivo C, entonces existe una prueba de C en el sistema de árboles de hipersecuentes para $\mathcal{ALC}$, esto es, $\vdash_H C$.*

3.2.3. Complejidad e implementación

La complejidad del sistema de árboles de hipersecuentes para $\mathcal{ALC}$ está en 2EXPTIME. Intuitivamente, las pruebas en este sistema son árboles binarios donde sus números de nodos son acotados exponencialmente por sus respectivas alturas, que están así mismo acotadas de manera exponencial po el tamaño del concepto de entrada.

Teorema 3.14 (Complexity)**.** *El sistema de árboles de hipersecuentes para $\mathcal{ALC}$ esta en 2EXPTIME.*

Demostración. La primera cota exponencial proviene de la cota en el número de nodos en árboles binarios con respecto a la altura del árbol. Recordemos de la prueba de complejidad del sistema de secuentes de $ALCS4$ que las reglas correspondientes a conectivos booleanos producen árboles de prueba binarios, debido a la propiedad de la subfórmula.

Vamos a demostrar la segunda cota exponencial: la altura de los árboles de prueba está acotad de manera exponencial por el tamaño del concepto de entrada. Para este propósito, primero recuérdese que cada nodo del árbol de prueba es un árbol de hipersecuente. A continuación, se procede a demostrar que el número de nodos para cada árbol de hipersecuente está limitado de manera exponencial por el tamaño del concepto.

En primer lugar, debido a la propiedad de la subfórmula, es fácil ver que el rango de cada árbol de está linealmente acotado por el tamaño del concepto. Sin embargo, debido a la regla de cuantificación universal en el lado izquierdo, hay duplicaciones (nodos en los árboles de hipersecuentes). Ahora, existe una bien conocida y directa reducción lineal de árboles n-arios a árboles binarios [3, 4, 5], de tal manera que la altura de los árboles binarios está linealmente acotada por el rango de los árboles n-arios. Entonces, la altura de la versión binaria de los árboles de los hipersecuentes es también lineal. Recordando de nuevo la cota exponencial sobre el número de nodos en árboles binarios con respecto a la altura, se obtiene que el tamaño de cada árbol de hipersecuente (nodo prueba) es exponencial con respecto al tamaño del concepto de entrada. □

La implementación del sistema de árboles de hipersecuentes para $\mathcal{ALC}$ es similar a la descrita para el sistema de secuentes de $ALCS4$ en la Sección 3.1. Sin embargo, en el caso del sistema para $\mathcal{ALC}$, la estructura fundamental de datos está compuesta por árboles de hipersecuentes, que es descrita en la figura 3.1. En esta figura, también se muestra un tipo de datos para árboles a prueba con árboles de hipersecuentes como nodos. Este tipo de datos puede tener ninguna regla aplicable, un axioma, o una regla con una o dos hipótesis. La prueba del axioma A4, presentada anteriormente, toma 19,0497 milisegundos en nuestra aplicación. El código completo se describe en el Apéndice A.

```
datatype THS =
        hyper of Sequent *
        ((Roles * THS) list);

datatype SystemGH = NoRule
| AXITHS of THS
| InRuOneH of THS
| InRuTwoH of THS * THS;
```

Figura 3.1: Estructura de datos para pruebas con árboles de hipersecuentes

Conclusiones

En este trabajo se caracterizaron propiedades de los roles en sistemas lógico descriptivos con complemento, lo cual permitió probar la correctud de los algoritmos propuestos. Introducimos las teorías de prueba básicas para dos lógicas descriptivas importantes (DL). El lenguaje conceptual básico proposicionalmente cerrado $\mathcal{ALC}$, y su variante con funciones reflexivas y transitivas, que nombramos $ALCS4$ debido a la directa correspondencia con la lógica multi-modal $S4$. El sistema para $ALCS4$ disfruta de la eliminación de corte, que permite demostrar decidibilidad. Además, este sistema es también libre de contracción. Estas dos características son requisitos importantes para la implementación de sistemas de tipo secuentes. También se proporciona la implementación. El sistema se ha demostrado también correcto: robusto y completo. Y se muestra que la complejidad del sistema está en EXPTIME.

También introducimos un sistema de prueba para ALC sin restricción en roles. El sistema se basa en árboles de hipersecuentes, que es la generalización de secuentes. Este sistema también es libre de corte y la contracción. También se dan las pruebas la correctud y complejidad (2EXPTIME) del sistema junto con la implementación.

Hemos cubierto tanto la parte teórica como la parte de implementación computacional, pero resta atender a la parte de los problemas reales de aplicación. Creemos que los marcos de razonamiento proporcionados en el actual trabajo permitirán más estudios en lógica descriptiva desde una perspectiva de teoría de prueba. En particular, estamos interesados en pruebas constructivas de interpolación de Craig en el contexto DL. Otra perspectiva de la investigación inmediata es el estudio del sistema de prueba de tipo secuente para lógicas descriptivas incluyendo razonamiento terminológico (TBoxes) y afirmacional (ABoxes). También en este contexto, planeamos estudiar lógicas descriptivas más expresivas, incluyendo restricciones aritméticas (numéricas), roles inversos y nominales [3, 4, 5].

