R. 27042

i1892557i

UNIVERSIDAD DE SEVILLA
SECRETARIA GENERAL

Queda registrada esta Tesis Doctoral al folio 62 número 195 del libro correspondiente.
Sevilla, 13 JUN. 2002
El Jefe del Negociado de Tesis,

UNIVERSIDAD DE SEVILLA
Biblioteca de Matemáticas
Fondo Antiguo

DCA

Universidad de Sevilla
Departamento de Ciencias de la Computación
e Inteligencia Artificial

043
388

Operadores de generalización para el aprendizaje clausal

UNIVERSIDAD DE SEVILLA
Depositado en la Secretaría del Dpto. de Ciencias de la Computación e I. Artificial de esta Universidad desde el día 6-Junio-2002 hasta el día 2-Julio-2002
Sevilla 3 de Julio de 2002
EL DIRECTOR DEL DPTO.,
Fdo: José A. Alonso Jiménez

Memoria presentada por
Miguel Ángel Gutiérrez Naranjo
para optar al grado de
Doctor en Matemáticas
por la Universidad de Sevilla

M.A. Gutiérrez N.

Miguel Ángel Gutiérrez Naranjo

V. B. Directores

José A. Alonso

D. José Antonio Alonso Jiménez

D. Joaquín Borrego Díaz

Sevilla, Junio de 2002

A mis padres

Índice General

Índice de Figuras

Capítulo 1

Introducción

En los últimos años, el interés de la comunidad científica por la adquisición de conocimiento de manera automática ha aumentado progresivamente. La inmensa cantidad de datos a los que se enfrenta cualquier investigador de las ciencias sociales, los requerimientos empresariales y en general, la necesidad de extraer información a partir del inmenso flujo de información que nos aborda todos los días exige una respuesta desde las Ciencias de la Computación.

No es de extrañar que desde hace varios años se hayan realizado grandes esfuerzos en automatizar el proceso de la *extracción del conocimiento* a partir de bases de datos. En este proceso, arduo y complejo, intervienen muchos factores y son muchas las técnicas mediante las cuales nos aproximamos al objetivo. Cuestiones tales como el filtrado de la información, la unificación de los datos procedentes de distintas fuentes en un único sistema de representación o el preprocesado de los datos hacen necesaria la aportación de muy diversas disciplinas.

En esta memoria fijamos nuestra atención en los procesos de extracción de conocimiento en los que la información está representada en un lenguaje clausal. Esta disciplina se conoce como Programación Lógica Inductiva (PLI) y se puede definir de manera informal como la intersección del Aprendizaje Automático y la Programación Lógica.

Nuestro objetivo es profundizar en los procesos de *generalización* asociados a todo proceso de aprendizaje y en cómo podemos sistematizar ese paso de lo particular a lo general cuando el aprendizaje se realiza sobre un lenguaje clausal.

Pero, ¿qué es aprender? ¿cuáles son los mecanismos por los que adquirimos nuevos conocimientos? El aprendizaje es un proceso complicado, lleno de matices y singularidades. Aprendemos comportamientos, habilidades y conocimientos, el

aprendizaje puede ser guiado por un profesor o por descubrimiento, puede ser espontáneo, sistemático, ...

Es difícil dar una definición de *aprendizaje* que englobe todos los aspectos, incluyendo la diversidad generada por la naturaleza del *sujeto* que realice el aprendizaje, esto es, contemplando la posibilidad de que el aprendizaje lo realice un humano, un animal o una máquina.

Quizá una buena aproximación para empezar sea la apuntada por T. M. Mitchell: *"Aprender es mejorar con la experiencia"*. Es un buen punto de partida, aunque para tratar formalmente el aprendizaje necesitamos definir cuestiones como a *qué* llamamos experiencia, *cómo* la representamos o *cuándo* se produce una mejora.

El modelo de aprendizaje seguido en Programación Lógica Inductiva es el que siguen las Ciencias Experimentales en la creación de teorías. Partimos de un conocimiento básico y una serie de experimentos en los que hemos obtenido distintos resultados. El objetivo en estos casos es establecer una *hipótesis* que junto con el conocimiento previo explique el resultado de los experimentos. El objetivo último de esta hipótesis no es sólo explicar los experimentos realizados sino usarla para predecir el resultado de futuros experimentos. La hipótesis se mantendrá como teoría aceptada por la comunidad científica mientras que los posteriores experimentos realizados la corroboren. Si un nuevo experimento contradice el estado actual de la teoría, esta debe ser revisada y modificada para poder explicar esta nueva situación.

Este modelo de aprendizaje es el que seguimos en PLI. Tomamos un conocimiento base y un conjunto de ejemplos positivos y negativos. Se los suministramos al sistema junto con datos relativos a las posibles hipótesis, para que devuelva una teoría que, junto con el conocimiento base, explique la clasificación de los resultados experimentales (ejemplos). Tanto el conocimiento base como los ejemplos o las hipótesis están expresados en un lenguaje formal (en nuestro caso, el lenguaje clausal).

El objetivo de esta memoria es el estudio de los procesos de generalización, el paso de lo particular a lo general, en este marco y con esta representación.

Empezamos nuestro estudio con un capítulo (Cap. 2) dedicado al aprendizaje de conceptos y a una breve reflexión sobre qué sentido tiene plantearnos la automatización de los procesos de generalización.

El proceso de generar explicaciones a partir de observaciones –*la abducción*– y de asignar un grado de *certeza* a la hipótesis elegida –*justificación*– son cuestiones

debatidas en el ámbito de la Filosofía durante siglos. Presentamos aquí algunas pinceladas de los puntos de vista más significativos en este campo, como los de Hume, Peirce o Popper entre otros. El debate sobre *cómo* el investigador desarrolla una teoría *nueva* a partir de una cantidad finita de experimentos o el grado de validez que asignamos a las hipótesis es un debate abierto en Filosofía de la Ciencia.

La presente memoria hunde sus raíces en esas cuestiones relativas al paso de lo particular a lo general, en ese salto que representa realizar una hipótesis y aventurarnos a predecir el resultado de un experimento a partir del análisis de otros previos. Pero nuestro objetivo no es ahondar en ese debate. Como decía Antonio Machado en sus *Proverbios*,

> ¿Dices que nada se crea?
> Alfarero a tus cacharros
> Haz tu copa y no te importe
> si no puedes hacer barro.

Por tanto, en la segunda mitad de este capítulo 2 volvemos a nuestros cacharros y dirigimos nuestra mirada hacia las Ciencias de la Computación.

En las secciones finales del capítulo hacemos un breve repaso de la teoría del aprendizaje de conceptos desde un punto de vista computacional, haciendo hincapié en el planteamiento conjuntista que nos acompañará en los capítulos posteriores: Un concepto es el conjunto de sus instancias y puesto que hemos adoptado un lenguaje clausal las instancias (los ejemplos) serán átomos cerrados de nuestro lenguaje. Terminamos este capítulo 2 con una introducción a la Programación Lógica Inductiva, disciplina en la que se enmarca esta memoria, y presentamos brevemente algunas de las técnicas empleadas en sus sistemas.

La aportación original de esta memoria comienza en el capítulo 3. El estudio de los operadores de generalización clausales viene motivado ante la diversidad de técnicas de generalización que encontramos en los sistemas de PLI. En la literatura es común encontrar algoritmos de generalización que han sido diseñados para resolver un problema concreto en un determinado sistema. Ante esta dispersión surge la necesidad de un estudio de los operadores de generalización en sí mismos, alejado de las características particulares de tal o cual sistema. Nuestro objetivo en este tercer capítulo es presentar una nueva familia de operadores, los *Operadores Clausales* (Definición 3.15), con la propiedad de ser operadores universales, esto es, que dadas dos cláusulas cualesquiera C_1 y C_2 podemos alcanzar C_2 desde

C_1 mediante la sucesiva aplicación de estos operadores (Teorema 3.22).

De esta manera podemos abordar el problema de la generalización de cláusulas mediante el uso exclusivo de estos operadores, i.e., para generalizar una cláusula respecto a cualquier orden no tenemos que buscar ningún operador fuera de los definidos en este capítulo, sino que bastará con elegir los operadores adecuados dentro de esta familia.

La definición de estos operadores se realiza a varios niveles ya que debemos compaginar la inserción de términos en otros términos, en literales y en cláusulas. Al final del capítulo apuntamos otra de las ideas básicas que nos acompañará en capítulos posteriores. La idea de proximidad, medida a partir del número mínimo de operadores necesarios para llevar una cláusula en otra.

En el capítulo 4 presentamos los *Operadores de Aprendizaje para la Subsunción (OAS)* (Definición 4.15). Una vez definidos los operadores clausales en el capítulo anterior determinamos un orden entre las cláusulas para poder hablar de *generalización.* El orden elegido en este capítulo es el orden de subsunción, presentado por Plotkin en 1970. Este es el orden usado en la mayoría de los sistemas de Programación Lógica Inductiva. Los operadores definidos en este capítulo, los OAS, son un subconjunto de los operadores definidos en el capítulo anterior, esto es, son *operadores clausales* y su principal propiedad es que representan una *caracterización* mediante operadores de la relación de subsunción entre cláusulas, esto es, dadas dos cláusulas cualesquiera C_1 y C_2, se verifica que C_1 subsume a C_2 si y sólo si podemos obtener C_1 a partir de C_2 mediante la aplicación de una cadena de OAS (Corolario 4.21).

La relación de subsunción es una relación cualitativa, no obstante, la propiedad fundamental de los OAS nos lleva a la idea de *cuantificar* la proximidad entre dos cláusulas según la relación de subsunción como el menor número de operadores (OAS) necesarios para llevar una cláusula en otra (Definición 4.24). La función que nos devuelve el número de operadores no es simétrica, puesto que la relación de subsunción no lo es, por lo que no podemos hablar de distancia, sino de *quasi-distancia.*

En lugar de ser un inconveniente, el uso de esta distancia orientada o quasi-distancia no sólo nos proporciona información cuantitativa sobre la proximidad entre las cláusulas sino también cualitativa sobre si se verifica o no la relación de subsunción.

Dedicamos la parte final del capítulo al cálculo efectivo de esta quasi-distancia y presentamos un algoritmo de cálculo (Sección 4.9). El principal inconvenien-

te de esta quasi-distancia es que decidir la subsunción entre dos cláusulas es un problema NP-completo. De todos modos, los tiempos de cálculo obtenidos en experimentos con cláusulas de tipo medio en programas Prolog justifican su uso. El capítulo termina con un resultado sobre las cotas obtenidas para la quasi-distancia que nos dan una estimación inmediata sobre el intervalo en el que se encuentra (bajo hipótesis de subsunción) y que puede ser usado para una estimación eficiente en conjuntos de cláusulas de gran tamaño (Teorema 4.43).

El capítulo 5 lo dedicamos a estudiar el orden de consecuencia ($\models$) entre cláusulas y su relación con los operadores clausales. La relación de consecuencia es la segunda relación usada en aprendizaje clausal. Es una relación más fuerte que la relación de subsunción, ya que si la cláusula C subsume a la cláusula D entonces C implica D, pero el recíproco no es cierto en general (Ejemplo 5.9). No obstante la relación de subsunción es la relación natural usada en sistemas de aprendizaje clausal y la relación de consecuencia se usa rara vez y bajo fuertes restricciones. La explicación es sencilla: La relación de consecuencia entre cláusulas no es decidible, mientras que la relación de subsunción sí lo es.

En nuestro estudio sobre la relación de consecuencia y los operadores de generalización nos apoyamos en el Teorema de Subsunción (Teorema 5.7), un teorema básico en Programación Lógica, que afirma que la relación de consecuencia puede descomponerse en la relación de derivación mediante resolución junto con la relación de subsunción. Usamos también un resultado clásico, el Lema de Gottlob (Lema 5.8). Este lema nos dice que si entre las cláusulas C y D se da la relación de consecuencia entonces se verifica la subsunción entre los literales positivos de ambas cláusulas, de un lado, y entre los negativos por otro lado.

Combinando ambos resultados se concluye que para poder generalizar cláusulas en el orden de consecuencia basta generalizar el orden de subsunción y el orden de derivación por resolución. Los operadores de generalización del orden de subsunción, los OAS, ya fueron estudiados en el capítulo 4 y por el Lema de Gottlob, para generalizar el orden de derivación por resolución tenemos que buscar operadores que generalicen de manera independiente los literales positivos y negativos de una cláusula. Por consiguiente, para poder definir operadores de generalización para la relación de derivación definimos unos *Operadores de Inversión Sesgados (OIS)* (Definición 5.13) que permiten generalizar los literales negativos y positivos de las cláusulas de manera independiente. De esta manera, mediante una combinación adecuada de Operadores de Aprendizaje para la Subsunción (OAS), que nos permiten generalizar la relación de subsunción, y Operadores de Inversión Segados

(OIS), con los que generalizamos la derivación por resolución, podemos obtener C a partir de D en el caso en que $C \models D$ y D no sea tautología (Corolario 5.18).

Los Operadores de Aprendizaje para la Subsunción (OAS) y los Operadores de Inversión Segados (OIS) tienen naturaleza muy distinta, puesto que están definidos para generalizar relaciones diferentes, no obstante podemos encontrar puntos en común que nos van a permitir definir un nuevo tipo de operadores: los *Operadores de Generalización Minimales, (OGM)* (Definición 5.19). Dedicamos la parte final del capítulo 5 a su estudio. Estos operadores representan las unidades mínimas de generalización clausal de manera que si C y D son cláusulas y están relacionadas por alguna de las tres relaciones estudiadas: subsunción, derivación o consecuencia, esto es $C \succeq D$, $C \vdash D$ o $C \models D$, entonces podemos obtener C a partir de D mediante una combinación apropiada de OGM (Corolario 5.28).

Terminamos el capítulo con una breve reflexión sobre distancias. Podríamos plantearnos el problema de asignar un valor cuantitativo a la relación de consecuencia entre cláusulas mediante una función que a cada par de cláusulas C y D le asignara la longitud de la cadena de OGM más corta que nos llevara D en C si $C \models D$ e infinito si $C \not\models D$, pero esta definición no tendría ningún sentido práctico puesto que la relación de implicación entre cláusulas no es decidible.

En el capítulo 6 presentamos una definición de subsunción entre programas. Nuestra definición y resultados extienden y precisan la noción de equivalencia entre programas basada en subsunción [Mah88]. En su artículo, M. J. Maher no define la subsunción entre programas, sólo la equivalencia entre programas basada en subsunción, pero del análisis de las demostraciones se deduce una definición de subsunción entre programas similar a nuestra propuesta.

Nuestros operadores permiten estudiar una definición *explícita* de la relación de subsunción con la que iniciamos el capítulo.

En las primeras secciones extendemos los Operadores Clausales para la Subsunción (OAS) del capítulo 4 a programas de manera natural y definimos de esta manera los *OAS compuestos* (Definición 6.4). Estos operadores, con ayuda de una función aplicador, llevan un programa sobre otro y, de manera análoga a como ocurría con los OAS, también representan una *caracterización* de la relación de subsunción entre programas (Corolario 6.12).

Tenemos por tanto que si P_1 y P_2 son dos programas lógicos vistos como conjuntos de cláusulas, P_1 subsume a P_2 si y sólo si podemos obtener algún subconjunto de P_1 aplicando a P_2 una cadena apropiada de OAS compuestos. Siguiendo la analogía con el capítulo 4, la longitud de la cadena más corta de

estos operadores entre dos programas también nos permite cuantificar la relación de subsunción entre programas. Nuevamente obtenemos una función que entra en el conjunto de *distancias débiles*, en este caso una *pseudo-quasi-distancia* que dota el conjunto de programas de estructura de *dominio cuantitativo* (Sección 6.3). Terminamos nuestro estudio sobre este dominio cuantitativo ofreciendo un método para el calculo efectivo de esa pseudo-quasi-distancia entre programas (Sección 6.4).

En las siguientes secciones nos restringimos a programas definidos. En el artículo de Maher antes citado encontramos algunos resultados que relacionan la subsunción entre programas con la semántica de programas definidos mediante el operador T_P de consecuencia de van Emden y Kowalski. Si definimos un concepto con el conjunto de todas sus instancias, la adaptación natural a la Programación Lógica es considerar un concepto como un subconjunto de la base de Herbrand de un lenguaje en el que todos los átomos comparten el mismo símbolo de predicado. Así, el menor modelo de Hebrand de un programa definido P puede considerarse como la unión de varios conceptos: Tantos como símbolos de predicado podamos encontrar en sus átomos.

Tiene sentido entonces establecer un orden de generalidad entre programas definidos asociado a la idea de aprendizaje: Dados dos programas definidos P_1 y P_2, diremos que P_1 es más general en el orden de aprendizaje que P_2 si el menor modelo de Herbrand de P_1 contiene al menor modelo de Herbrand de P_2 (Definición 6.25).

Por tanto, estableciendo un puente a través del operador T_P tenemos que si el programa definido P_1 subsume a P_2, o de manera equivalente, si mediante la aplicación a P_2 de OAS compuestos podemos obtener un subconjunto de P_1, entonces P_1 es más general que (o equivalente a) P_2 en el orden de aprendizaje (Proposición 6.30).

En la parte final de este capítulo 6 nos planteamos cuestiones relativas a procesos infinitos: ¿Podemos aplicar operadores de generalización indefinidamente? ¿Tiene sentido plantearse la existencia de procesos en los que obtengamos programas más y más generales en el orden de aprendizaje? La respuesta es afirmativa. Finalmente presentamos una sucesión infinita de programas cada uno de los cuales es estrictamente más general que el anterior tanto en el orden de subsunción como en el de aprendizaje (Ejemplo 6.35). Esto nos lleva a plantearnos cuestiones sobre procesos transfinitos a los que dedicamos el capítulo siguiente.

Llegamos así al capítulo 7 donde seguimos profundizando en el estudio de

procesos infinitos que iniciamos en el capítulo anterior. En las primeras secciones hacemos un repaso de distintos resultados conocidos relativos a cadenas infinitas relacionadas con el orden de subsunción, como el hecho de que el conjunto de literales de un lenguaje es noetheriano, esto es, no existe una cadena infinita de literales $\{L_n\}_{n\in\mathbb{N}}$ tal que L_{n+1} sea más general que L_n por subsunción, o el hecho conocido de la existencia de cadenas infinitas de cláusulas tanto ascendentes, esto es, donde la cláusula C_n subsume estrictamente a la cláusula C_{n+1}, como descendentes, donde la cláusula C_{n+1} subsume estrictamente a la cláusula C_n.

Fijamos nuestra atención en las cadenas infinitas descendentes, esto es, las cadenas en las que se produce una cantidad infinita de generalizaciones. Nuestra aportación original en esta primera parte del capítulo corresponde al estudio de propiedades comunes a todas estas cadenas (p.e. Teor. 7.6). En la literatura podemos encontrar algún ejemplo de este tipo de cadenas pero esta es la primera vez que se estudian propiedades generales.

En la segunda parte del capítulo abordamos el problema del *diámetro* del conjunto de cláusulas basado en la relación de subsunción. En la primera hemos visto que existen cadenas infinitas donde cada cláusula es más general que la anterior. Para algunas de estas sucesiones infinitas podemos encontrar una *cota*, esto es, una cláusula más general que todas las cláusulas de la sucesión. Esta cota a su vez puede ser la primera cláusula de otra sucesión infinita descendente. Si consideramos que cada una de esas cadenas infinitas representa una inmersión del ordinal ω en el conjunto de cláusulas, se define el diámetro como el supremo de todos los ordinales que pueden sumergirse en el conjunto de cláusulas (Definición 7.12).

El diámetro del conjunto de cláusulas por subsunción depende del lenguaje en el que expresemos las cláusulas. Finalizamos el capítulo demostrando que si el lenguaje tiene al menos un símbolo de predicado binario entonces el diámetro está acotado inferiormente por ω^2 (Teorema 7.13).

La memoria termina con un último capítulo, en el que presentamos las conclusiones y las distintas líneas que apuntamos como posibilidades para continuar profundizando en el estudio del aprendizaje clausal.

Como colofón, hemos añadido dos apéndices, el primero con una breve historia de los operadores de refinamiento y el segundo con las implementaciones para SWI-Prolog de las definiciones, algoritmos y relaciones más significativas presentadas en la memoria.

Capítulo 2

Preliminares

El aprendizaje es un proceso complejo y fascinante que todavía no comprendemos en su totalidad. Es materia de estudio de la Pedagogía, Psicología, Neurología y Filosofía entre otras disciplinas a las cuales se han unido, desde la segunda mitad del siglo XX, las Ciencias de la Computación.

A pesar de tratarse de disciplinas de naturaleza muy distinta, cualquier aproximación al aprendizaje desde cada una de ellas debe tener en cuenta ciertas líneas comunes:

- De un lado, los investigadores de estas disciplinas deben considerar temas como la representación del conocimiento, la organización en la memoria y su desarrollo y los mecanismos que nos permitan extraer conocimiento más allá del mero almacenamiento de datos.

- Por otro lado, tanto en las Ciencias de la Computación como en las demás disciplinas debe tenerse en cuenta que el aprendizaje aparece en cualquier dominio que conlleve inteligencia, entre los que podemos destacar planificación de las actividades a realizar, diagnóstico o lenguaje.

2.1 ¿Qué es aprender?

"El aprendizaje es uno de esos fenómenos del conocimiento que se resisten a una definición general." De esta forma elude Flach [Fla92b] dar una definición categórica del núcleo central de su trabajo. No obstante podemos considerar la siguiente definición informal, ofrecida por Mitchell [Mit97], como punto de

partida: *"Aprendizaje es la capacidad de adquirir o mejorar el conocimiento"*, o como afirma el autor en la misma obra, *"Aprender es mejorar con la experiencia"*.

Obviamente, se trata de una definición informal. Para hablar con rigor deberíamos definir previamente a qué nos referimos cuando hablamos de *conocimiento* y cómo lo *representamos*, como podemos establecer si un *cambio* en ese conocimiento representa una *mejora* y cómo podemos adquirir esa *experiencia.*

En sentido estricto, el almacenamiento en memoria de datos puede considerarse la forma más simple de aprendizaje. Si memorizamos el número de teléfono de Juan tardaremos menos tiempo en realizar la llamada que si tenemos que buscar el número cada vez. La acción *llamar a Juan* necesita menos tiempo, esto es, se ha producido una mejora, con la experiencia, puesto que busqué una vez su número y no necesito buscarlo más.

Pero, intuitivamente, podemos pensar que aprender es algo más que el mero almacenamiento de datos.

Dentro del ámbito del aprendizaje, independientemente de su soporte físico, se pueden aprender diversos aspectos del mundo en que vivimos. Podemos aprender a conducir un vehículo, podemos aprender alemán, aprender a resolver problemas, a tomar decisiones o podemos aprender a no tocar cables eléctricos sin protección. Pero también podemos aprender qué es un triángulo o una bicicleta. Todos estos tipos de aprendizaje nos sirven para desenvolvernos en la vida cotidiana. En esta memoria nos dedicaremos únicamente al último tipo de aprendizaje: el aprendizaje de conceptos.

El aprendizaje de conceptos necesita de la extracción de reglas generales a partir de la experiencia que nos permitan enfrentarnos con éxito, o al menos eso esperamos, a situaciones desconocidas. Son estas las dos características esenciales de un *sistema* de aprendizaje, ya sea con soporte natural o artificial:

- En primer lugar, ser capaz de reproducir con éxito situaciones ya vividas *(experiencia).* Cuando hablemos de aprendizaje de conceptos, haremos referencia a la *correcta clasificación de instancias observadas.* Para este punto basta el aprendizaje memorístico, que como apuntamos, estaría en la frontera de lo que consideramos aprendizaje. Un alumno puede aprender el resultado de todas las integrales de su libro y reproducirlas sin error si se le pregunta, pero difícilmente diremos que el alumno sabe calcular integrales.

- Es en la segunda parte, en la capacidad predictiva, donde radica el núcleo central del aprendizaje. No podemos hablar de aprendizaje propiamente di-

cho si el análisis de la experiencia no nos prepara para enfrentarnos a situaciones nuevas. Esta idea ha sido muy estudiada y en este capítulo veremos algunas aproximaciones históricas, pero en esencia, la idea central radica en la hipótesis de la *regularidad en la naturaleza* y en la suposición de que ante situaciones análogas obtendremos resultados parecidos. En la literatura, podemos encontrar varias formulaciones para esta idea, véase por ejemplo la *hipótesis del aprendizaje inductivo* en [Mit97]: *"Cualquier hipótesis que aproxime la función objetivo sobre un conjunto de entrenamiento suficientemente grande también aproximará la función objetivo sobre ejemplos no observados"*.

2.2 El aprendizaje de conceptos

El problema del aprendizaje de conceptos puede ser formulado, de manera informal, del siguiente modo: *Dada una teoría T y una entrada externa E, devolver una nueva teoría T^* que mejore T.* Obviamente, cómo sean esas teorías, la naturaleza de la entrada externa y qué entendemos por *mejorar* dependerá del tipo de aprendizaje considerado.

Stephen B. Klein, en [Kle97], comenta que hay dos teorías sobre el Aprendizaje de Conceptos:

- **Teoría asociativa:** Presentada por Clark Hull en 1920 [Hul20], considera el Aprendizaje de Conceptos como una forma de aprendizaje discriminativo (discriminamos entre instancias positivas y negativas del concepto). Según esta teoría, los conceptos tienen atributos relevantes e irrelevantes. En cada ensayo de un estudio de aprendizaje de conceptos el sujeto determina si el objeto o evento mostrado es característico del concepto. Un sujeto que responde correctamente es reforzado mediante retroalimentación (diciéndole que la respuesta es correcta). Como consecuencia del reforzamiento aumenta la fuerza de la respuesta ante los atributos característicos del concepto. Esta teoría fue defendida por la mayor parte de los psicólogos hasta finales de la década de los 50.

- **Teoría cognitiva:** Presentada por Bruner *et al.* [BGA56] en 1956. Para ellos, un concepto se aprende poniendo a prueba una hipótesis sobre la solución correcta. Si la hipótesis formulada en primer lugar es correcta, el

individuo ha aprendido el concepto. Sin embargo, si la hipótesis es incorrecta, se formulará otra hipótesis y se comprobará después. La comprobación de hipótesis continuará hasta que se descubre una solución correcta.

Tanto la teoría asociativa de Clark Hull como la cognitiva presentada por Bruner *et al.* nos conducen al problema del aprendizaje con método científico y nos aproximan a dos de las cuestiones básicas tanto en Aprendizaje de Conceptos como en Ciencias Experimentales:

- ¿Cómo generamos las hipótesis?
- ¿Cómo podemos saber que una hipótesis es *correcta*?

La primera pregunta sobre *cómo* generamos esas posibles explicaciones de los fenómenos observados nos lleva al problema de la *abducción*. La segunda pregunta, que nos cuestiona sobre los criterios para discriminar entre las distintas hipótesis y sobre el grado de *seguridad* con que podemos afirmar la certeza de la hipótesis elegida, nos lleva al problema de la *justificación*. Nos dedicaremos a ellos a continuación.

2.3 El problema de la abducción

La abducción es el proceso mediante el cual se generan explicaciones a partir de las observaciones. Es una forma de razonamiento no monótono, ya que las *explicaciones* que son consistentes con una determinada información, puede ser inconsistente con nuevas observaciones. Además, la existencia de *múltiples explicaciones* es una característica fundamental del razonamiento abductivo.

Veamos un ejemplo, tomado de [Pea87], sobre cómo actúa la abducción

Ejemplo 2.1 Consideremos la siguiente teoría T

$$\textit{el césped está mojado} \leftarrow \textit{llovió anoche}$$
$$\textit{el césped está mojado} \leftarrow \textit{ha caído rocío}$$
$$\textit{mis zapatos están mojados} \leftarrow \textit{el césped está mojado}$$

Si observamos que nuestros zapatos están mojados y queremos conocer la razón, *llovió anoche* es una posible explicación, esto es, una hipótesis que junto al conocimiento básico T implica las observaciones realizadas. Nótese que esta posible explicación no es única, también podemos suponer la hipótesis *ha caído rocío*.

Veamos cómo ha ido evolucionando el problema de la abducción a lo largo de la historia.

2.4 Un poco de Historia

Las principales fuentes para esta sección han sido [Mug92, AL97, Fla95] y [Pau93].

La historia del razonamiento inductivo está relacionada con el desarrollo de su contrapartida deductiva. En la antigua Grecia el razonamiento inductivo jugó un papel central en las discusiones dialécticas de Sócrates, como describe Platón en *Crito*. En estas discusiones los conceptos se definían y refinaban mediante ejemplos y contraejemplos procedentes de la vida cotidiana.

La inducción ya fue reconocida por Aristóteles como un modo particular de razonamiento en su *Analytica Posteriora*. Da una definición de lo que hoy conocemos como inducción completa *(epagoge)*, que es de hecho, una forma válida de razonamiento deductivo, por tanto no es relevante para nuestra idea de aprendizaje.

No obstante, podemos relacionar la abducción, como razonamiento desde la evidencia a la explicación, con la *apagoge* de Aristóteles (ver [AL97]), el cual es un tipo de razonamiento no deductivo en sentido estricto cuyas conclusiones no son *necesarias* sino *posibles*.

Ya en el siglo XVII, Francis Bacon fue uno de los primeros filósofos en destacar la importancia de la inducción como método en la formación de teorías científicas en su *Novum Organum* en 1620.

En 1739, el filósofo escocés David Hume, en su *Treatise on Human Nature*, concluyó que la idea de causalidad es algo que existe únicamente en la mente, una imagen mental provocada por la observación en un cierto número de ocasiones que situaciones de un determinado tipo vienen seguidas de situaciones de un segundo tipo. En otras palabras, no hay necesidad lógica inherente en la causalidad y cualquier intento de formular una base lógica para la inducción estará abocada al fracaso. En lugar de necesidad, Hume fue el primero en formular las conclusiones inductivas en términos de *probabilidad*, pero para él esta probabilidad no se entiende en sentido matemático, sino como el grado de deseo o intención que tenemos de aceptar las hipótesis a partir de las observaciones realizadas, o dicho de otro modo, en un sentido subjetivo.

En 1843, John Stuart Mill desarrolló una metodología científica inductiva en su *System of Logic*. Mill identifica cuatro *"Methods of Experimental Enquiry"* y

formula cinco *Canons* o principios que los soportan y justifican.

El filósofo americano Charles Sander Peirce [Pei32, Pei33] advirtió que ver la inducción como el proceso de asignar probabilidades presupone que la hipótesis inductiva ya está dada y no responde a la pregunta del origen de esas hipótesis. Peirce introdujo el término *abducción* para denotar el proceso de formación de hipótesis explicativas y defendió tal proceso como un proceso de inferencia lógica.

Para Peirce, *la abducción es el proceso de formar una hipótesis explicativa.* Si tenemos la definición de un concepto y descubrimos una nueva instancia en nuestro universo *(ejemplo)* que no es consistente con nuestra definición, debemos cambiar la definición que teníamos en ese momento y proponer una nueva definición *(hipótesis)* para nuestro concepto que cubra ese nuevo ejemplo. Ese proceso es el proceso de abducción. Más precisamente, Peirce definió la abducción de la siguiente manera[1]:

- *The surprising fact C is observed.*
- *But if A were true, C would be a matter of course.*
- *Hence, there is reason to suspect that A is true.*

En la misma línea, podemos encontrar trabajos de Polya [Pol45] donde hablaba de un nuevo tipo de silogismo, llamado *silogismo heurístico*, en contraposición del silogismo *demostrativo*:

- En el *silogismo demostrativo*, de $A \to B$ y B falso, concluimos $\neg A$.
- En un *silogismo heurístico*, de $A \to B$ y B cierto, concluimos que A es *más creíble.*

En [Pei32] Peirce distinguió entre tres tipos de reglas de inferencia, aunque en la literatura relativa a la PLI, las fronteras entre abducción e inducción no son tan precisas como Peirce las definió. Son las siguientes:

Deducción. Es el proceso analítico de aplicar reglas de inferencia a las premisas, con la conclusión como resultado.

Inducción. Es el razonamiento sintético en el que obtenemos la regla de inferencia a partir de las premisas y la conclusión

[1]Respetamos el lenguaje en el que Peirce lo definió.

Abducción. Es otro tipo de razonamiento sintético, en el que obtenemos las premisas a partir de las reglas de inferencia y la conclusión.