Apéndice A

Implementación en ML

ML es un lenguaje de programación funcional, interpretado, fuertemente tipificado que dentro de sus características importantes se encuentra el polimorfismo parametrizado, lo cual permite definir funciones aplicables a diferentes parámetros o tipos de datos. ML es acrónimo de Meta Lenguaje, fue desarrollado por Robin Milner y otros a finales de los años 70's como un lenguaje para desarrollar tácticas de demostración en el sistema LCF (un demostrador automático de teoremas).

En la actualidad están disponibles varios lenguajes de la familia ML, entre los principales se encuentran: Standard ML (SML) y Ocaml (Ocaml contiene la sintaxis de ML como un subconjunto).

Los lenguajes de la familia ML se aplican principalmente en diseño y manipulación de lenguajes de programación (compiladores, analizadores, demostradores de teoremas), así como en bioinformática, sistemas financieros, protocolos de sincronización, etc.

Para crear un tipo datos en ML se utiliza el comando datatype el cual funciona como en el siguiente ejemplo:

```
datatype Roles=  role of string;

datatype Conc= conc of string |
               nega of Conc |
               conj of Conc * Conc|
               cuan of Roles * Conc;
```

Aquí datatype permite construir los tipos de datos Roles y Conc. Role tiene un constructor llamado role que admite elementos del tipo string, así por ejemplo role(" R") es un elemento de Role. Conc tiene cuatro constructores conc que admite elementos de tipo string, nega que admite elementos de Conc, conj que admite parejas de elementos de Conc

y cuan que admite parejas con primer elemento en Roles y segundo en Conc, así por ejemplo cuan(role("R"),nega(conc("A"))) es un elemento de Conc y representa al concepto descriptivo $\forall R.\neg A$.

Para darle un alias a una estructura se utiliza la palabra type como en el siguiente ejemplo:

```
type Seq = Conc list;
```

Aquí type permite referirse a listas de Conc cuando se menciona a Seq.

Las listas son una estructura de datos que se puede definir inductivamente. La lista vacía es su elemento base y cada elemento de la lista es la lista vacía o tiene un primer elemento y lo demás es una lista.

Lo anterior podría ser definido formalmente de la siguiente manera:

- [ ] es una lista y
- si a es el primer elemento de $a :: r$ y r es una lista entonces $a :: r$ es una lista.

Así por ejemplo $[a]$ la lista que sólo tiene a a como único elemento puede ser escrita como $a :: [\,]$.

A continuación se presenta la implementación en ML de los sistemas descritos en el presente trabajo.

A.1. Implementación del sistema de secuentes para $ALCS4$

```
(*Tipo de datos para escribir conceptos*)
datatype Roles=  role of string;
datatype Conc= conc of string |
               nega of Conc |
               conj of Conc * Conc|
               cuan of Roles * Conc;


(*Alias para secuencia de conceptos*)
type Seq = Conc list;


(*Tipo de datos para escribir secuentes*)
datatype Sequent = lr of Seq * Seq;


(*Tipo de datos para escribir árboles de secuentes*)
```

```
datatype SystemG = Ax of Sequent
|         InRuOne of Sequent
|         InRuTwo of Sequent * Sequent;
```

(*Reorganiza una lista enviando las cuantificaciones al final*)

```
fun reorgcuan(nil,pa)=pa
|   reorgcuan(cuan(x)::r,pa)=
    reorgcuan(r,pa@[cuan(x)])
|   reorgcuan(c::r,pa)=
    reorgcuan(r,c::pa);
```

(*Reorganiza una lista enviando los conceptos atómicos al final*)

```
fun reorgconc(nil,pa)=pa
|   reorgconc(conc(a)::r,pa)=
    reorgconc(r,pa@[conc(a)])
|   reorgconc(list,pa)=
    list@pa;
```

(*Reorganiza una lista enviando los conceptos atómicos al final,y en penúltimo lugar las cuantificaciones*)

```
fun reorg(lista)=reorgconc(reorgcuan(lista,nil),nil);
```

(*Determina si existe un concepto con operadores lógicos en una lista y retorna una lista con el dicho concepto a la cabeza*)

```
fun thereIsOperator(nil,pa) = (false,pa)
|   thereIsOperator(conc(a)::r,pa) =
    thereIsOperator(r,conc(a)::pa)
|   thereIsOperator(x,pa) = (true,x@pa);
```

(*Determina si existe un conepto con conjunción o negación en una lista*)

```
fun thereIsCoNeg(nil) = false
|   thereIsCoNeg(conc(a)::r) =
    thereIsCoNeg(r)
|   thereIsCoNeg(cuan(x)::r) =
    thereIsCoNeg(r)
|   thereIsCoNeg(_) =true;
```

(*Determina si un concepto esta en una lista *)

```
fun esta(_,nil) = false
|   esta(p,c::r) = p=c
    orelse esta(p,r);
```

(*Determina si hay una intersección entre dos listas *)

```
fun intersect(nil,_) = false
|   intersect(c::r,list) = esta(c,list)
orelse intersect(r,list);
```

(*Determina si un concepto es una i-cuantificación *)

```
fun isCuan(i,cuan(r,a))=
i=r
|   isCuan(i,c)=false;
```

(*Determina si en una lista sólo tiene i-cuantificación *)

```
fun ThereIsOnlyCuan(i,nil)= true
|   ThereIsOnlyCuan(i,a::r)=
if isCuan(i,a)
then ThereIsOnlyCuan(i,r)
else false;
```

(*Determina si un concepto es una r-cuantificación y regresa en rol r*)

```
fun isCuan1(cuan(role "r",a))= (true, role "r")
|   isCuan1(_)=(false, role "");
```

(*Determina si una lista sólo tiene i-cuantificaciones y regresa el rol i*)

```
fun isOnlyCuan(nil)= (true, role "")
|   isOnlyCuan(a::r)=
let val(valor,rol) = isCuan1(a)
in
if ThereIsOnlyCuan(rol,r)
then (true,rol)
else (false, role "")
end;
```

(*Quita la i-cuantificación a una lista*)

```
fun takeAwayCuan(i,nil)= nil
|   takeAwayCuan(i,cuan(l,a)::r)=
if i=l then a::takeAwayCuan(i,r)
else cuan(l,a)::takeAwayCuan(i,r)
|   takeAwayCuan(i,b::r)= b::takeAwayCuan(i,r);
```

(*Transcribe los roles a código latex*)

```
fun rol2tex(role(a))= a;
```

(*Transcribe los conceptos a código latex*)

```
fun conc2lat(conc(a))= a
|   conc2lat(conj(a,b))=
"("^conc2lat(a)^"\\sqcap "^conc2lat(b)^")"
|   conc2lat(nega(t))=
"("^"\\neg "^conc2lat(t)^")"
|   conc2lat(cuan(role(r),a))=
"("^"\\forall "^r^"."^conc2lat(a)^")";
```

(*Transcribe una lista a código latex*)

```
fun Seq2te(nil) = " "
|   Seq2te([algo]) = conc2lat(algo)
|   Seq2te(c::r) =
conc2lat(c)^","^Seq2te(r);
```

(*Determina cuando es aplicable la regla CuanK*)

```
fun esAplicableCuank(lr(gamma,c::nil)) =
let val(valor,rol) = isCuan1(c)
in
  let val(valor1,rol1) = isOnlyCuan(gamma)
  in
  valor andalso valor1
  andalso (rol=rol1 orelse gamma=nil)
  end
end
|   esAplicableCuank(lr(gamma,delta)) = false;
```

(*Decide que si aplica alguna regla o termina el árbol de secuente*)

```
fun decide(lr(delta,gama)) =
if thereIsCoNeg(gama)
```

```
orelse esAplicableCuank(lr(delta,gama))
then InRuOne(lr(delta,reorg(gama)))
else
let
val(valor,delta1) = thereIsOperator(delta,nil)
in
if valor
then InRuOne(lr(delta1,gama))
else Ax(lr(delta,gama))
end;
```

(*Aplica las reglas de secuentes y los va reduciendo*)

```
fun reduce2tex1(lr(nega(a)::gamma,delta),apstr) =
(InRuOne(lr(reorg(gamma),reorg(a::delta))),
"\\infer[\\neg␣A␣]"^"{"^Seq2te(nega(a)::gamma)^
"\\Rightarrow␣"^Seq2te(delta)^"}\n")
|   reduce2tex1(lr(conj(a,b)::gamma,delta),apstr) =
(InRuOne(lr(reorg(a::b::gamma),reorg(delta))),
"\\infer[\\sqcap␣A␣]"^"{"^Seq2te(conj(a,b)::gamma)^
"\\Rightarrow␣"^Seq2te(delta)^"}\n")
|   reduce2tex1(lr(gamma,nega(a)::delta),apstr) =
(InRuOne(lr(reorg(a::gamma),reorg(delta))),
"\\infer[␣\\neg␣K␣]"^"{"^Seq2te(gamma)^
"\\Rightarrow␣"^Seq2te(nega(a)::delta)^"}\n")
|   reduce2tex1(lr(gamma,conj(a,b)::delta),apstr) =
(InRuTwo(lr(reorg(gamma),reorg(a::delta)),
lr(gamma,b::delta)),
"\\infer[␣\\sqcap␣␣K␣]"^"{"^Seq2te(gamma)^
"\\Rightarrow␣"^Seq2te(conj(a,b)::delta)^"}\n")
|   reduce2tex1(lr(cuan(r,a)::gamma,delta),apstr) =
(InRuOne(lr(reorg(a::gamma),reorg(delta))),
"\\infer[\\forall␣"^rol2tex(r)^"␣A]"^
"{"^Seq2te(cuan(r,a)::gamma)^
"\\Rightarrow␣"^Seq2te(delta)^"}\n")
|   reduce2tex1(any,apstr) = (decide(any),apstr);
```