Podemos ilustrar esta clasificación con el siguiente ejemplo, original de Peirce:

- DEDUCCION
  - Regla: Todas las judías de esta bolsa son blancas
  - Premisa: Estas judías proceden de esta bolsa
  - Resultado: Estas judías son blancas
- INDUCCION
  - Premisa: Estas judías proceden de esta bolsa
  - Resultado: Estas judías son blancas
  - Regla: Todas las judías de esta bolsa son blancas
- ABDUCCION
  - Regla: Todas las judías de esta bolsa son blancas
  - Resultado: Estas judías son blancas
  - Premisa: Estas judías proceden de esta bolsa

En los primeros trabajos [Pei32], Peirce ve la abducción como el proceso de extraer el efecto de la causa, considerado éste como *explicación*, en contraste con la inducción que conjetura leyes generales a partir de casos particulares. De esta forma, en lugar de sintetizar explicaciones, la inducción *clasifica* pero no añade nuevos conocimientos a la teoría. En trabajos posteriores, Peirce enfatiza la conexión existente entre ambos tipos de razonamiento de la siguiente manera[2]: *Induction is an argument which sets out from a hypothesis, resulting from Abduction ....*

La idea de tomar la lógica como la base del razonamiento científico tuvo su cumbre en los trabajos del Wiener Kreis. El *Wiener Kreis* –Círculo de Viena– que incluía Rudolf Carnap [Car52] y Carl G. Hempel [Hem43] defendió la doctrina del positivismo lógico. Influidos por Wittgenstein y su *Tractatus Logico-Philosophicus* defendieron que muchos de los problemas que trataba la filosofía no eran tales, sino que procedían de un uso impreciso o incorrecto del lenguaje.

[2]Ver p. 198 en [Gou50].

Según esta corriente, el auténtico conocimiento debe ser *verificable* y el método científico es el único que permite alcanzarlo.

El concepto de abducción no fue hasta 1973 cuando aparece en trabajos de inteligencia artificial. El primer campo en el que se utilizó fue en la diagnosis [Pop73], pero se ha extendido a campos tan diversos como la interpretación de textos [Sti90], la actualización de bases de datos [KM90] o visión artificial [CM85]. En 1985 volvió a retomarse con fuerza el estudio de la abducción gracias a trabajos como los de Charniak y McDermott [CM85].

2.5 La abducción en lógica

En una primera aproximación, el objetivo de la abducción puede caracterizarse como el problema siguiente: Dado un conjunto de sentencias T que llamaremos *conocimiento base*, y una sentencia G, una *observación*, encontrar un conjunto de sentencias H (una *explicación abductiva*) tal que

1. $T \cup H \models G$

2. $T \cup H$ sea consistente.

Esta caracterización de la abducción es independiente del lenguaje en el que T, G y H sean formulados. El signo de implicación lógica $\models$ en (1) puede ser reemplazado por el operador de deducción $\vdash$. La condición de consistencia expresada en (2) no es explícita en la caracterización de Peirce, más informal, pero es una condición que surge de forma natural.

De hecho esas dos condiciones son todavía demasiado débiles para atrapar la idea de Peirce. Además, parece natural que exijamos que H sea en realidad la *causa* (o una de las posibles causas) que expliquen nuestras observaciones, es decir, no queremos explicar efectos en virtud de otros *efectos*, sino de las causas primeras. Estos criterios nos van a determinar una clase especial de sentencias de entre las cuales vamos a tomar H y que la literatura en la materia llama *abducibles*.

Los criterios antes mencionados parecen ser universalmente aceptados a la hora de considerar el conjunto de sentencias abducibles, pero además podemos considerar otros.

Poole demostró en [Poo88] que podemos restringir nuestro conjunto de sentencias abducibles sin perder generalidad y permitir que sólo los átomos puedan ser considerados como abducibles.

Existen otros muchos criterios encaminados a restringir el conjunto de sentencias abducibles, entre los que podemos citar el de Sattar y Goebel [SG89] en el que se utilizan *literales cruciales* para decidir entre dos explicaciones incompatibles, o la técnica de *corroboración* de Evans y Kakas [EK92] para seleccionar hipótesis, en la que una hipótesis falla en el test de corroboración si alguna de sus consecuencias lógicas no es observada. En general, los criterios para limitar el conjunto de abducibles son de lo más variado.

En Cox y Pietrzykowski [CP86], entre otros, encontramos otras propiedades deseables de las explicaciones abducibles:

- Una explicación debería ser **básica**, esto es, no debería ser explicable en términos de otras explicaciones. Así en el ejemplo 2.1 la explicación

  el césped está mojado

  para la observación

  mis zapatos están mojados

  no es básica, mientras que las explicaciones

  llovió anoche
  ha caído rocío

  sí lo son.

- Una explicación debería también ser **minimal**, por ejemplo, si tenemos el programa

  $$p \leftarrow q$$
  $$p \leftarrow q, r$$

  y realizamos la observación $\{p\}$, la explicación $\{q\}$ es minimal, mientras que $\{q, r\}$ no lo es.

- Las explicaciones deberían ser no triviales. Una explicación H de una observación O es **no trivial** si $\not\models H \rightarrow O$, o lo que es lo mismo, se dice **trivial** si todo modelo de H lo es de O. De esta forma eliminamos las posibles explicaciones que no están relacionadas con el conocimiento base, sino que por sí mismas implican la observación. En nuestro ejemplo, la explicación

Todo está mojado

para la observación

mis zapatos están mojados

es trivial, pero *llovió anoche* no lo es.

- Por último, una explicación H debe ser *consistente* con el conocimiento base K, esto es, $K \wedge H$ debe tener algún modelo.

2.6 Justificación

En el problema de la justificación abordamos el segundo problema crucial en la adquisición de nuevo conocimiento: ¿Podemos estar seguros de que elegimos la hipótesis correcta? Al igual que el problema de la abducción, la justificación ha sido muy estudiada en Filosofía y podemos encontrar respuestas muy dispares.

Una de las respuestas más radicales la encontramos en los filósofos empiristas, con David Hume en su expresión más radical. Hume, en su *Treatise on Human Nature* afirmó que la idea de una *ley causal* es algo que existe únicamente en la mente y no hay una *conexión lógica* inherente a las leyes causales. La *creencia* de que el agua de la vasija se calentará al arrimarla al fuego proviene del *hábito* o *costumbre* de haber observado en el pasado que siempre que sucedió lo primero, sucedió también lo segundo.

El punto de vista de Hume es extremadamente radical. Karl R. Popper [Pop58, Pop63] dirigió su atención, como Hume había hecho antes, al hecho de que las teorías científicas no pueden ser *verificadas* de forma conclusiva. Pero sí pueden ser *falsificadas* de manera conclusiva. Si realizamos un nuevo experimento que *corrobora* nuestra teoría, no podemos afirmar que no podemos encontrar en el futuro un ejemplo que la *falsifique*, esto es, mediante el estudio de una cantidad, siempre finita, de ejemplos, no podemos *verificar* nuestra teoría de manera conclusiva. Ahora bien, si encontramos un ejemplo que contradiga nuestra actual teoría, que la *falsifique*, sí podemos tener seguridad de que nuestra teoría es errónea y debe ser modificada.

En el siglo pasado podemos encontrar puntos de vista como los de Carnap y Hempel[3].

[3] Cf. [Fla95].

Hempel [Hem43, Hem45] desarrolló su teoría desde un punto de vista *cualitativo* mediante la definición de una relación binaria entre dos fórmulas lógicas E y H que formalizara que la proposición E *confirma* H.

R. Carnap [Car50, Car52] desarrolló su teoría desde un punto de vista *cuantitativo* definiendo una función que asociara a la fórmula H un número entre 0 y 1 una vez considerada la experiencia E. Este punto de vista *cuantitativo* y matematizable es seguido por muchos investigadores en PLI para establecer criterios de aceptación de las hipótesis obtenidas.

Para finalizar, cabe destacar que los términos abducción, inducción y justificación podemos encontrarlos en la literatura relativa a PLI con significados muy distintos a los aquí expuestos. Por ejemplo, la interpretación que dan De Raedt y Muggleton al término *inducción* es muy diferente a la que presentó Peirce. Para ellos, *inducción* es el proceso contrario a la *deducción* y justifican esta terminología porque según ellos, les permite dar un tratamiento más operativo a la *inducción*.

De Raedt y Muggleton comentan este doble problema en [MDR94] y lo resumen con la siguiente ecuación:

$$\boxed{\text{Inducción} = \text{Abducción} + \text{Justificación}}$$

donde para ellos, la abducción es el proceso de formación de las hipótesis y la justificación es el grado de credibilidad asociado a una hipótesis a partir de una evidencia.

2.7 Aprendizaje automático

El Aprendizaje Automático estudia el aprendizaje desde una perspectiva computacional, pero dentro de este marco los investigadores se aproximan a él por diferentes caminos. Como bien apunta Pat Langley, [Lan96] los investigadores rara vez hacen explícitos sus objetivos, pero podemos agrupar estos en cuatro grandes líneas:

- El primer gran objetivo está relacionado con el modelado computacional de los mecanismos que subyacen en el aprendizaje humano. En este marco *psicológico*, los investigadores desarrollan algoritmos de aprendizaje consistentes generalmente con la arquitectura cognitiva humana, diseñados para explicar comportamientos específicos de aprendizaje observados en seres vivos.

- Un segundo grupo de investigadores sigue una aproximación *empírica* en el estudio del Aprendizaje Automático. Su objetivo es descubrir principios generales que relacionen de un lado las características de los algoritmos de aprendizaje y las de los dominios en los que operan, para aprender su comportamiento. La aproximación estándar consiste en realizar experimentos que varíen o bien en el algoritmo o bien en el dominio de aplicación y observar el impacto que tienen estas variaciones en el aprendizaje.

- Un tercer grupo, también interesado en principios generales, estudia el aprendizaje automático desde un punto de vista *matemático*. Este punto de vista teórico trata de formular y demostrar resultados generales sobre las posibilidades y limitaciones de las técnicas y algoritmos usados en Aprendizaje Automático. En esta línea se incluye esta memoria.

- Podemos destacar un cuarto y último grupo de investigadores en Aprendizaje Automático. Aquellos que siguen una línea de estudio guiada por las *aplicaciones* del aprendizaje automático al mundo real. Hoy día, las aplicaciones a dominios del mundo real son uno de los motores de esta disciplina, que se encuentra en el núcleo de otras tales como el Descubrimiento de Conocimiento en Bases de Datos *(KDD)* o la Minería de Datos.

Con una visión unitaria, podemos decir que todas estas aproximaciones, por diferentes que sean, estudian el desarrollo y evaluación de *algoritmos* de aprendizaje.

2.8 Aprendizaje de conceptos en máquinas

Los dos principales problemas a los que se enfrenta el Aprendizaje de Conceptos son los mismos que el Aprendizaje sobre sustrato biológico (sección 2.1):

- La generación de posibles hipótesis que expliquen las observaciones realizadas y

- La decisión de tomar una hipótesis determinada.

En el *aprendizaje automático de conceptos* la entrada son ejemplos positivos y negativos del concepto que se quiere aprender, junto con información adicional *(conocimiento básico)*, que según qué sistema, podrá ser utilizada en el proceso de aprendizaje. Si partimos de la idea de que un concepto puede ser definido como el conjunto de todas sus instancias, podemos formalizar la idea de conjunto

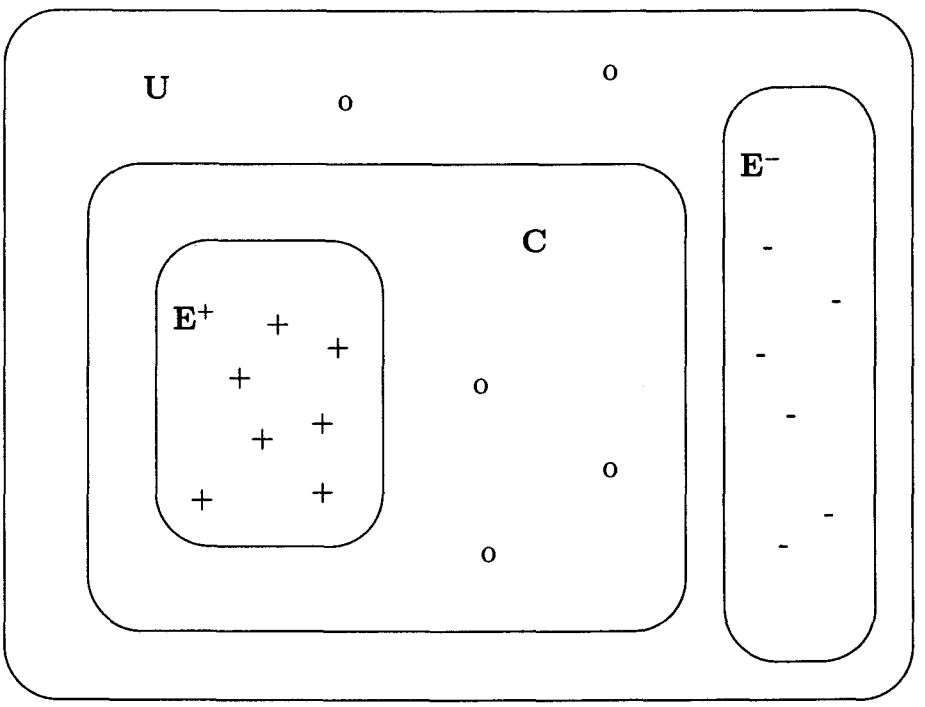


Figura 2.1: Conceptos como conjuntos

como una función booleana sobre el universo de instancias, donde asignamos 1 a las instancias que pertenecen al concepto y 0 a las que no.

Supongamos que tenemos un conjunto de instancias para las que conocemos su clasificación: Los ejemplos positivos E^+ y los ejemplos negativos E^- (ver figura 2.1). Salvo casos muy excepcionales, existirán elementos en el universo U que no pertenezcan a ninguno de estos dos conjuntos (i.e., ejemplos de clasificación desconocida). Nuestro objetivo es extender la función de clasificación de los ejemplos conocidos a todo el universo, esto es, debemos *establecer una hipótesis* h sobre la clasificación de *todas* las instancias del universo. De manera más formal, consideramos un concepto c como una función booleana sobre el universo U de todas las posibles instancias $c : U \rightarrow \{0, 1\}$ y un conjunto de ejemplos D, donde los elementos de D son pares $\langle x, c(x)\rangle$, donde x es un elemento de U y $c(x)$ es la clasificación (positiva o negativa) de la instancia. Consideramos también un conjunto H de hipótesis, formado por funciones $h : U \rightarrow \{0, 1\}$. El objetivo es, por una parte, encontrar una función $h \in H$ tal que $h(x) = c(x)$ para toda instancia $x \in U$.

En una primera aproximación debemos exigir como requisito básico que la hipótesis h clasifique correctamente las instancias de clasificación conocida. Como

apuntamos en la sección 2.1, el interés del proceso de aprendizaje radica en la *capacidad predictiva* del sistema, esto es, que la hipótesis h que conjeturamos clasifique correctamente las instancias que no pertenecen al conjunto de ejemplos. A la hora de decidir qué conjunto tomamos como hipótesis h tenemos, como es natural, varias posibilidades. El menor conjunto que podemos tomar es E^+ y el mayor conjunto el complementario de E^-, $\overline{E^-}$, pero en general podemos tomar como posible definición cualquier conjunto D tal que $E^- \subseteq D \subseteq \overline{E^-}$. De este modo aseguramos la clasificación correcta de los ejemplo conocidos.

A partir de esta formalización, el objetivo del Aprendizaje Automático de Conceptos se centra en la selección de una de las hipótesis posibles en función de las elecciones realizadas respecto la representación y el método de búsqueda en el espacio de hipótesis.

Supongamos que queremos aprender un concepto de forma incremental, esto es, tenemos una hipótesis h_0 que *explica* todas las observaciones (positivas y negativas) realizadas hasta el momento, y encontramos un nuevo ejemplo. Se tiene una de las siguientes situaciones:

1. El ejemplo es positivo y pertenece a nuestro concepto.

2. El ejemplo es positivo y *no* pertenece a nuestro concepto.

3. El ejemplo es negativo y pertenece a nuestro concepto.

4. El ejemplo es negativo y *no* pertenece a nuestro concepto.

En los casos 1 y 4 nuestra definición *explica* o *cubre* ese nuevo ejemplo y no tenemos que cambiar nada. En el caso 2 tenemos que extender nuestra definición hasta cubrir esa nueva observación. Este proceso es el que conocemos como *generalización*. El proceso dual a la generalización es la *especialización* y tiene lugar cuando nos enfrentamos a un caso del tipo 3. En esta situación tenemos que hacer más pequeña nuestra definición hasta dejar fuera la nueva observación.

Podemos obtener una visión de la generalización y especialización considerando el *espacio de conceptos*, esto es, el conjunto de todos los conceptos, parcialmente ordenados por la relación *ser subconjunto de*. Podemos generalizar un concepto subiendo por este orden parcial y especializarlo bajando por él.