(*Aplica la regla CuanK siempre que sea posible*)

```
fun reduce2tex2(lr(gamma,delta),apstr) =
if esAplicableCuank(lr(gamma,delta))
then
let val(cuan(r,c)::nil) = delta
in
```

```
(InRuOne(lr(gamma,c::nil)),
"\\infer[\\forall␣"^rol2tex(r)^"␣K]"^
"{"^Seq2te(gamma)^"\\Rightarrow␣"^
Seq2te(delta)^"}\n")
end
else reduce2tex1(lr(gamma,delta),apstr);
```

(*Aplica la regla Ax siempre que sea posible*)

```
fun reduce2tex(lr(gamma,delta),apstr) =
if intersect(gamma,delta)
then (Ax(lr(gamma,delta)),apstr)
else reduce2tex2(lr(gamma,delta),apstr);
```

(*Escribe recursivamente la prueba en código latex*)

```
fun prueba2tex(Ax(lr(l,r)),apstr) =
if  intersect(l,r)
then
    if apstr="" then "\\infer[Ax]{"^Seq2te(l)^
    "\\Rightarrow␣"^Seq2te(r)^"}\n{}"
    else apstr^"{\\infer[Ax]{"^Seq2te(l)^
    "\\Rightarrow␣"^Seq2te(r)^"}\n{}}"
else
    if apstr="" then "\\infer{"^Seq2te(l)^
    "\\Rightarrow␣"^Seq2te(r)^"}\n{}"
    else apstr^"{\\infer{"^Seq2te(l)^
    "\\Rightarrow␣"^Seq2te(r)^"}\n{}}"
|   prueba2tex(InRuOne(up),apstr) =
if apstr=""
then prueba2tex(reduce2tex((up),""))
else apstr^"{"^prueba2tex(reduce2tex((up),""))^"}"
|   prueba2tex(InRuTwo(up1,up2),apstr) =
if apstr=""
then prueba2tex(reduce2tex((up1),""))^"\n"^
prueba2tex(reduce2tex((up2),""))^"\n"
else apstr^"{"^prueba2tex(reduce2tex((up1),""))^
"␣&␣"^prueba2tex(reduce2tex((up2),""))^"}\n";
```

(*Ésta función ya nos imprime todos los pasos de la cadena para llevarlo a latex y compilar*)

```
fun gentzenProof(prop) =
```

```
"\\documentclass{article}"^"\n"^
"\\usepackage{proof,amsmath,amssymb,amscd}"^"\n"^
"\\begin{document}"^"\n"^
"\\[\\text{Prueba en L\\'{o}gica }ALCS4
\\text{ por Secuentes de Gentzen de:}\\]"^
"\\["^conc2lat(prop)^"\\]"^"\n"^"\n"^
"\\["^prueba2tex(InRuOne(lr([],[prop])),"")^"\\]"^
"\n"^"\\end{document}";
```

(*Flujo de datos a un archivo tex*)

```
load ("BasicIO");
open BasicIO;
val pruebaDeLaProposicion =
cuan(role("s"),conc("C"));
val ostreamId = open_out
("C:\\DemostradoresAutomaticos\\
JAVA\\ALCS4\\ALCS4.tex");
output
(ostreamId,gentzenProof(pruebaDeLaProposicion));
close_out(ostreamId);
```

(*Compilación de nuestro archivo tex en pdf*)

```
load("OS");
open OS;
Process.system "pdflatex ALCS4.tex -output-directory
=C:\\DemostradoresAutomaticos\\JAVA\\ALCS4 ";
Process.system "C:\\DemostradoresAutomaticos\\
JAVA\\ALCS4\\ALCS4.pdf";
```

A.2. Sistema de árboles de hipersecuentes para $\mathcal{ALC}$

(*Tipo de datos para escribir conceptos*)

```
datatype Roles=  role of string;
datatype Conc= conc of string |
               nega of Conc |
               conj of Conc * Conc|
               cuan of Roles * Conc;
```

```
(* Alias para secuencia de conceptos *)
type Seq = Conc list;

(*Tipo de datos para escribir secuentes*)
datatype Sequent = lr of Seq * Seq;

(*Tipo de datos para escribir árboles de hipersecuentes *)
datatype THS =
hyper of Sequent * ((Roles * THS) list);

(*Tipo de datos para escribir el árbol de prueba *)
datatype SistemaG = AXITHS of THS
| InRuOne of THS
| InRuTwo of THS * THS;

(*Determina si un elemento está en una lista *)
fun esta(x,nil) = false
|   esta(x,c::r) = if x=c
                   then true
                   else esta(x,r);

(*Determina si las dos lista de un secuente se intersectan*)
fun intersecta(lr(nil,list))= false
|   intersecta(lr(c::r,list))=
if esta(c,list)
then true
else intersecta(lr(r,list));

(*Determina si se trata de un árbol de hipersecuentes inicial *)
fun isIths(hyper(s,nil))=
intersecta(s)
|   isIths(hyper(s,(rolc,c)::r))=
intersecta(s) orelse
isIths(c) orelse
isIths(hyper(s,r));
```

```
(*Elimina elementos repetidos de una lista*)

fun delRep(nil)=nil
|   delRep(list)= let val(c1::r1)=rev(list)
                  in if esta(c1,r1)
                     then delRep(rev(r1))
                     else delRep(rev(r1))@[c1] end;

(*Simplifica un secuente, eliminando los elementos repetidos de cada
lado*)

fun delRepSeq(lr(g,d))= lr(delRep(g),delRep(d));

(*Aplica una función a la primer coordenada de los elementos de una
lista*)

fun aplRec(fun1,nil) = nil
|   aplRec(fun1,(x,c)::r) =
[(x,fun1(c))]@aplRec(fun1,r);

(*Simplifica un hipersecuente eliminando elemento repetidos de cada
secuente*)

fun delRepHip(hyper(seq1,nil))=
hyper(delRepSeq(seq1),nil)
|   delRepHip(hyper(seq1,list1))=
hyper(delRepSeq(seq1),aplRec(delRepHip,list1));

(*Reorganiza una lista enviando las cuantificaciones al final*)

fun reorgcuan(nil,pa)=pa
|   reorgcuan(cuan(x)::r,pa)=
    reorgcuan(r,pa@[cuan(x)])
|   reorgcuan(c::r,pa)=
    reorgcuan(r,c::pa);

(*Reorganiza una lista enviando los conceptos atómicos al final*)

fun reorgconc(nil,pa)=pa
|   reorgconc(conc(a)::r,pa)=
    reorgconc(r,pa@[conc(a)])
|   reorgconc(list,pa)= list@pa;
```

(*Reorganiza una lista enviando los conceptos atómicos al final, y en penúltimo lugar las cuantificaciones*)

```
fun reorg(lista)=
    reorgconc(reorgcuan(lista ,nil),nil);
```

(*Reorganiza un secuente enviando los conceptos atómicos al final, y en penúltimo lugar las cuantificaciones*)

```
fun reorgSeq(lr(g,d))= lr(reorg(g),reorg(d));
```

(*Reorganiza un hipersecuente enviando los conceptos atómicos al final, y en penúltimo lugar las cuantificaciones*)

```
fun reorgHyper(hyper(s,list1))=
hyper(reorgSeq(s),aplRec(reorgHyper,list1));
```

(*Aplica la regla NegA a un secuente*)

```
fun NegASeq(lr(nil,d)) = lr(nil,d)
|   NegASeq(lr(nega(prop)::g,d))=
    lr(g,d@[prop])
|   NegASeq(lr(g,d))= lr(g,d);
```

(*Aplica la regla NegA a un hipersecuente *)

```
fun NegAHyp(hyper(s,nil)) = hyper(NegASeq(s),nil)
|   NegAHyp(hyper(s1,list1)) =
hyper(NegASeq(s1),aplRec(NegAHyp,list1));
```

(*Determina si es aplicable la regla NegA a un secuente*)

```
fun isNegASeq(lr(nega(conc1)::g,d)) = true
|   isNegASeq(lr(g,d)) = false;

fun segCord(nil) = nil
|   segCord((x,y)::r) = [y]@segCord(r);

fun segCorOrd(nil) = false
|   segCorOrd(c::r) = c orelse segCorOrd(r);
```

(*Determina si es aplicable la regla NegA a un hipersecuente*)

```
fun isNegAHyp(hyper(s,nil)) = isNegASeq(s)
|   isNegAHyp(hyper(s,list1)) = isNegASeq(s)
orelse segCorOrd(segCord(aplRec(isNegAHyp,list1)));
```

(*Aplica la regla NegK a un secuente*)

```
fun NegKSeq(lr(g,nil)) = lr(g,nil)
|   NegKSeq(lr(g,nega(prop)::d))= lr(g@[prop],d)
|   NegKSeq(lr(g,d))= lr(g,d);
```

(*Aplica la regla NegK a un hipersecuente*)

```
fun NegKHyp(hyper(s,nil)) = hyper(NegKSeq(s),nil)
|   NegKHyp(hyper(s1,list1)) =
hyper(NegKSeq(s1),aplRec(NegKHyp,list1));
```

(*Determina si es aplicable la regla NegK a un secuente*)

```
fun isNegKSeq(lr(g,nega(conc1)::d)) = true
|   isNegKSeq(lr(g,d)) = false;
```

(*Determina si es aplicable la regla NegK a un hipersecuente*)

```
fun isNegKHyp(hyper(s,nil)) = isNegKSeq(s)
|   isNegKHyp(hyper(s,list1)) = isNegKSeq(s) orelse
segCorOrd(segCord(aplRec(isNegKHyp,list1)));
```

(*Aplica la regla ConjA a un secuente*)

```
fun ConASeq(lr(nil,d)) = lr(nil,d)
|   ConASeq(lr(conj(prop1,prop2)::g,d))=
    lr(prop1::prop2::g,d)
|   ConASeq(lr(g,d))= lr(g,d);
```

(*Aplica la regla ConA a un hipersecuente*)

```
fun ConAHyp(hyper(s,nil)) = hyper(ConASeq(s),nil)
|   ConAHyp(hyper(s1,list1)) =
    hyper(ConASeq(s1),aplRec(ConAHyp,list1));
```