Un avance en los sistemas de representación de conceptos lo representó el teorema de Mitchell sobre espacios de versiones [Mit82]. Se define el *espacio de versiones* como el conjunto de todos los conjuntos (conceptos) que contienen a

los ejemplos positivos y a ningún ejemplo negativo (ver [Mit82]), esto es, una hipótesis h es consistente con un conjunto de ejemplos D si y sólo si $h(x) = c(x)$ para todo $\langle x, c(x)\rangle \in D$. Si h es consistente con D lo representamos

$$Consistente(h, D)$$

y el *espacio de versiones* respecto al espacio de hipótesis H y al conjunto de ejemplos D, denotado por $VS_{H,D}$ es el conjunto de hipótesis de H consistentes con el conjunto de ejemplos D.

$$VS_{H,D} = \{h \in H \mid Consistente(h, D)\}$$

Es fácil ver que este conjunto es *convexo* para la relación de contención entre conjuntos, en el sentido de que dados dos elementos cualesquiera todo elemento entre ellos pertenece al espacio de versiones. Para enunciar el teorema de Mitchell necesitamos algunas definiciones previas

Definición 2.2 *Sean h_k y h_j dos funciones booleanas definidas sobre un conjunto X. Entonces h_j es* más general que o igual que *h_k, denotado por $h_j \geq_g h_k$ si y sólo si*

$$(\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

Estamos ahora en condiciones de definir los conjuntos $\mathcal{S}$ y $\mathcal{G}$ presentados por Mitchell en [Mit82].

Definición 2.3 *La* cota general *$\mathcal{G}$ respecto al espacio de hipótesis H y el conjunto de entrenamiento (ejemplos) D es el conjunto*

$$\mathcal{G} = \{g \in H \mid Consistente(g, D) \wedge (\neg\exists g' \in H)[(g' \geq_g g) \wedge Consistente(g', D)]\}$$

La cota específica *$\mathcal{S}$ respecto al espacio de hipótesis H y el conjunto de entrenamiento D es el conjunto*

$$\mathcal{S} = \{s \in H \mid Consistente(s, D) \wedge (\neg\exists s' \in H)[(s \geq_g s') \wedge Consistente(s', D)]\}$$

$\mathcal{S}$ determina una cota inferior del conjunto de todos los conceptos consistentes con D. Análogamente $\mathcal{G}$ determina una cota superior. Mitchell demostró que ambos conjuntos $\mathcal{S}$ y $\mathcal{G}$ determinan el *espacio de versiones* $\mathcal{VS}_{H,D}$.

Teorema 2.4 ([Mit82]) *Sea U un conjunto de instancias y sea H un conjunto de funciones booleanas definidas sobre U. Sea $c : U \leftarrow \{0,1\}$ un concepto definido sobre U y sea D un conjunto de pares $\langle x, c(x)\rangle$, donde x es una instancia y $c(x)$ es su imagen por la función c. Entonces*

$$\mathcal{VS}_{H,D} = \{h \in H \mid (\exists s \in \mathcal{S})(\exists g \in \mathcal{G})(s \subseteq h \subseteq g\}$$

El objetivo, como apuntamos antes, es la búsqueda del *mejor* elemento del espacio de versiones. La definición de *mejor* dependerá del dominio de aplicación, del sistema de búsqueda en el espacio y del sistema de representación que elijamos. Como veremos más adelante, la definición extensiva de concepto que hemos dado no es operativa y necesitamos encontrar lenguajes suficientemente expresivos para captar las diferencias entre los elementos del espacio de versiones mediante definiciones intensivas.

2.9 El lenguaje de valores de atributos

Presentamos aquí uno de los lenguajes más simples que podemos utilizar en los sistemas de aprendizaje: El lenguaje de los valores de atributos. Más adelante veremos que la búsqueda de sistemas de representación más expresivos es una de las motivaciones del nacimiento de la PLI. Este lenguaje describe los ejemplos y los conceptos en términos de valores de una serie de atributos tales como el color, el tamaño o la forma. Por ejemplo la expresión

$$\begin{array}{lcll} \textit{tamaño} & = & \textit{grande} & y \\ \textit{color} & = & \textit{rojo} & y \\ \textit{forma} & = & \textit{cuadrado} & \end{array}$$

denota el concepto de cuadrados grandes rojos.

La misma expresión se usa para describir un elemento de ese concepto, es decir, un cuadrado grande rojo. Así, el lenguaje de valores de atributos no distingue entre ejemplos y conceptos. Tiene la misma potencia expresiva que la lógica proposicional. Este hecho de usar la misma representación para ejemplos y conceptos se conoce con el nombre de *truco de la representación única (single representation trick)* [DLCD82]. Como resultado de este truco, comprobar si un ejemplo pertenece a un conjunto es lo mismo que comprobar si un concepto está contenido en otro.

La complejidad del aprendizaje depende fuertemente de la complejidad del lenguaje. Así si sólo permitimos la conjunción en expresiones de valores de atributos, la generalidad entre conceptos se implementa fácilmente comprobando que cada par atributo–valor que ocurra en una expresión ocurre también en la otra. Generalizar entonces es subir en el orden parcial y dicha generalización se realiza eliminando pares atributo–valor en la conjunción. Así, el concepto C_1

$$\begin{array}{lcll} \textit{tamaño} & = & \textit{grande} & y \\ \textit{color} & = & \textit{rojo} & y \\ \textit{forma} & = & \textit{circulo} & \end{array}$$

contiene al concepto C_0

$$\begin{array}{lcll} \textit{tamaño} & = & \textit{grande} & y \\ \textit{color} & = & \textit{rojo} & y \\ \textit{forma} & = & \textit{circulo} & y \\ \textit{peso} & = & \textit{ligero} & \end{array}$$

y puede ser generalizado al concepto C_2

$$\begin{array}{lcll} \textit{color} & = & \textit{rojo} & y \\ \textit{forma} & = & \textit{circulo} & \end{array}$$

Análogamente la especialización se lleva a cabo añadiendo a la conjunción un nuevo par atributo–valor. Así, si tuviéramos en un momento determinado el siguiente concepto

$$\begin{array}{lcll} \textit{color} & = & \textit{rojo} & y \\ \textit{forma} & = & \textit{circulo} & \end{array}$$

y consideramos el siguiente ejemplo negativo (esto es, un ejemplo que no debe pertenecer a nuestro concepto)

$$\begin{array}{lcll} \textit{tamaño} & = & \textit{grande} & y \\ \textit{color} & = & \textit{rojo} & y \\ \textit{forma} & = & \textit{circulo} & \end{array}$$

basta añadir a nuestro concepto un nuevo par atributo–valor que no ocurra en el ejemplo para que éste no pertenezca al nuevo concepto.

Evidentemente la expresividad de este lenguaje es muy limitada. Dos posibles direcciones para esa generalización serían por una parte permitir la combinación

de conjunciones y disyunciones es una misma fórmula y por otra, la inclusión de la negación dentro de las fórmulas.

Obviamente una mayor expresividad del lenguaje lleva asociada una mayor complejidad del proceso de aprendizaje. Los sistemas de aprendizaje buscan el equilibrio entre ambos extremos. Necesitamos un sistema de representación que nos permita una expresividad aceptable, pero que a su vez permita desarrollar algoritmos con costes computacionales reducidos.

2.10 Hacia un sistema de representación clausal

La búsqueda de sistemas formales de representación más expresivos que el lenguaje de atributos de valores nos lleva a la lógica clausal. Este estudio del aprendizaje automático mediante el uso del lenguaje y las técnicas de la Programación Lógica nos conducirá a la Programación Lógica Inductiva.

Los conceptos descritos por conjunciones de pares atributo–valor pueden escribirse como cláusulas del siguiente modo

$$\begin{aligned} concepto1(X) \leftarrow\ & rojo(X), \\ & cuadrado(X) \end{aligned}$$

donde el predicado *concepto1* representa el concepto que debe ser aprendido. Nótese que los atributos no aparecen en la cláusula, y que por tanto hemos perdido información del tipo *rojo y verde son valores del mismo atributo, y por tanto, mutuamente excluyentes.* Si necesitamos esa información debemos darla de forma explícita en el *conocimiento base T.*

Pueden existir varias cláusulas en la definición de un mismo concepto, por ejemplo

$$\begin{aligned} concepto2(X) \leftarrow\ & rojo(X), \\ & cuadrado(X). \\ concepto2(X) \leftarrow\ & verde(X), \\ & circulo(X). \end{aligned}$$

que corresponde al término del lenguaje de valores de atributos

$$\begin{array}{llll} (color & = & rojo & y \\ forma & = & cuadrado) & o \\ (color & = & verde & y \\ forma & = & circulo) & \end{array}$$

podemos considerar un conjunto de ejemplos representados como cláusulas unitarias positivas sin variables.

$$grande(grande\_rojo\_cuadrado)$$
$$rojo(grande\_rojo\_cuadrado)$$
$$cuadrado(grande\_rojo\_cuadrado)$$

grande_rojo_cuadrado es una instancia de *concepto1*, puesto que, como veremos

$$concepto1(grande\_rojo\_cuadrado)$$

puede ser probado. Notaremos como *Desc(objeto)*, a un conjunto de cláusulas *(programa)* cada una de las cuales está compuesto de un único literal.

$$(\bigcup_{i=1}^{i=n}\{p_i(objeto)\}) \cup (\bigcup_{j=1}^{j=m}\{\neg q_j(objeto)\})$$

donde *objeto* es una constante de nuestro lenguaje, los $p_i(objeto)$ son todos los ejemplos positivos en los que ocurre *objeto* y los $q_j(objeto)$ son los ejemplos negativos en los que ocurre.

En general, dado un conocimiento base T y la descripción de un objeto *Desc(objeto)*, diremos que un concepto

$$concepto(X) \leftarrow condiciones(X)$$

clasifica un *objeto* de forma *positiva* si

$$\left\{ \begin{array}{ll} T & \wedge \\ Desc(objeto) & \wedge \\ concepto(X) \leftarrow condiciones(X) & \end{array} \right\} \vdash concepto(objeto)$$

En caso de que la demostración falle, consideramos que la clasificación es *negativa* (Negación como regla de fallo finito). Evidentemente, nuestro deseo es encontrar una hipótesis *(una definición del concepto)* tal que clasifique correctamente todos los ejemplos. Partimos de una hipótesis original. Si esta hipótesis no clasificara correctamente los ejemplos, pasaríamos a especializarla o a generalizarla.

2.11 Programación Lógica Inductiva

La Programación Lógica Inductiva (PLI) es el área del Aprendizaje de Conceptos en la que tanto las técnicas como la representación del conocimiento se basan en Programación Lógica.

La PLI puede ser definida de manera informal como la intersección entre la Programación Lógica y el Aprendizaje Automático (S. Muggleton [Mug91]). Del aprendizaje automático la PLI toma su objetivo, esto es, el estudio de técnicas que permitan formular hipótesis a partir de observaciones (ejemplos) y sintetizar nuevos conocimientos a partir de la experiencia. De la Programación Lógica la PLI toma su representación formal, su orientación semántica y sus técnicas. Con el uso de las técnicas de la lógica computacional la PLI intenta superar los dos principales obstáculos que hasta ahora ha tenido el aprendizaje automático:

1. El uso de una representación formal limitada, esencialmente la lógica proposicional

2. Las dificultades del uso del conocimiento base en los procesos de aprendizaje

En la literatura podemos encontrar muchas definiciones de PLI. Como cualquier otra disciplina incipiente es difícil de definir y cada autor pone el acento en una faceta, así Bergadano y Gunetti [BG95] afirman que la PLI *estudia el desarrollo de programas lógicos con ayuda de ejemplos.* Siguiendo a Flach [Fla92a], podemos definir la Programación Lógica Inductiva como el proceso de ampliar una teoría parcial T con una hipótesis inductiva H tal que la teoría ampliada *explique* un conjunto de ejemplos E.

Nada Lavrac y Luc De Raedt [LR95] afirman que la PLI puede definirse como el área de investigación en la intersección del Aprendizaje Automático y la Lógica Computacional cuyo principal objetivo es el desarrollo de teorías y algoritmos prácticos para el aprendizaje inductivo de programas lógicos.

Al igual que ocurre con otras ramas del Aprendizaje de Conceptos, estos sistemas buscan una definición (expresada, en este caso, como programa lógico) que permita clasificar las observaciones realizadas, definición que se verá reforzada si futuras observaciones son consistentes con dicha clasificación. Si por el contrario las nuevas observaciones no coinciden con la clasificación esperada, nuestra definición debe ser modificada.

El sistema no sólo debe clasificar correctamente instancias ya observadas, sino que debe ser capaz de extraer las propiedades que caracterizan una determinada

clasificación y usar ese criterio para clasificar instancias no observadas. Como hemos visto (Sección 2.1), es precisamente esta capacidad predictiva el principal objetivo de un sistema de aprendizaje.

A los métodos que parten de una explicación muy pobre (puede ser vacía) que debemos ir generalizando los llamamos métodos *ascendentes*. Esta primera explicación (o hipótesis) suele corresponder con el conjunto vacío. Si identificamos el concepto que queremos definir con el conjunto de sus instancias, esta primera hipótesis clasifica negativamente todos los elementos del universo. Si conocemos algún ejemplo *positivo* debemos *generalizar* la hipótesis, para que clasifique positivamente este ejemplo, esto es, debemos hacer la hipótesis más general.

A los métodos en los que partimos de una explicación muy general, que iremos haciendo más específica con el uso de los ejemplos, los llamamos *descendentes*. Es el caso dual del anterior. Podemos empezar por una hipótesis que *cubra* todas las instancias del universo. Si conocemos un ejemplo *negativo*, nuestra hipótesis debe hacerse *más específica*, para que no cubra el ejemplo negativo.

Antes de presentar formalmente el problema general inductivo que intenta resolver la PLI tenemos que considerar los siguientes tres lenguajes:

- $\mathcal{L}_O$: El lenguaje de las observaciones (ejemplos)
- $\mathcal{L}_B$: El lenguaje del conocimiento base
- $\mathcal{L}_H$: El lenguaje de las hipótesis

En una primera aproximación, el problema que intenta resolver la PLI es el siguiente: Dado un conjunto consistente de ejemplos u observaciones O expresado en el lenguaje $\mathcal{L}_O$ y un conocimiento básico B consistente expresado en $\mathcal{L}_B$, encontrar una hipótesis H expresada en $\mathcal{L}_H$ tal que

$$B \wedge H \vdash O$$

Veamos un ejemplo tomado de [LD94].

Ejemplo 2.5 Supongamos que queremos encontrar una definición para la relación $hija(X, Y)$ que se verifica si la persona X es hija de la persona Y. Para ello contamos con ejemplos positivos y negativos del predicado $hija/2$ así como definiciones correspondientes a las relaciones $mujer/1$ y $progenitor/2$ que actúan como conocimiento base (Fig. 2.2) . Supongamos que el lenguaje para las

Conjunto de entrenamiento		Conocimiento base	
hija(maria,ana)	$\oplus$	progenitor(ana,maria)	mujer(ana)
hija(eva,tomas)	$\oplus$	progenitor(tomas,eva)	mujer(maria)
hija(tomas,ana)	$\ominus$	progenitor(ana,tomas)	mujer(eva)
hija(eva,ana)	$\ominus$		

Figura 2.2: Ejemplo

hipótesis nos permite expresar la siguiente cláusula

$$hija(X,Y) \longleftarrow mujer(X), progenitor(Y,X)$$

Esta hipótesis clasifica correctamente los ejemplos con ayuda del conocimiento base. Dependiendo del conocimiento base y de las restricciones sobre el lenguaje de las hipótesis, también podremos encontrar hipótesis con más de una cláusula

$$hija(X,Y) \longleftarrow mujer(X), padre(Y,X)$$
$$hija(X,Y) \longleftarrow mujer(X), madre(Y,X)$$

En la literatura podemos encontrar distintas clasificaciones de los sistemas PLI basándonos en los criterios más variados, como la separación en métodos ascendentes o descendentes que antes comentamos, pero en general, estas clasificaciones suelen tener un sentido pedagógico y la tendencia es a desarrollar sistemas mixtos en busca de la eficiencia. A modo de ejemplo, De Raedt y Bruynooghe [DB92] clasifican los sistemas de la siguiente manera:

Progamación Lógica Inductiva Empírica dentro de la cual entrarían sistemas como FOIL [Qui90], LINUS [LDG91] o GOLEM [MF90] las cuales *aprenden* la definición de una única relación a partir de una gran cantidad de ejemplos de forma muy eficiente mediante cálculos heurísticos.

Programación Lógica Inductiva Interactiva entre los que podemos considerar los sistemas MIS [Sha83], MARVIN [SB86] y CLINT [DRB89, DR91] que *aprenden* definiciones de múltiples relaciones a partir de pocos ejemplos y consultas al oráculo (el usuario).