(*Determina si es aplicable la regla ConjA a un secuente*)

```
fun isConjASeq(lr(conj(a,b)::g,d)) = true
|   isConjASeq(lr(g,d)) = false;
```

```
   (*Determina si es aplicable la regla ConjA a un hipersecuente*)

fun isConjAHyp(hyper(s,nil)) = isConjASeq(s)
|   isConjAHyp(hyper(s,list1)) = isConjASeq(s)
orelse segCorOrd(segCord(aplRec(isConjAHyp,list1)));


   (*Aplica la regla ConjK1 a un secuente*)

fun ConK1Seq(lr(g,nil)) = lr(g,nil)
|   ConK1Seq(lr(g,conj(prop1,prop2)::d))=
    lr(g,prop1::d)
|   ConK1Seq(lr(g,d))= lr(g,d);


   (*Aplica la regla ConK2 a un secuente*)

fun ConK2Seq(lr(g,nil)) = lr(g,nil)
|   ConK2Seq(lr(g,conj(prop1,prop2)::d))=
    lr(g,prop2::d)
|   ConK2Seq(lr(g,d))= lr(g,d);


   (*Aplica la regla ConK1 a un hipersecuente*)

fun ConK1Hyp(hyper(s,nil)) = hyper(ConK1Seq(s),nil)
|   ConK1Hyp(hyper(s1,list1)) =
    hyper(ConK1Seq(s1),aplRec(ConK1Hyp,list1));


   (*Aplica la regla ConK2 a un hipersecuente*)

fun ConK2Hyp(hyper(s,nil)) = hyper(ConK2Seq(s),nil)
|   ConK2Hyp(hyper(s1,list1)) =
    hyper(ConK2Seq(s1),aplRec(ConK2Hyp,list1));


   (*Determina si es aplicable la regla Conjk a un secuente*)

fun isConjKSeq(lr(g,conj(a,b)::d)) = true
|   isConjKSeq(lr(g,d)) = false;


   (*Determina si es aplicable la regla ConjK a un hipersecuente*)

fun isConjKHyp(hyper(s,nil)) = isConjKSeq(s)
|   isConjKHyp(hyper(s,list1)) = isConjKSeq(s)
orelse segCorOrd(segCord(aplRec(isConjKHyp,list1)));
```

(* Por último las reglas de cuantificación*)

(*Extrae los roles al primer nivel de un secuente una lista de hipersecuentes *)

```
fun listaRoles(nil) = nil
|   listaRoles((rol1,c1)::r1) =
    [rol1]@listaRoles(r1);

fun listaRoles2(nil) = nil
|   listaRoles2((rol1,c1)::r1) =
    [c1]@listaRoles2(r1);

fun pegalos(nil)=nil
|   pegalos(c::r)= c@pegalos(r);
```

(*Da la lista de los roles a los que se les aplica CuanA*)

```
fun rolesCuanAHyp(hyper(lr(cuan(r1,c1)::g,d),list1))
= if esta(r1,listaRoles(list1))
  then
[r1]@pegalos(listaRoles2(
      aplRec(rolesCuanAHyp,list1)))
  else
[]@pegalos(listaRoles2(aplRec(rolesCuanAHyp,list1)))
|   rolesCuanAHyp(hyper(lr(g,d),list1)) =
[]@pegalos(listaRoles2(
   aplRec(rolesCuanAHyp,list1)));
```

(*Determina si es aplicable la regla CuanA a un hipersecuente*)

```
fun isCuanAHyp(h1) = if rolesCuanAHyp(h1)=nil
                     then false else true;

fun agregaCuan(cuan(rol1,c1), nil)= nil
|   agregaCuan(cuan(rol1,c1),
(r1,hyper(lr(g,d),list1))::list2)=
if rol1=r1 then [(r1,hyper(lr([c1]@g,d),list1))]@
                agregaCuan(cuan(rol1,c1),list2)
           else [(r1,hyper(lr(g,d),list1))]@
                agregaCuan(cuan(rol1,c1),list2);
```

(*Aplica la regla CuanA a un hipersecuente y obtiene el rol al que se le aplico*)

```
fun CuanAHyp(hyper(lr(cuan(r1,c1)::g,d),list1))=
if esta(r1,listaRoles(list1))
then hyper(lr(g,d),agregaCuan(cuan(r1,c1),
            aplRec(CuanAHyp,list1)))
else hyper(lr(cuan(r1,c1)::g,d),
      aplRec(CuanAHyp,list1))
|   CuanAHyp(hyper(lr(g,d),list1)) =
hyper(lr(g,d),aplRec(CuanAHyp,list1));

fun agregaCuanK(cuan(rol1,c1),list1)=
list1@[(rol1,hyper(lr([],[c1]),[]))];
```

(*Aplica la regla CuanK a un hipersecuente y obtiene el rol al que se le aplico*)

```
fun CuanKHyp(hyper(lr(g,cuan(rol1,c1)::r),list1))=
hyper(lr(g,r),agregaCuanK(cuan(rol1,c1),
       aplRec(CuanKHyp,list1)))
|   CuanKHyp(hyper(lr(g,r),list1)) =
hyper(lr(g,r),aplRec(CuanKHyp,list1));
```

(*Da la lista de los roles a los que se les aplica CuanK*)

```
fun rolesCuanKHyp(hyper(lr(g,cuan(r1,c1)::d),list1))
=[r1]@pegalos(listaRoles2(
       aplRec(rolesCuanKHyp,list1)))
|   rolesCuanKHyp(hyper(lr(g,d),list1)) =
[]@pegalos(listaRoles2(
   aplRec(rolesCuanKHyp,list1)));
```

(*Determina si es aplicable la regla CuanK a un hipersecuente*)

```
fun isCuanKHyp(h1) = if rolesCuanKHyp(h1)=nil
                     then false else true;
```

(*Escritura en codigo latex *)
(*Transcribe los roles a código latex*)

```
fun rol2tex(role(a))= a;
```

(*Transcribe una lista de roles a código latex*)

```
fun list2Rol2Tex(nil)= ""
|   list2Rol2Tex([any])= rol2tex(any)
```

```
|    list2Rol2Tex(c::r)=
     rol2tex(c)^","^list2Rol2Tex(r);
```

(*Transcribe los conceptos a código latex*)

```
fun conc2lat(conc(a))= a
|    conc2lat(conj(a,b))=
"("^conc2lat(a)^"\\sqcap␣"^conc2lat(b)^")"
|    conc2lat(nega(t))=
"("^"\\neg␣"^conc2lat(t)^")"
|    conc2lat(cuan(role(r),a))=
"("^"\\forall␣"^r^"."^conc2lat(a)^")";
```

(*Transcribe una lista a código latex*)

```
fun Seq2te(nil) = "␣"
|    Seq2te([algo]) = conc2lat(algo)
|    Seq2te(c::r) =
conc2lat(c)^","^Seq2te(r);
```

(*Transcribe un secuente a código latex*)

```
fun sequent2Lat(lr(g,d))=
Seq2te(g)^"␣\\Rightarrow␣"^Seq2te(d);

fun pareja2Lat(nil)=""
|    pareja2Lat((role(r),cad1)::nil) =
r^"␣:␣"^"(␣"^cad1^"␣)"
|    pareja2Lat((role(r),cad1)::list1) =
r^"␣:␣"^"(␣"^cad1^"␣)"^"␣;␣"^pareja2Lat(list1);
```

(*Transcribe un hipersecuente a código latex*)

```
fun hyp2Lat(hyper(s,nil))= sequent2Lat(s)
|    hyp2Lat(hyper(s,list1))=
sequent2Lat(s)^"␣/␣"^
pareja2Lat(aplRec(hyp2Lat,list1));
```

(*Simplifica y reorganiza un hipersecuente*)

```
fun simpReorg(h1)= reorgHyper(delRepHip(h1));
```

(*Aplica las reglas de hipersecuentes y los va reduciendo*)

```
fun reduce2tex(h1,apstr) =
if isIths(h1)
then (AXITHS(h1),apstr)
else if isNegAHyp(h1)
   then (InRuOne(NegAHyp(h1)),"\\infer[\\neg␣A␣]"
   ^"{"^hyp2Lat(h1)^"}\n")
   else if isNegKHyp(h1)
      then (InRuOne(NegKHyp(h1)),
      "\\infer[\\neg␣K␣]"^"{"^hyp2Lat(h1)^"}\n")
      else if isConjAHyp(h1)
         then (InRuOne(ConAHyp(h1)),
         "\\infer[\\sqcap␣A␣]"^
         "{"^hyp2Lat(h1)^"}\n")
         else if isConjKHyp(h1)
            then (InRuTwo(ConK1Hyp(h1),
            ConK2Hyp(h1)),"\\infer[\\sqcap␣A␣]"
            ^"{"^hyp2Lat(h1)^"}\n")
            else if isCuanKHyp(h1)
               then (InRuOne(CuanKHyp(h1)),
               "\\infer[\\forall␣"^
               list2Rol2Tex(rolesCuanKHyp(h1))
               ^"␣K␣]"^"{"^hyp2Lat(h1)^"}\n")
               else if isCuanAHyp(h1)
                  then (InRuOne(CuanAHyp(h1)),
                  "\\infer[\\forall␣"^
                  list2Rol2Tex(rolesCuanAHyp(h1))
                  ^"␣A␣]"^"{"^hyp2Lat(h1)^"}\n")
                  else (AXITHS(h1),apstr);

fun reduce2tex2(h1,apstr) =
reduce2tex(simpReorg(h1),apstr);
```