Antes de pasar a ver *cómo* se resuelve el problema veamos *qué* problema queremos resolver. Tenemos que *interpretar* el problema para fijar, de forma declarativa, unos objetivos. Existen dos interpretaciones fundamentales denominadas

semántica normal y *semántica no monótona.* La primera es la que siguen la inmensa mayoría de los sistemas de PLI, sobre todo en su versión para programas definidos, la *semántica definida.* La segunda, la semántica *no monótona*, podemos encontrarla en Helft [Hel89], De Raedt y Džeroski [RD94], la tesis de Džeroski [Dž95] o en Flach [Fla92a, Fla94, Fla95].

Para la presentación de la *semántica normal* seguimos [MDR94].

Definición 2.6 (Semántica normal) *En esta formalización se permite que los ejemplos, hipótesis y conocimiento básico estén compuestos por fórmulas de primer orden. Dados un conocimiento básico B, dos conjuntos de ejemplos E^+ y E^-, el objetivo es encontrar una hipótesis H tal que se verifiquen las siguientes condiciones:*

- **Consistencia a priori** $B \wedge E^- \not\models \square$
- **Consistencia a posteriori** $B \wedge H \wedge E^- \not\models \square$
- **Necesidad a priori** $B \not\models E^+$
- **Suficiencia a posteriori** $B \wedge H \models E^+$

Al criterio de *suficiencia a posteriori* se le llama a veces *completitud con los ejemplos positivos* y al de *consistencia a posteriori* se le llama *consistencia con los ejemplos negativos.*

En muchos sistemas de PLI sólo se permite que el conocimiento base y las hipótesis sean cláusulas definidas, así como que los ejemplos sean átomos cerrados. En este caso el problema de la semántica es mucho más sencillo, puesto que entonces la teoría clausal T tiene un menor modelo de Herbrand $M^+(T)$.

Definición 2.7 (Semántica definida) *Aquí los criterios que deben verificarse son los siguientes:*

- **Consistencia a priori** *Todo $e \in E^-$ es falso en $M^+(B)$, i.e.,*

$$E^- \cap M^+(B) = \emptyset$$

- **Consistencia a posteriori** *Todo $e \in E^-$ es falso en $M^+(B \wedge H)$, i.e.,*

$$E^- \cap M^+(B \wedge H) = \emptyset$$

- **Necesidad a priori** *Algún* $e \in E^+$ *es falso en* $M^+(B)$, *i.e.,*

$$E^+ \not\subseteq M^+(B)$$

- **Suficiencia a posteriori** *Todo* $e \in E^+$ *es verdadero en* $M^+(B \wedge H)$, *i.e.,*

$$E^+ \subseteq M^+(B \wedge H)$$

En la literatura podemos encontrar distintas variantes de estas definiciones. Así, Nienhuys-Cheng y de Wolf en [NCdW97] consideran que en la semántica *normal* el conocimiento básico y los ejemplos deben ser cláusulas. Incluso Muggleton y De Raedt, en [MDR94], distinguen el subcaso de la semántica definida en que los ejemplos son átomos sin variables.

El segundo gran grupo de sistemas se engloban dentro de la semántica no monótona[4]. El término *"no monótono"* fue introducido por Helft [Hel89] para enlazar estos sistemas con otras formas de razonamiento no monótono, debido a su relación con la Hipótesis del Mundo Cerrado y sus variantes. Los sistemas que siguen esta semántica tienen en común que la teoría aprendida no *implica* los ejemplos positivos, sino que determinan un conjunto de relaciones que son ciertas para el conjunto de ejemplos. Aquí los ejemplos son *interpretaciones de Herbrand.* El dato de entrada es un conjunto $\mathcal{I}^+$ de interpretaciones de Herbrand que son ejemplos positivos y un conjunto $\mathcal{I}^-$ de ejemplos negativos. El objetivo es encontrar un conjunto de cláusulas que sean válidas en todos los ejemplos positivos y ninguno de los negativos. Se puede ver cada interpretación de Herbrand como la "descripción"de una situación y el objetivo que exprese regularidades en los ejemplos positivos. Si M^+ representa el conjunto de *ejemplos positivos* y H es el conjunto de cláusulas que representa la teoría aprendida, deben verificarse los siguientes criterios:

- **Validez:** Toda $h \in H$ es válida en toda interpretación de M^+
- **Completitud:** Si una cláusula c es válida en toda interpretación de $M^+(B)$ entonces $H \models c$
- **Minimalidad:** No existe ningún subconjunto propio G de H que sea válido y completo.

[4]Para la presentación de la *semántica no monótona* seguimos [NCdW97, MDR94].

La condición de validez asegura que todas las cláusulas de H reflejan propiedades de los datos. La condición de completitud garantiza que toda la información válida en el conocimiento base está codificada en el conjunto hipótesis. Por último, el criterio de minimalidad evita la existencia de cláusulas redundantes.

2.12 Un algoritmo general de PLI

Estamos ahora en condiciones de presentar un algoritmo general de la PLI. En 1992 De Raedt y Bruynooghe [DRB92] presentaron un algoritmo general denominado GENCOL (GENeric COncept–Learning) en el que hacían abstracción de los principales sistemas de PLI que existían hasta el momento. El algoritmo de la figura 2.3 está tomado de [MDR94] y está basado en GENCOL.

Comenzar:QH
Repetir:
    Borrar H de QH
    Elegir las reglas de inferencia $r_1, \ldots, r_n$ que se van a aplicar sobre H
    Aplicar las reglas de inferencia elegidas y obtener $H_1, \ldots, H_n$
    Añadir $H_1, \ldots, H_n$ a QH
    Podar QH
Hasta que QH satisfaga los criterios de parada

Figura 2.3: GENCOL

Este algoritmo general parte de un conjunto inicial de hipótesis QH que dependerá de la implementación que tomemos y sobre el que se llevarán a cabo una serie de operaciones hasta que el conjunto verifique los criterios de parada. Los métodos *ascendentes* empiezan con una hipótesis muy específica, generalmente la hipótesis vacía, a partir de la cual vamos generalizando. Los métodos *descendentes* parten de una hipótesis muy general que debemos especializar con el uso de operadores. Las operaciones sobre QH son las siguientes:

1. Borrar. Los pasos dados en esta operación vienen marcados por la estrategia de búsqueda en el espacio de hipótesis. Al contrastar la hipótesis actual con el conocimiento base y los ejemplos observamos que ciertas cláusulas deben ser eliminadas (y posiblemente sustituidas por otras).

2. Elegir, Aplicar y Añadir. En estos pasos el sistema determina qué operadores deben ser aplicados para transformar la hipótesis actual, los aplica y añade al programa las cláusulas obtenidas.

3. Podar. El uso de los distintos operadores aumenta el tamaño del espacio de búsqueda, por lo que son necesarios mecanismos que nos permitan eliminar cláusulas antes de aplicar los criterios de parada.

En general sería deseable encontrar una hipótesis que cubriera *todos* los ejemplos positivos y ninguno de los negativos. En los problemas de la vida real el conjunto de ejemplos puede tener *ruido (noise)* y se considera suficiente que la hipótesis cubra un porcentaje de ejemplos suficientemente alto para detener el proceso.

2.13 Abducción y PLI

A mediados de los ochenta se empezó a considerar tímidamente el problema de la abducción desde la Programación Lógica y ha dado pie a dos campos de investigación: el primero bajo el nombre de **Programación Lógica Abductiva** [Pau93, KKT93]) y la segunda conocida como **Programación Lógica Inductiva.** Las fuentes de las que bebe esta memoria de investigación son las de la Programación Lógica Inductiva, y los matices que diferencian la PLI de la Programación Lógica Abductiva caen fuera del alcance de esta memoria. Sólo destacar que en muchas ocasiones las diferencias son relativas más a la terminología que al objetivo en sí mismo. La relación entre ambas aproximaciones ha sido estudiada en profundidad por P. Flach y A. Kakas (p.e. [FK00c, FK00a] o [FK98]). En el 2000, Flach y Kakas editaron una colección de ensayos de varios autores [FK00b] sobre la relación e integración de la inducción y la abducción.

La implementación de las distintas técnicas mediante las cuales buscamos una *explicación* para las observaciones (abducción) se conocen en la literatura como *reglas de inferencia inductivas.* Estas reglas de inferencia son en realidad *operadores* mediante los cuales llevamos a cabo el proceso de aprendizaje. Estos operadores pueden ser *ascendentes* o *descendentes* según generalicen o especialicen una determinada hipótesis.

Muggleton y De Raedt [MDR94] relacionan generalización con inducción y especialización con deducción, de forma que para ellos inducción y deducción son conceptos inversos. Declaran, a su vez, que adoptan esta postura, pese a ser

muy controvertida, porque ofrece la posibilidad de tratar la inducción de forma operativa.

Dedicaremos las siguientes secciones a presentar algunas de las técnicas que se llevan a cabo para intentar resolver el problema de la abducción en Programación Lógica Inductiva, esto es, *cómo* generar esas *hipótesis* que *expliquen* las observaciones.

2.14 Inversión de resolución

Los primeros operadores que vemos son un tipo de reglas inductivas presentadas por Muggleton y Buntine en [MB88] para la *inversión de resolución.* Las reglas de inferencia utilizadas para ello son de dos tipos, los denominados *V–operadores* y los *W– operadores.* Veamos ahora cómo se computan[5].

V–operadores

Recordemos por un momento la regla de inferencia *resolución* [Rob65] que queremos invertir. Dadas dos cláusulas C_1 y C_2 con las variables separadas, consideramos dos literales L_1 y L_2 de C_1 y C_2 respectivamente, tales que exista un unificador de máxima generalidad θ de $\neg L_1$ y L_2. Podemos probar que existen dos sustituciones θ_1 y θ_2 tales que $\theta = \theta_1\theta_2$ y $\neg L_1\theta_1 = L_2\theta_2$. A partir de ahí podemos inferir una nueva cláusula

$$C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\})\theta_2 \tag{2.1}$$

La figura 2.4 muestra esquemáticamente un paso de resolución. La resolución deriva la cláusula C en el vértice de la V a partir de las cláusulas C_1 y C_2 de los brazos.

Para la presentación de los V–operadores necesitamos dos cláusulas C_1 y C_2 arbitrarias (no necesariamente cláusulas de Horn), que puedan ser resueltas mediante los literales L_1 (positivo) y L_2 (negativo), esto es, que existan dos literales L_1 y L_2 en C_1 y C_2 respectivamente y unas sustituciones θ_1 y θ_2 tales que

$$\neg L_1\theta_1 = L_2\theta_2$$

Necesitamos imponer la llamada *asunción de separabilidad (separability assumption)* que exige que $(C_1 - \{L_1\})\theta_1$ y $(C_2 - \{L_2\})\theta_2$ no tengan literales en común.

[5]Los detalles de esta presentación están tomados de [LN92].

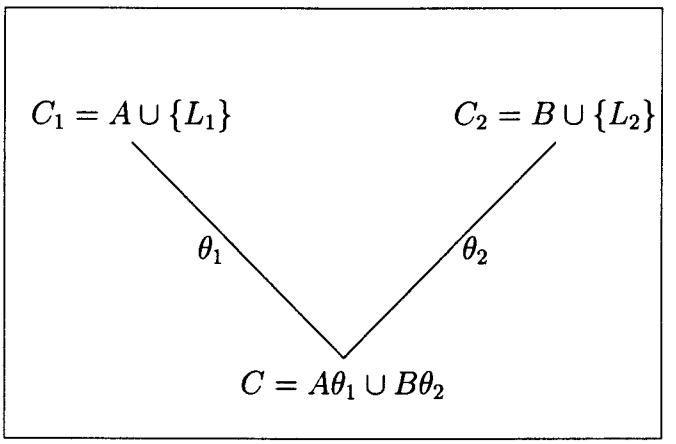


Figura 2.4: V–operador

Mediante una fácil manipulación de la ecuación 2.1 podemos obtener[6]

$$C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{L_2\} \tag{2.2}$$

El proceso de obtener C_2 a partir de C y C_1 se denomina *absorción*. De forma análoga se describe el proceso de obtener C_1 a partir de C y C_2 conocido como *identificación*. La ecuación 2.2 contiene cuatro variables desconocidas L_1, L_2, θ_1

Absorción:	$\dfrac{q \leftarrow A}{q \leftarrow A}$	$\dfrac{p \leftarrow A, B}{p \leftarrow q, B}$
Identificación:	$\dfrac{p \leftarrow A, B}{q \leftarrow B}$	$\dfrac{p \leftarrow A, q}{p \leftarrow A, q}$

Figura 2.5: V–operadores

y θ_2^{-1}. Las posibles elecciones de valores para estas variables van a determinar el enorme tamaño del espacio de hipótesis.

El sistema CIGOL [MB88] presentado por Muggleton y Buntine asumía que C_1 era una cláusula unidad, i.e., $C_1 = \{L_1\}$, por lo que $L_2 = \neg L_1\theta_1\theta_2^{-1}$. Con esto la ecuación (2.2) queda

$$C_2 = (C \cup \{\neg L_1\})\theta_1\theta_2^{-1} \tag{2.3}$$

[6]Dados un término t y una sustitución θ, existe una única *sustitución inversa* tal que $t\theta\theta^{-1} = t$. Ver [MB92] para los detalles.

y por tanto la indeterminación viene dada por las posibles elecciones de θ_1 y θ_2^{-1}.

W–operadores

Las otras dos reglas de inferencia para la inversión de resolución son la *interconstrucción* y la *intraconstrucción*, que reciben el nombre conjunto de W–operadores. Su estudio no es tan sencillo como el de los V– operadores, ya que estas nuevas reglas de inferencia introducen en el lenguaje un nuevo símbolo de predicado, por lo que forman parte de un problema más general de la PLI denominado *invención de predicados*[7].

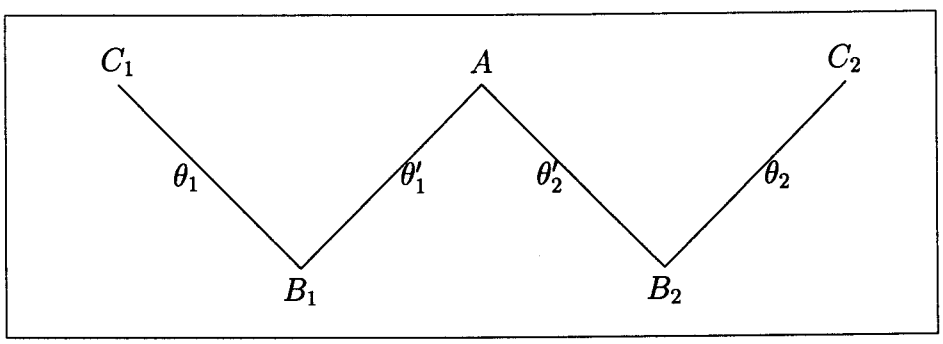


Figura 2.6: W–operador

Combinando juntos dos V–operadores obtenemos un W–operador (ver figura 2.6). Consideramos que las cláusulas C_1, C_2 resuelven con la cláusula A mediante el literal L y dan como resultado las cláusulas B_1 y B_2. El objetivo de los W–operadores es construir C_1, C_2 y A a partir de B_1 y B_2. El W–operador se denomina *intra–construcción* cuando L es negativo e *inter–construcción* cuando L es positivo.

Intra–construcción:

$$\frac{p \leftarrow A, B \qquad\qquad p \leftarrow A, C}{q \leftarrow A, B \quad p \leftarrow A, q \quad q \leftarrow A, C}$$

Inter–construcción:

$$\frac{p \leftarrow A, B \qquad\qquad q \leftarrow A, C}{p \leftarrow r, A, B \quad r \leftarrow A \quad q \leftarrow r, A, C}$$

Figura 2.7: W–operadores

[7]Un estudio en profundidad sobre la invención de predicados en PLI la podemos encontrar en los trabajos de I. Sthal (p.e [Sta93, SW94, Sta94, Sta95] o en Muggleton [Mug94] o en Ling [Lin91].

Puesto que hacemos resolución con A y C_1 y con A y C_2 para obtener B_1 y B_2 sobre el mismo literal L, puede ocurrir que el símbolo de predicado de L no ocurra en B_1 ni en B_2. Por tanto, si queremos obtener C_1, C_2 y A a partir de B_1 y B_2 tendremos que dotar al sistema que la capacidad de *inventar predicados.*

El sistema CIGOL tiene implementada una forma restringida de intra-construcción en la cual las cláusulas C_i constan de un único literal.[8]

2.15 Negación como regla de fallo

Otra de las soluciones al problema de la abducción es la presentada por Kakas *et al.* en [KKT93] donde usan la SLDNF–resolución como técnica de abducción[9].

La idea es la siguiente: Cuando una SLDNF–derivación falla al intentar unificar un literal seleccionado, un *subobjetivo*, con la cabeza de una cláusula, ese subobjetivo puede ser considerado como una hipótesis H, puesto que si ampliamos el programa con esa hipótesis H la derivación podría continuar.

Veamos un ejemplo

Ejemplo 2.8 Supongamos que tenemos un programa P con una sola cláusula:

$$mortal(X) \leftarrow hombre(X)$$

y realizamos la siguiente observación O:

$$mortal(Sócrates) \leftarrow$$

nuestra intención encontrar una hipótesis H tal que $P \cup H \models O$. Para ello intentamos encontrar una refutación a partir del objetivo G:

$$\leftarrow mortal(Sócrates)$$

Puesto que $P \not\models O$, no podemos encontrar una refutación de $P \cup \{G\}$, por lo que las derivaciones o son infinitas, o terminan en una cláusula cuyo literal seleccionado no unifica con la cabeza de ninguna cláusula del programa. En nuestro caso, mediante la unificación $\theta = \{X/Sócrates\}$ del literal del objetivo

[8] Ver [MB92] para los detalles.

[9] La idea ya aparece en Cox y Pietrzykowski [CP86].

con la cabeza de la cláusula del programa tenemos

$$\leftarrow hombre(Sócrates)$$

y la derivación termina ahí, puesto que el literal $hombre(Sócrates)$ no unifica con la cabeza de ninguna cláusula del programa por lo que esa derivación *falla*. La solución propuesta por Kakas *et al.* es considerar como hipótesis H la cláusula

$$hombre(Sócrates) \leftarrow$$

y con esto $P \cup H \models O$, esto es, si sabemos que todo hombre es mortal y vemos que Sócrates es mortal, podemos suponer que esto ocurre porque Sócrates es hombre.

2.16 Subsunción

El cuerpo principal de la presente memoria se basa en el estudio de operadores de aprendizaje basados en la relación de subsunción y de consecuencia y las nociones básicas relacionadas con la relación de subsunción entre cláusulas podemos encontrarlas en el apéndice A, no obstante hacemos aquí una breve reseña.

La relación de subsunción entre cláusulas fue estudiada por Plotkin [Plo70, Plo71] antes de que existiera la Programación Lógica. Según su definición, decimos que la cláusula C_1 θ–subsume (o simplemente *subsume*) a la cláusula C_2 si existe una sustitución θ tal que $C_1\theta \subseteq C_2$. En tal caso diremos que C_1 es una generalización de C_2 (y C_2 una especialización de C_1) bajo subsunción. Nótese que si C_1 θ–subsume a C_2 entonces $C_1 \models C_2$. La otra implicación no es cierta, como ilustra el siguiente ejemplo:

Ejemplo 2.9 Sean C_1 y C_2 las siguientes cláusulas

$$C_1 = \{p(f(X)), \neg p(X)\}$$
$$C_2 = \{p(f(f(Y))), \neg p(Y)\}$$

Se tiene que $C_1 \models C_2$, pero no existe ninguna sustitución θ tal que $C_1\theta \subseteq C_2$.

Plotkin [Plo70] dio un algoritmo, ya clásico, de construcción de la menor generalización general bajo subsunción de dos cláusulas dadas C_1 y C_2:

Sea $\phi : TERM \times TERM \rightarrow V$ una biyección[10]. Calculamos con ayuda de ϕ la *mgg* de dos términos

$$mgg(s,t) = \begin{cases} f(mgg(s_1,t_1),\ldots,mgg(s_n,t_n)) & \text{si} \quad s = f(s_1,\ldots,s_n) \\ & \text{y} \quad t = f(t_1,\ldots,t_n) \\ \phi(s,t) & \text{en otro caso} \end{cases}$$

La *mgg* de dos átomos

$$mgg(s,t) = \begin{cases} p(mgg(s_1,t_1),\ldots,mgg(s_n,t_n)) & \text{si} \quad s = p(s_1,\ldots,s_n) \\ & \text{y} \quad t = p(t_1,\ldots,t_n) \\ \textit{No definida} & \text{en otro caso} \end{cases}$$

El *mgg* de dos literales se define de manera natural y por último la *mgg* de dos cláusulas C_1 y C_2 se define

$$mgg(C_1,C_2) = \{mgg(L_1,L_2) : L_1 \in C_1,\ L_2 \in C_2\}$$

2.17 Subsunción relativa

Plotkin, en su tesis doctoral [Plo71] extiende la noción de subsunción entre cláusulas a la *subsunción relativa.* En primer lugar define las *c*-derivaciones

Definición 2.10 *Una derivación basada en resolución D de la cláusula C a partir de la conjunción de cláusulas T es una c–derivación si cada cláusula de T aparece a lo sumo una vez en D.*

Veamos ahora cómo se define la subsunción relativa.

Definición 2.11 *Diremos que la conjunción de cláusulas T subsume relativamente a la cláusula C si existe una c–derivación de una cláusula D a partir de T tal que D subsuma a C.*

Podemos definir por tanto la menor generalización general de dos cláusulas relativas a un programa.

[10]La elección de una u otra biyección ϕ es irrelevante, puesto que la *mgg* es única salvo renombramiento de variables.

UNIVERSIDAD DE SEVILLA
FACULTAD DE MATEMÁTICAS
BIBLIOTECA

Definición 2.12 *Dado un conocimiento base K, la menor generalización general relativa de las cláusulas C_1 y C_2, $mggr_K(C_1, C_2)$, es el supremo de C_1 y C_2 en el orden establecido por la subsunción relativa.*

Plotkin probó que la *mggr* de dos cláusulas no es necesariamente finita, no obstante, Muggleton y Feng [MF92] prueban que con determinadas restricciones sobre el lenguaje, (la ij-determinación) se puede construir una *mggr* finita de dos cláusulas. Esta técnica del cálculo de subsunción relativa es la base de su sistema GOLEM. Ilustramos esta técnica con un ejemplo tomado de [LD94]:

Ejemplo 2.13 Dados los ejemplos positivos

$$e_1 = hija(maria, ana)$$
$$e_2 = hija(eva, tomas)$$

y el conocimineto base $\mathcal{B}$ del ejemplo 2.5, la menor generalización general de e_1 y e_2 relativa a $\mathcal{B}$ se computa como

$$mggr_{\mathcal{B}}(e_1, e_2) = mgg((e_1 \leftarrow K)(e_2 \leftarrow K))$$

donde K es la conjunción de literales

$$\begin{array}{lll} progenitor(ana, maria) & \wedge & progenitor(ana, tomas) \quad \wedge \\ progenitor(tomas, eva) & \wedge & mujer(ana) \quad \wedge \\ mujer(maria) & \wedge & mujer(eva) \end{array}$$

Después de eliminar los *literales irrelevantes* (ver [MF90]) el sistema GOLEM encuentra

$$hija(X, Y) \leftarrow mujer(X), progenitor(Y, X)$$

2.18 Inversión de implicación

El orden de generalidad natural en PLI es el orden de subsunción, pese a que, como hemos visto, si C_1 y C_2 son cláusulas y C_1 subsume a C_2 entonces $C_1 \models C_2$ y la implicación inversa no es cierta. La razón es simple: La subsunción entre cláusulas es decidible, mientras que la implicación no lo es [SS88, MP92]. Además, la cuestión de saber si dos cláusulas de Horn tienen una *menor generalización bajo implicación* en el conjunto de las cláusulas de Horn ha sido resuelta

negativamente, lo que añade una nueva dificultad al diseño de algoritmos. El siguiente ejemplo está tomado de [MDR94].

Ejemplo 2.14 Consideremos las cláusulas siguientes

$$\begin{array}{lll} C_1 = p(f(x)) \leftarrow p(x) & & D_1 = p(f(f(x))) \leftarrow p(x) \\ C_2 = p(f(f(y))) \leftarrow p(x) & & D_2 = p(f(f(f(x)))) \leftarrow p(x) \end{array}$$

Se tiene que $C_1 \models \{D_1, D_2\}$ y $C_2 \models \{D_1, D_2\}$ y además, cualquier cláusula de Horn que sea más específica que C_1 o C_2 no implica simultáneamente D_1 y D_2. Así C_1 y C_2 son ambas *mínimas generalizaciones bajo implicación* de $\{D_1, D_2\}$. Puesto que C_1 y C_2 son incomparables bajo implicación, no existe *la menor generallización bajo implicación* de $\{D_1, D_2\}$ en el conjunto de cláusulas de Horn. No obstante, Muggleton y Page probaron en [MP94] que la cláusula

$$C = \{p(f(x)), p(f(f(x))), \neg p(x)\}$$

es la *menor generalización bajo implicación* de $\{D_1, D_2\}$.

Estas limitaciones no han impedido que S. Muggleton tome la inversión de implicación como base para su sistema Progol [Mug95] en el cual se imponen fuertes restricciones sobre el lenguaje. La idea es generar sólo la hipótesis más específica bajo implicación dentro del lenguaje permitido.

2.19 Otros aspectos de la PLI

En las secciones anteriores hemos presentado algunas de las técnicas usadas en PLI para buscar una explicación a las observaciones con ayuda de un concimiento base. Por supuesto, existen más técnicas de abducción. No hemos comentado, por ejemplo, la *subunificación* de [LM92] o la búsqueda con el uso de patrones del sistema RDT [KW92] integrado en Mobal [MWKE93].

La lista de sistemas está en continuo crecimiento y cada sistema pone el énfasis en un determinado aspecto de la búsqueda de expliciones. Así, para el estudio y diseño de sistemas PLI tenemos que tener en cuenta aspectos generales que se engloban bajo el nombre de *sesgo*.

Según Utgoff y Mitchell [UM82] el sesgo es todo aquello que influye en los procesos de inferencia inductiva a partir de la evidencia. El concepto de sesgo en PLI es amplio, pero de manera genérica podemos hablar de dos tipos de sesgo:

el *sesgo declarativo*, que define el espacio de hipótesis que debe ser considerado en el proceso de aprendizaje y el *sesgo preferencial*, que determina cómo buscar en el espacio. Estas elecciones en el diseño del sistema de aprendizaje en las que hacemos *suposiciones a priori* sobre la naturaleza del concepto que debemos aprender son cruciales en el proceso de aprendizaje, hasta el punto que, como afirma Mitchell en [Mit97] (p. 42): *"Un sistema de aprendizaje que no hace suposiciones a priori respecto a la identidad del concepto objetivo no tiene base racional para clasificar instancias no observadas"*.

Otros aspectos como la *"capacidad de aprender"(learnability)*, cuyo marco va mucho más allá del aprendizaje basado en lógica clausal también tiene interés dentro de la PLI. Las dos principales aproximaciones al problema son la *identificación en el límite* de Gold [Gol67] y el aprendizaje PAC de Valiant [Val84]. Podemos encontrar diversos resultados sobre la *capacidad de aprender* de los sistemas en trabajos de De Raedt y Bruynooghe [RB92], Banerji [Ban87], Kietz [Kie93] o Cohen [Coh93]. Otros aspectos pueden ser la *invención de predicados* (ver sección 2.14) o el manejo de datos con *ruido* (p.e. en [Für93, Für94, GL01]).

Como vemos, podemos abordar el problema de la PLI desde muchos puntos de vista: Podemos poner el acento en el estudio de los procesos abductivos e intentar reproducirlos en las máquinas basándonos en una representación clausal, podemos estudiar propiedades generales de los algoritmos ya existentes intentando modelar sus limitaciones y capacidades como sistemas de aprendizaje, etc. Otras aproximaciones centran sus esfuerzos en la búsqueda de nuevos algoritmos o depuración de los ya existentes con el objetivo de reducir costes computacionales.

Por último, destacar el enorme éxito que han tenido los sistemas PLI en problemas del mundo real y la creciente lista de dominios de aplicación donde se está usando con éxito, en campos como la biología (p.e. [KK01, KKCD00, KS95] o [TMS01]), procesamiento del lenguaje natural (p.e. [Kaz99, KME99]) o ingeniería [DIJ94, Fen92, Pop98], por citar algunos[11].

[11] S. Dzeroski y L. Todorovski mantienen una página sobre aplicaciones de la PLI en `http://www-ai.ijs.si/~ilpnet2/apps/index.html`.

Capítulo 3

Operadores clausales

El objetivo de la PLI es encontrar un programa lógico que clasifique las instancias observadas de acuerdo con un conocimiento base. Para ello, partimos de un programa al que vamos aplicando sucesivas transformaciones. Como hemos visto, una de las técnicas básicas de los sistemas PLI es la transformación de una cláusula en otra mediante el uso de un operador. Después de sucesivas transformaciones de las cláusulas del programa inicial se espera que la sucesión de programas obtenida converja, en algún sentido, al programa objetivo.

Una de las posibilidades para esta transformación es la aplicación de un operador de refinamiento a una única cláusula y el estudio de la influencia que este cambio tiene sobre el comportamiento del programa. En el presente capítulo estudiamos estos operadores de refinamiento y proponemos la definición de un operador clausal donde cada operador se define como una aplicación del conjunto de cláusulas $\mathbb{C}$ en sí mismo con las siguientes características:

- Dada una cláusula C obtenemos como imagen de C una única cláusula, esto es, el operador de refinamiento es una función de $\mathbb{C}$ en $\mathbb{C}$ en sentido matemático.

- Una vez fijado el operador, dada cualquier cláusula C, podemos encontrar de manera efectiva la cláusula resultante de aplicar el operador sobre C. Esto representa una ventaja desde el punto de vista computacional respecto a otras aproximaciones, puesto que, como describimos en el apéndice A, en los operadores clausales descritos por Nienhuys–Cheng [NCdW97, vdLNC98] la imagen de una cláusula por el operador era un conjunto de cláusulas posiblemente infinito.

- Otra ventaja está relacionada con la composición de operadores. Al ser el operador una aplicación de $\mathbb{C}$ en $\mathbb{C}$ y poder conocer de manera explícita la imagen de cualquier cláusula, la composición de operadores también es una aplicación de $\mathbb{C}$ en $\mathbb{C}$ y podemos conocer y computar cual es el resultado de aplicar de manera sucesiva varios operadores a una cláusula. Esto está en contraposición con la dificultad en el manejo de las definiciones de operador de refinamiento en las que la imagen de una cláusula es un conjunto de cláusulas y resulta difícil saber si una cláusula D pertenece o no a la imagen de una cláusula C tras aplicar varios operadores de refinamiento.

Al final del capítulo veremos que con nuestra definición obtenemos la máxima capacidad expresiva que podemos esperar en un operador clausal, esto es, dadas dos cláusulas cualesquiera C y D, podemos hacer sucesivas manipulaciones sintácticas, cada una de ellas asociada a un operador, de manera que partiendo de C obtengamos D. Dicho de otro modo, si $\mathbb{OP}$ es el conjunto de los operadores clausales, $\mathbb{OP}^*$ es el cierre por composición de $\mathbb{OP}$ y C y D son dos cláusulas cualesquiera, entonces existe $f \in \mathbb{OP}^*$ tal que $f(C) = D$.

Este resultado nos permite afirmar que si buscamos operadores clausales de $\mathbb{C}$ en $\mathbb{C}$ útiles para el Aprendizaje Automático no tenemos que buscar fuera de $\mathbb{OP}^*$. En capítulos posteriores definiremos un subconjunto apropiado de $\mathbb{OP}^*$ con el que caractericemos las relaciones entre cláusulas que nos permita formalizar la relación *"más general que"* esenciales en PLI.

3.1 Definiciones básicas

Dedicamos esta sección a fijar la notación y las definiciones básicas que usaremos a continuación. Cualquier otra definición puede encontrarse en [Llo87] o [NCdW97].

En toda la memoria consideramos un lenguaje de primer orden $\mathcal{L}$ numerable con al menos un símbolo de función. Var es el conjunto de sus variables, $Pred$ el conjunto de sus símbolos de predicado, $Term$ el conjunto de sus términos, Lit el conjunto de literales formados con símbolos del lenguaje y $B_{\mathcal{L}}$ la base de Herbrand del lenguaje. En esta memoria no estamos interesados en el significado procedural de las cláusulas y no consideramos el orden entre sus literales, esto es, cláusulas como

$$padre(x, y) \leftarrow progenitor(x, y), hombre(x)$$
$$padre(x, y) \leftarrow hombre(x), progenitor(x, y)$$

se consideran iguales, por tanto definimos una *cláusula* como un conjunto finito de literales. Denotamos por $\mathbb{C}$ el conjunto de todas las cláusulas. Un programa es un conjunto finito no vacío de cláusulas no vacías. Denotaremos por $\mathbb{P}$ el conjunto de todos los programas.

Una cláusula *definida* es una cláusula que contiene un literal positivo y cero o más literales negativos. Un *programa definido* es un conjunto finito no vacío de cláusulas definidas.