(* Escribe recursivamente la prueba en codigo latex *)

```
fun prueba2tex(AXITHS(h1),apstr) =
if  isIths(h1)
then
    if apstr=""
  then "\\infer[AXTHS]{"^hyp2Lat(h1)^"}\n{}"
  else apstr^"{\\infer[AXTHS]{"^hyp2Lat(h1)^"}\n{}}"
else
    if apstr=""
  then "\\infer{"^hyp2Lat(h1)^"}\n{}"
  else apstr^"{\\infer{"^hyp2Lat(h1)^"}\n{}}"
```

```
|    prueba2tex(InRuOne(up),apstr) =
if apstr=""
then prueba2tex(reduce2tex2((up),""))
else apstr^"{"^prueba2tex(reduce2tex2((up),""))^"}"
|    prueba2tex(InRuTwo(up1,up2),apstr) =
if apstr=""
then prueba2tex(reduce2tex2((up1),""))^"\n"^
prueba2tex(reduce2tex2((up2),""))^"\n"
else apstr^"{"^prueba2tex(reduce2tex2((up1),""))^
" & "^prueba2tex(reduce2tex2((up2),""))^"}\n";
```

(*Finalmente la compilación*)
(*Imprime todos los pasos de la cadena para llevarlo a latex y compilar*)

```
fun gentzenProof(conc1) =
"\\documentclass{article}"^"\n"^
"\\usepackage{proof,amsmath,amssymb,amscd}"^"\n"^
"\\begin{document}"^"\n"^
"\\[ \\text{Prueba en L\\'{o}gica Descriptiva }
\\mathcal{ALC} \\text{ por Hipersecuentes de:} \\]"^
"\\[ "^conc2lat(conc1)^"\\]"^"\n"^"\n"^"\\[ "^
prueba2tex(InRuOne(hyper(lr([],[conc1]),[])),"")
^"\\]"^"\n"^"\\end{document}";
```

(*Flujo de datos a un archivo tex*)

```
load ("BasicIO");
open BasicIO;
val pruebaDelConcepto = cuan(role "R",conc "C");
val ostreamId = open_out("C:\\
DemostradoresAutomaticos\\JAVA\\ALC\\hiper.tex");
output(ostreamId,gentzenProof(pruebaDelConcepto));
close_out(ostreamId);
```

(*Compilación de nuestro archivo tex en pdf*)

```
load("OS");
open OS;
Process.system "pdflatex hiper.tex -output-directory
=C:\\DemostradoresAutomaticos\\JAVA\\ALC ";

Process.system "C:\\DemostradoresAutomaticos
\\JAVA\\ALC\\hiper.pdf";
```

Bibliografía

[1] Arnon Avron. The method of hypersequents in the proof theory of propositional non-classical logics. In *IN LOGIC: FROM FOUNDATIONS TO APPLICATIONS, EUROPEAN LOGIC COLLOQUIUM*, pages 1–32. Oxford University Press, 1994.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[3] Everardo Bárcenas, Pierre Genevès, Nabil Layaïda, and Alan Schmitt. Query reasoning on trees with types, interleaving, and counting. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 718–723, 2011.

[4] Everardo Bárcenas and Jesús Lavalle. Expressive reasoning on tree structures: Recursion, inverse programs, Presburger constraints and nominals. In *12th Mexican International Conference on Artificial Intelligence*, pages 80–91, 2013.

[5] Everardo Bárcenas and Jesús Lavalle. Global numerical constraints on trees. *Logical Methods in Computer Science*, 10(2), 2014.

[6] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[7] Patrick Blackburn, Johan F. A. K. van Benthem, and Frank Wolter. *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. Elsevier Science Inc., New York, NY, USA, 2006.

[8] Gerhard Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1934.

[9] Michael J. C. Gordon, Robin Milner, L. Morris, Malcolm C. Newey, and Christopher P. Wadsworth. A metalanguage for interactive proof in LCF. In *Conference Record of the Fifth Annual ACM*

Symposium on Principles of Programming Languages, pages 119–130, 1978.

[10] Brian Hill and Francesca Poggiolesi. A contraction-free and cut-free sequent calculus for propositional dynamic logic. *Studia Logica*, 94(1):47–72, 2010.

[11] Ian Horrocks and Peter Patel-Schneider. Reducing owl entailment to description logic satisfiability. *Web Semant.*, 1(4):345–357, October 2004.

[12] Marvin Minsky. A framework for representing knowledge. Technical report, Cambridge, MA, USA, 1974.

[13] Juan-Pablo Muñoz-Toriz, Ivan Martínez-Ruiz, and José Arrazola-Ramírez. On automatic theorem proving with ML. In *13th Mexican International Conference on Artificial Intelligence*, 2014.

[14] Francesca Poggiolesi. *The Method of Tree-Hypersequents for Modal Propositional Logic*, pages 31–51. Springer Netherlands, Dordrecht, 2009.

[15] Alexandre Rademaker. *A Proof Theory for Description Logics*. Springer Briefs in Computer Science. Springer, 2012.

[16] Alexandre Rademaker. *A Proof Theory for Description Logics*. SpringerBriefs in Computer Science. Springer, 2012.

[17] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *In Proc. of IJCAI-91*, pages 466–471, 1991.

[18] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26, 1991.

[19] John F. Sowa. *Semantic networks*. Wiley, 1987.

[20] Y. Sun and Y. Sui. A gentzen system for the description logic. In *2014 9th International Conference on Computer Science Education*, pages 99–102, Aug 2014.

[21] Stephan Tobies. Pspace reasoning for graded modal logics. *Journal of Logic and Computation*, 11(1):85–106, 2001.