Si $C = \{A, \neg B_1, \ldots, \neg B_n\}$ es una cláusula definida el operador T_C de consecuencia de Kowalski y van Emden [VK76] es una aplicación $T_C : 2^{B_\mathcal{L}} \to 2^{B_\mathcal{L}}$ tal que si $I \subseteq B_\mathcal{L}$ entonces

$$T_C = \{A' \in B_\mathcal{L} \mid A' \leftarrow B'_1, \ldots, B'_n \in Bas(C) \wedge \{B'_1, \ldots, B'_n \subseteq I\}\}$$

donde $Bas(C)$ es el conjunto de todas las instancias básicas de C. De manera análoga, si P es un programa definido el operador T_P de consecuencia es una aplicación $T_P : 2^{B_\mathcal{L}} \to 2^{B_\mathcal{L}}$ tal que si $I \subseteq B_\mathcal{L}$ entonces

$$T_P = \{A' \in B_\mathcal{L} \mid \exists\, A' \leftarrow B'_1, \ldots, B'_n \in Bas(P) \wedge \{B'_1, \ldots, B'_n \subseteq I\}\}$$

donde $Bas(P)$ es el conjunto de todas las instancias básicas de cláusulas de P.

Si A es un conjunto, entonces $\mathcal{P}A$ es el conjunto de sus partes y $|A|$ representa el cardinal del conjunto. De este modo, denotamos por $|C|$ al número de literales de la cláusula C. Si f y g son aplicaciones, $g \circ f$ es la composición de f y g.

Una relación binaria sobre un conjunto es un quasi-orden si verifica las propiedades reflexiva y transitiva. Si la relación verifica las propiedades reflexiva, simétrica y transitiva entonces es una relación de equivalencia.

Sea $S \subseteq Var$ un conjunto finito de variables. Una *sustitución* es una aplicación $\theta : S \to Term$ tal que $(\forall x \in S)[x \neq \theta(x)]$. Usamos la notación usual $\theta = \{x/t : x \in S\}$, (donde $t = \theta(x)$, o $x\theta = t$). Un par x/t se llama un *enlace*. El conjunto S es el dominio de θ y se representa por $Dom(\theta)$. Consideraremos de manera natural la extensión de la definición de sustitución a términos, literales y cláusulas.

La sustitución vacía la representamos, como es habitual, por ϵ. Diremos que una sustitución es *elemental* si está formada por un único enlace, esto es, si es de la forma $\theta = \{x/t\}$. Como es usual, usaremos la expresión *umg* para referirnos al unificador de máxima generalidad.

Un *renombramiento* es una sustitución inyectiva θ tal que

$$(\forall x \in Dom(\theta))\,[x\theta \in Var]$$

Si A es un átomo, diremos que los literales A y $\neg A$ son complementarios.

Sean L_1, L_2 dos literales. Se dice que L_1 es una *instancia* de L_2, si existe una sustitución θ tal que $L_1 = L_2\theta$. Si θ es un renombramiento de variables y L un literal, entonces decimos que $L\theta$ es una *variante* de L y que L y $L\theta$ son *variantes*. Análogamente, si C es una cláusula y θ un renombramiento, $C\theta = \{L\theta \mid L \in C\}$ es una variante de C y diremos que C y $C\theta$ son variantes.

En nuestra definición de los operadores usaremos inserciones y sustituciones de términos. En ellas el símbolo de predicado y signo de un literal no variará, por lo que será útil la siguiente definición.

Definición 3.1 *Sea L un literal. Se define $par(L) = \langle p, s\rangle$ donde p es el símbolo de predicado de L y s su signo $(+)$ o $(-)$. Si C es una cláusula. se define*

$$SP(C) = \{par(L) \mid L \in C\}$$

Por último, se define el conjunto SP como el conjunto de todos los pares que se pueden formar con los símbolos de predicado del lenguaje dado.

El siguiente lema es una adaptación del lema 2.5 de [Apt97].

Lema 3.2 *Sean L_1 y L_2 dos literales. L_1 es una variante de L_2 si y sólo si L_1 es una instancia de L_2 y L_2 es una instancia de L_1.*

3.2 Motivación

Como vimos en el capítulo 2, una de las técnicas de abducción usadas en la implementación de sistemas PLI es la transformación de una cláusula en otra mediante el uso de un operador. Se espera que después de sucesivas transformaciones de las cláusulas, el programa inicial converja, en algún sentido, al programa objetivo. Desde un punto de vista general, las técnicas de abducción usadas en PLI pueden verse como transformaciones resultantes de aplicar un operador que nos lleva una cláusula en otra. Veamos con un par de ejemplos cómo dos herramientas básicas en PLI pueden verse bajo este punto de vista.

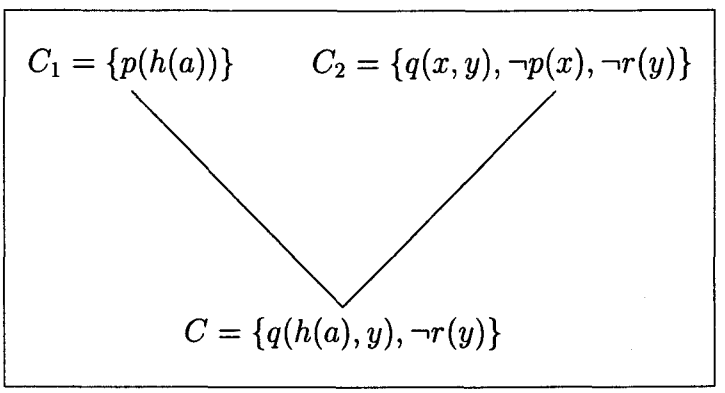


Figura 3.1: Absorción

Ejemplo 3.3 Sea L el literal $p(h(a))$ y sean C_1 y C las cláusulas $C_1 = \{L\}$ y $C = \{q(h(a)), y), \neg r(y)\}$. Podemos usar el operador *absorción* para obtener

$$C_2 = \{q(x,y), \neg p(x), \neg r(y)\}$$

La cláusula C_2 la obtenemos mediante la transformación de la cláusula $C \cup \{\neg L\}$ por un operador ψ_1 que invierte la sustitución $\theta = \{x/h(a)\}$.

Ejemplo 3.4 Supongamos que en un sistema ascendente después de saturar el ejemplo $p(a)$ y realizar varios pasos de reducción obtenemos la cláusula

$$C = \{p(X), \neg q(X, f(a)), \neg r(X,Z), \neg menor\_que(Z,5), \neg menor\_que(Z,10)\}$$

El literal $menor\_que(Z,10)$ no aporta información y podemos aplicar el operador $\psi_2 \equiv$ `eliminar_el_literal_`$menor\_que(Z,10)$ a la cláusula C para obtener

$$\psi_2(C) = \{p(X), \neg q(X, f(a)), \neg r(X,Z), \neg menor\_que(Z,5)\}$$

Estos ejemplos nos llevan a reconsiderar las técnicas usadas en PLI como aplicación de operadores que transforman una cláusula en otra y que pueden ser consideradas dentro de un marco general de *operadores clausales*. Dedicaremos este capítulo a introducir estos operadores y a demostrar sus propiedades básicas.

3.3 Posiciones e inserciones

En esta sección presentamos las definiciones de posiciones e inserciones necesarias en la definición de nuestros operadores. Seguimos a Gallier [Gal86] y Huet [Hue80].

Definición 3.5 *Una posición es una n–upla de enteros positivos, con n un número natural positivo. Si u es una posición, notaremos por longitud(u) a ese número n.*

- *Notaremos por* $\mathbb{N}^+$ *al conjunto de todas las posiciones.*
- *Se dice que dos posiciones u y v son* independientes *si u no es un prefijo de v y v no es un prefijo de u. Un conjunto de posiciones P se dice independiente si* $(\forall u, v \in P)[u \neq v \Rightarrow$ *u y v son independientes*$]$.

En nuestra descripción del operador será fundamental el manejo de las distintas posiciones que ocupa un mismo término en un literal.

Definición 3.6 *La posición de* t_i *en los literales* $q(t_1, \ldots, t_n)$ *y* $\neg q(t_1, \ldots, t_n)$ *es* $\langle i \rangle$. *Si el término* $f(s_1, \ldots, s_m)$ *ocupa la posición* $\langle p_1, \ldots, p_k \rangle$, *entonces* s_j *tiene la posición* $\langle p_1, \ldots, p_k, j \rangle$. *Escribiremos* $p_1 \cdot p_2 \cdot \cdots \cdot p_n$ *en lugar de* $\langle p_1, \ldots, p_n \rangle$.

Definición 3.7 *Adoptaremos las siguientes notaciones:*

- *Dado un literal L,* $Pos(L)$ *es el conjunto (posiblemente vacío) de todas las posiciones de L.*
- *Dado un literal L y un término t,* $Pos(L, t)$ *es el conjunto de todas las posiciones de t en L. Si t no ocurre en L, entonces* $Pos(L, t) = \emptyset$.
- *Si L es un literal y u una posición de L,* L/u *denotará el término que ocupa la posición u en el literal L.*
- *Sea L un literal y P un conjunto de posiciones, entonces* $L/P = \{L/u \mid u \in P\}$

Nota 3.8 Sea $P \subseteq Pos(L)$. Nótese que

$$|L/P| = 0 \Leftrightarrow L/P = \{L/u \mid u \in P\} = \emptyset \Leftrightarrow P = \emptyset$$

Cuando aplicamos la sustitución $\theta = \{x/a\}$ al literal $L_1 = p(x, a)$ y obtenemos $L_2 = L_1\theta = p(a, a)$, estamos insertando la constante a en *todas* las posiciones que ocupa la variable x en L_1. Pero para poder definir la *sustitución inversa* que nos permita obtener L_1 a partir de L_2 tenemos que determinar el conjunto de posiciones de la constante a en la que queremos insertar la variable x. Esta idea de insertar términos en distintas posiciones es fundamental en nuestra descripción.

Definición 3.9 *Sea $\{t_1, \ldots, t_n\}$ un conjunto de términos, p un símbolo de función o predicado n-ario, $s \in Term$ y u una posición. La* inserción *de s en la posición u de $p(t_1, \ldots, t_n)$, denotada por $p(t_1, \ldots, t_n)[u \leftarrow s]$, se define de la siguiente manera:*

- *Si $longitud(u) = 1$, entonces*

$$p(t_1, \ldots, t_n)[u \leftarrow s] = p(t_1, \ldots, t_{u-1}, s, t_{u+1}, \ldots, t_n)$$

- *Si $u = i\hat{}v$ con $i \geq 1$ y $v \in \mathbb{N}^+$, entonces*

$$p(t_1, \ldots, t_n)[u \leftarrow s] = p(t_1, \ldots, t_{i-1}, t_i[v \leftarrow s], t_{i+1}, \ldots, t_n)$$

A esta definición le corresponde el predicado `inserta_pos/4` del apéndice B.

Nótese que si $u, v \in \mathbb{N}^+$ son independientes y $s \in Term$, entonces

$$(L[u \leftarrow s])[v \leftarrow s] = (L[v \leftarrow s])[u \leftarrow s]$$

Definición 3.10 *Sea L un literal, s un término y $P \subseteq Pos(L)$ un conjunto independiente de posiciones. La inserción de s mediante P en L se define como sigue:*

- *Si $P = \emptyset$, entonces $L\{P/s\} = L$*
- *Si $u \in P$ y $P' = P \backslash \{u\}$, entonces $L\{P/s\} = (L\{P'/s\})[u \leftarrow s]$.*

Podemos extender de manera natural esta definición al caso en que P sea un subconjunto independiente de posiciones arbitrario (no necesariamente contenido en $Pos(L)$). Para ello basta considerar que si P y $Pos(L)$ son disjuntos entonces $L\{P/t\} = L$

A esta definición le corresponde el predicado `inserta_en_posiciones/4` en el apéndice B.

3.4 Operadores clausales

La idea que guía la definición de nuestros operadores clausales es encontrar una descripción unificada de los procesos de generalización. Para fijar ideas, en una primera aproximación, podemos pensar en la relación de subsunción como formalización de *ser más general que.*

Si C y D son cláusulas tales que C subsume a D entonces existe una sustitución θ tal que $C\theta \subseteq D$. Si $\sigma_1, \ldots, \sigma_n$ son sustituciones elementales tales que $\theta = \sigma_1 \ldots \sigma_n$ (ver Teorema 4.5), $C = C_0$ y $C_j = C_{j-1}\sigma_j$, $1 \leq j \leq n$, se tiene

$$C = C_0 \xrightarrow{\sigma_1} C_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_{n-1}} C_{n-1} \xrightarrow{\sigma_n} C_n = C\theta \subseteq D$$

La idea es obtener $n+1$ funciones que notaremos

$$\begin{array}{rccc} \{\Delta/t\}: & \mathbb{C} & \longrightarrow & \mathbb{C} \\ & C & \mapsto & C\{\Delta/t\} \end{array}$$

tales que $D\{\Delta_{n+1}/x_{n+1}\} = C\theta$ y $C_j\{\Delta_j/x_j\} = C_{j-1}$, $1 \leq j \leq n$.

No obstante, en este capítulo, presentamos una definición mucho más general. Vamos a definir los *operadores clausales* basándonos en la sustitución de unos términos por otros y la eliminación de literales en determinadas posiciones.

Veamos en primer lugar un ejemplo. Consideremos la siguiente cláusula con un único literal $C_1 = \{L\}$ donde $L = p(f(x), a, f(x))$. Veamos cómo aplicamos nuestro operador sobre C_1. Para ello,

(a) Elegimos un término t en L, por ejemplo $t = f(x)$,

(b) Elegimos varios subconjuntos de $Pos(L,t) = \{1,3\}$, e.g. $P_1 = \{1\}, P_2 = \emptyset, P_3 = \{1,3\}$,

(c) Elegimos el término que queremos insertar, en este caso, la variable z_1, y

(d) Construimos la cláusula

$$\begin{aligned} C_2 &= \{L[P_i \leftarrow z_1] \mid i = 1,2,3\} \\ &= \{p(z_1, a, f(x)), p(f(x), a, f(x)), p(z_1, a, z_1)\} \end{aligned}$$

De este modo hemos obtenido una cláusula C_2 tal que $C_2\{z_1/t\} = C_1$,

Si la cláusula tiene varios literales, por ejemplo, $C_1 = \{L_1, L_2, L_3\}$, con $L_1 = p(f(x))$, $L_2 = q(b, f(x))$ y $L_3 = r(f(x), h(f(x)))$, efectuamos la operación sobre

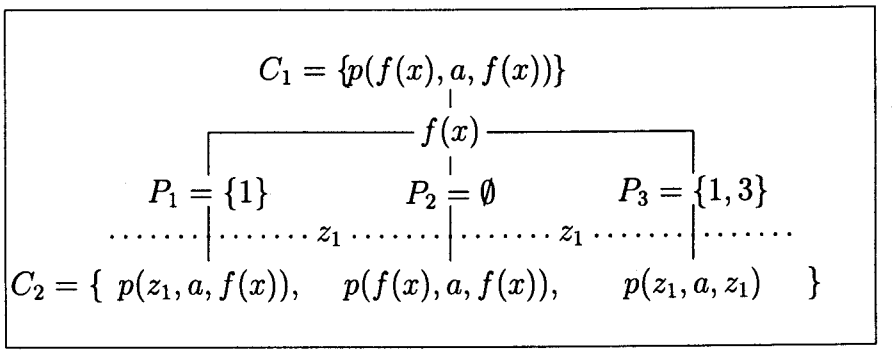


Figura 3.2: Ejemplo

todos los literales *simultáneamente*. En primer lugar, elegimos el mismo término en todos los literales de C_1, pongamos $t = f(x)$. Entonces, para todo literal $L_1 \in C_1$, elegimos algunos subconjuntos de $Pos(L_i, t)$, e.g.,

$$P_1^* = \{\{1\}\} \subseteq \mathcal{P}Pos(L_1, t)$$
$$P_2^* = \{\{2\}, \emptyset\} \subseteq \mathcal{P}Pos(L_2, t)$$
$$P_3^* = \{\{1, 2{\cdot}1\}, \{1\}\} \subseteq \mathcal{P}Pos(L_3, t)$$

Después tomamos un término, por ejemplo la variable z_1, y construimos los siguientes conjuntos

$$L_1 \xrightarrow{P_1^*} \{p(z_1)\}$$
$$L_2 \xrightarrow{P_2^*} \{q(b, z_1), q(b, f(x))\}$$
$$L_3 \xrightarrow{P_3^*} \{r(z_1, h(z_1)), r(z_1, h(f(x)))\}$$

Finalmente, la cláusula C_2 es la unión de estos conjuntos

$$C_2 = \{p(z_1), q(b, z_1), q(b, f(x)), r(z_1, h(z_1)), r(z_1, h(f(x)))\}$$

y, obviamente, $C_2\{z_1/t\} = C_1$. En nuestra descripción general empezamos con un estudio de las relaciones entre sustituciones e inserciones.

3.5 Sustituciones vs. inserciones

Las siguientes definiciones muestran cómo podemos determinar un término en una cláusula. Para ello necesitamos definir el concepto de *compatibilidad* a varios

niveles. En primer lugar definimos la compatibilidad de un conjunto de posiciones P respecto a un par $\langle L, t\rangle$ donde L es un literal y t es un término. En el siguiente nivel consideramos un conjunto de conjunto de posiciones y generalizamos la definición de compatibilidad del nivel anterior.

Definición 3.11 *Sea L un literal y t un término. Un conjunto de posiciones P se dice compatible con el par $\langle L, t\rangle$ si $P \subseteq Pos(L, t)$.*

A esta definición le corresponde el predicado `compatible_1/3` en el apéndice B.

Por ejemplo, si $L = p(f(x), a, f(x))$ y $t = f(x)$, entonces $P_1 = \{1\}$, $P_2 = \emptyset$ y $P_3 = \{1, 3\}$ son compatibles con $\langle L, t\rangle$ y $P_4 = \{1 \cdot 1, 2\}$ y $P_5 = \{1, 4 \cdot 3\}$ no lo son. Más aún, nótese que:

(a) Si P es compatible con $\langle L, t\rangle$, entonces P es independiente

(b) $P = \emptyset$ es compatible con cualquier par $\langle L, t\rangle$

(c) Si t no ocurre en L entonces P es compatible con $\langle L, t\rangle$ si y sólo si $P = \emptyset$.

La siguiente definición extiende la compatibilidad a conjuntos de conjuntos de posiciones

Definición 3.12 *Sea P^* un conjunto cuyos elementos son conjuntos de posiciones, esto es, $P^* \in \mathcal{P}\mathcal{P}\mathbb{N}^+$. Sea L un literal y t un término. Se dice que P^* es* compatible *con el par $\langle L, t\rangle$ si todo elemento $P \in P^*$ es compatible con $\langle L, t\rangle$.*

A esta definición le corresponde el predicado `compatible_2/3` en el apéndice B.

Por ejemplo, si $L = p(f(x), a, f(x))$ y $t = f(x)$, consideremos $P_1 = \{1, 3\}$, $P_2 = \{1\}$, $P_3 = \emptyset$ y $P_4 = \{2\}$. Supongamos que $P_1^* = \{P_1, P_2, P_3\}$ y $P_2^* = \{P_2, P_4\}$. Entonces P_1^* es compatible con $\langle L, t\rangle$ y P_2^* no lo es.

3.6 Operadores

Las aplicaciones que presentamos a continuación son fundamentales en la definición de nuestro operador. Según apuntamos en el ejemplo, la clave está en determinar un conjunto de posiciones en cada literal, ocupadas todas por el mismo término. Esto se hace mediante las asignaciones de posiciones a literales que definimos a continuación.

Definición 3.13 *Una aplicación* $\Delta : Lit \to \mathcal{PPN}^+$ *se llama una* asignación de posiciones *si existe un término* t *tal que, para todo literal* L, $\Delta(L)$ *sea compatible con el par* $\langle L, t\rangle$.

A esta definición le corresponde el predicado `comprueba_delta/2` en el apéndice B.

Nótese que el término t tal que para todo literal L, $\Delta(L)$ sea compatible con el par $\langle L, t\rangle$ no tiene por qué ser único. Basta considerar la siguiente asignación, que llamaremos *asignación vacía*

$$\begin{array}{rccl} \Delta: & Lit & \longrightarrow & \mathcal{PPN}^+ \\ & L & \mapsto & \Delta(L) = \emptyset \end{array}$$

o la asignación siguiente que llamaremos *asignación identidad*

$$\begin{array}{rccl} \Delta: & Lit & \longrightarrow & \mathcal{PPN}^+ \\ & L & \mapsto & \Delta(L) = \{\emptyset\} \end{array}$$

o cualquier asignación $\Delta : Lit \to \{\emptyset, \{\emptyset\}\} \subseteq \mathcal{PPN}^+$. En estos casos, dado un literal L, $\Delta(L)$ es compatible con $\langle L, t\rangle$ *para todo término* t. Pero salvo estos casos, la idea que formaliza la definición de asignación es que si Δ determina en dos literales L_1 y L_2 los términos t_1 y t_2 respectivamente, esto es, existen posiciones u_1 y u_2 tales que $L_1/u_1 = t_1$ y $L_2/u_2 = t_2$ y existen $P_1 \in \Delta(L_1)$ y $P_2 \in \Delta(L_2)$ y con $u_1 \in P_1$ y $u_2 \in P_2$, entonces t_1 y t_2 son el mismo término.

La siguiente definición, básica en nuestra formalización, relaciona inserciones y asignaciones. Dado $L \in Lit$ y una asignación Δ, para cada $P \in \Delta(L)$ generamos un literal insertando un término en cada posición de P. Formalmente

Definición 3.14 *Sea* $\Delta : Lit \to \mathcal{PPN}^+$ *una asignación de posiciones y sea* t *un término. Para cada literal* L *se define el siguiente conjunto*

$$L\{\Delta(L)/t\} = \{L\{P/t\} \mid P \in \Delta(L)\}$$

A esta definición le corresponde el predicado `operador_literal/4` en el apéndice B.

Por ejemplo, si $L = p(f(x), a, f(x))$, z es una variable, P_1^* lo tomamos del último ejemplo y Δ es una asignación tal que $\Delta(L) = P_1^*$

$$\begin{aligned} L\{\Delta(L)/z\} &= \{L\{P/z\} \mid P \in \Delta(L)\} \\ &= \{L\{P/z\} \mid P \in P_1^*\} \\ &= \{L\{P_1/z\}, L\{P_2/z\}, L\{P_3/z\} \\ &= \{p(z, a, z), p(z, a, f(x)), p(f(x), a, f(x))\} \end{aligned}$$

Nótese que

(a) Si $\Delta(L) = \emptyset$, entonces $L\{\Delta(L)/t\} = \emptyset$, para todo término t y para todo literal L.

(b) Si $\Delta(L) = P^* = \{P_1, \ldots, P_n\}$, entonces $L\{\Delta(L)/t\} = \{L_1, \ldots, L_n\}$, donde $L_i = L\{P_i/t\}$, i.e., L_i es el literal obtenido a partir del literal L insertando t en las posiciones de P_i.

(c) Si $\Delta(L) = P^* = \{\emptyset\}$, entonces $L\{\Delta(L)/t\} = \{L\{\emptyset/z\}\} = \{L\}$.

A continuación damos nuestra definición de operador

Definición 3.15 *Sea Δ una asignación y t un término. La aplicación definida por*

$$\begin{aligned} \{\Delta/t\} : \ \mathbb{C} &\longrightarrow \mathbb{C} \\ C &\mapsto C\{\Delta/t\} = \bigcup_{L \in C} L\{\Delta(L)/t\} \end{aligned}$$

se denomina operador clausal *asociado a Δ y t. Denotaremos por $\mathbb{OP}$ al conjunto de todos los operadores clausales.*

A esta definición le corresponde el predicado `operador_clausal/3` en el apéndice B.

Nótese que $C\{\Delta/t\}$ es la cláusula obtenida insertando el término t en cada literal L de C según determina la asignación $\Delta(L)$. Si C es la cláusula vacía entonces $C\{\Delta/t\} = \emptyset$ para todo término t y toda asignación Δ. Volviendo al ejemplo anterior, si $C = \{L_1, L_2, L_3\}$, con $L_1 = p(f(x))$, $L_2 = q(b, f(x))$, $L_3 = r(f(x), h(f(x)))$, y

$$\begin{aligned} \Delta(L_1) &= P_1^* = \{\{1\}\} \\ \Delta(L_2) &= P_2^* = \{\{2\}, \emptyset\} \\ \Delta(L_3) &= P_3^* = \{\{1, 2{\cdot}1\}, \{1\}\} \end{aligned}$$

y si tomamos z como el término que queremos insertar, entonces

$$\begin{aligned} C\{\Delta/z\} &= \bigcup_{L\in C} L\{\Delta(L)/z\} \\ &= L_1\{\Delta(L_1)/z\} \cup L_2\{\Delta(L_2)/z\} \cup L_3\{\Delta(L_3)/z\} \\ &= \{p(z)\} \cup \{q(b,z), q(b,f(x))\} \cup \{r(z,h(z)), r(z,h(f(x)))\} \\ &= \{p(z), q(b,z), q(b,f(x)), r(z,h(z)), r(z,h(f(x)))\} \end{aligned}$$

En el ejemplo 3.3 veíamos que la cláusula $C_2 = \{q(x,y), \neg p(x), \neg r(y)\}$ que obteníamos a partir de las cláusulas $C_1 = \{p(h(a))\}$ y $C = \{q(h(a),y), \neg r(y)\}$ por absorción, también podía obtenerse por aplicación de un operador de refinamiento a la cláusula

$$C \cup \{\neg p(h(a))\} = \{q(h(a),y), \neg r(y), \neg p(h(a))\}$$

De hecho, si consideramos la asignación

$$\begin{array}{rccl} \Delta: & Lit & \longrightarrow & \mathcal{PPN}^+ \\ & L & \mapsto & \Delta(L) = \left\{ \begin{array}{ll} \{\{1\}\} & \text{si } L \in \{q(h(a),y), \neg p(h(a))\} \\ \{\emptyset\} & e.o.c. \end{array} \right. \end{array}$$

por definición del operador se tiene que $(C \cup \{\neg L\})\{\Delta/x\} = C_2$.

En el ejemplo 3.4, la eliminación de un literal también puede verse como el resultado de aplicar un operador clausal. De esta manera, si

$$C = \{p(X), \neg q(X, f(a)), \neg r(X,Z), \neg menor\_que(Z,5), \neg menor\_que(Z,10)\}$$

y consideramos un término cualquiera t y la asignación

$$\begin{array}{rccl} \Delta: & Lit & \longrightarrow & \mathcal{PPN}^+ \\ & L & \mapsto & \Delta(L) = \left\{ \begin{array}{ll} \emptyset & \text{si } L = \neg menor\_que(Z,10) \\ \{\emptyset\} & e.o.c. \end{array} \right. \end{array}$$

se tiene que

$$C\{\Delta/t\} = \{p(X), \neg q(X, f(a)), \neg r(X,Z), \neg menor\_que(Z,5)\}$$

Por lo tanto, podemos simular la absorción y la eliminación de literales con operadores clausales. Veamos a continuación unas propiedades inmediatas de los

operadores que serán útiles más adelante.

Lema 3.16 *Sean C_1 y C_2 dos cláusulas y $\{\Delta/t\}$ un operador clausal. Se verifican las siguientes propiedades*

1. *Si $C_1 \subseteq C_2$ entonces se tiene que $C_1\{\Delta/t\} \subseteq C_2\{\Delta/t\}$*
2. $C_1\{\Delta/t\} \cup C_2\{\Delta/t\} = (C_1 \cup C_2)\{\Delta/t\}$
3. $(C_1 \cap C_2)\{\Delta/t\} \subseteq C_1\{\Delta/t\} \cap C_2\{\Delta/t\}$
4. *Si C_1 es una cláusula sin literales negativos (resp. positivos), entonces la cláusula $C_1\{\Delta/t\}$ tampoco tiene literales negativos (resp. positivos).*

Demostración:
Trivial. ⊣

Nota 3.17 En el apartado (3) del lema 3.16, el otro sentido la contención no es cierta. Por ejemplo, sean $C_1 = \{p(a,x)\}$ y $C_2 = \{p(x,a)\}$ con la asignación

$$\begin{array}{rccl} \Delta: & Lit & \longrightarrow & \mathcal{PPN}^+ \\ & L & \mapsto & \Delta(L) = \left\{ \begin{array}{ll} \{\{1\}\} & \text{si } L = p(a,x) \\ \{\{2\}\} & \text{si } L = p(x,a) \\ \emptyset & e.o.c. \end{array} \right. \end{array}$$

$$(C_1 \cap C_2)\{\Delta/x\} = \emptyset\{\Delta/x\} = \emptyset$$
$$C_1\{\Delta/x\} \cap C_2\{\Delta/x\} = \{p(x,x)\} \cap \{p(x,x)\} = \{p(x,x)\}$$

3.7 Extensiones

Según la nota 3.17, no podemos *especializar* la cláusula C_1 para pasar a C_2 si $C_1 \subseteq C_2$ y existe $L \in C_2$ tal que $par(L) \notin SP(C_1)$.

Esta limitación sólo afecta en el caso que queramos *especializar* la cláusula y no para *generalizar* el orden de subsunción o de generalización. De todas formas, podemos realizar una extensión de los operadores definidos considerando la posibilidad de insertar en la cláusula un literal con el símbolo de predicado y signo deseado. Para ello consideraremos un símbolo extralógico Ξ y extenderemos el conjunto SP a $SP^{\Xi} = SP \cup \{\Xi\}$ y en la definición del operador tendremos en cuenta la elección de un elemento $\alpha \in SP \cup \{\Xi\}$. Si $\alpha = \Xi$ entonces el operador

se define como hasta ahora. Si $\alpha \in SP$ entonces a la cláusula resultante de la acción del operador le añadimos un literal con el símbolo de predicado y signo de α. Lo vemos con más detalle.

Definición 3.18 *Sea $\{\Delta/t\}$ un operador clausal, sea SP el conjunto de pares $\langle p, s\rangle$ donde p es un símbolo de predicado del lenguaje y $x \in \{+,-\}$. Sea Ξ un símbolo que no pertenezca al lenguaje y sea $SP^{\Xi} = SP \cup \{\Xi\}$. Sea $\alpha \in SP^{\Xi}$. Se define*

$$\begin{array}{rccl} \{\Delta/t+\alpha\}: & \mathbb{C} & \longrightarrow & \mathbb{C} \\ & C & \mapsto & C\{\Delta/t+\alpha\} = \begin{cases} C\{\Delta/t\} & si\ \alpha = \Xi \\ C\{\Delta/t\} \cup \{L_\alpha^t\} & si\ \alpha \neq \Xi \end{cases} \end{array}$$

donde L_α^t es un literal cuyo símbolo de predicado y signo son los de α y todos sus argumentos son iguales a t. Llamaremos a $\{\Delta/t+\alpha\}$ el operador clausal extendido asociado a la terna $\langle \Delta, \alpha, t\rangle$.

A esta definición le corresponde el predicado `operador_clausal_extendido/4` en el apéndice B.

3.8 Resultado final

Terminamos el capítulo con un resultado de carácter técnico. Probaremos que la composición de operadores clausales junto con esta extensión definida en 3.18 nos permiten alcanzar cualquier cláusula con independencia de cual sea la cláusula de partida. Dicho de otro modo, cualquier operador que se considere en aprendizaje automático que lleve la cláusula C en la cláusula D puede caracterizarse como composición de los operadores vistos en este capítulo. Veamos las definiciones de los conjuntos $\mathbb{OP}^*$ y $\mathbb{OP}_{\Xi}^*$.

Puesto que los operadores son funciones de $\mathbb{C}$ en $\mathbb{C}$, la composición de operadores no es más que composición de funciones en sentido matemático. Si $\{\Delta_1/t_1\}$ y $\{\Delta_2/t_2\}$ son operadores clausales, notaremos la composición de ambos como $\{\Delta_1/t_1\} \circ \{\Delta_2/t_2\}$ y para todo $C \in \mathbb{C}$ se tiene que $C(\{\Delta_1/t_1\} \circ \{\Delta_2/t_2\}) = (C\{\Delta_1 \mathbf{t}_1\})\{\Delta_2/t_2\}$ Además, si tenemos en cuenta que

$$\begin{array}{rl} & L\{P/t_1\}\{\Delta_2/t_2\} \\ = & L\{P/t_1\}\{\Delta_2(L\{P/t_1\})/t_2\} \\ = & \{L\{P/t_1\}\{P'/t_2\} \mid P' \in \Delta_2(L\{P/t_1\})\} \end{array}$$

podemos expresar la aplicación de la composición de dos operadores a una cláusula de la siguiente manera

$$\begin{aligned} & C(\{\Delta_1/t_1\} \circ \{\Delta_2/t_2\}) \\ = \; & (C\{\Delta_1/t_1\})\{\Delta_2/t_2\} \\ = \; & [\cup_{L \in C} L\{\Delta_1(L)/t_1\}]\{\Delta_2/t_2\} \\ = \; & [\cup_{L \in C}\{L\{P/t_1\} \mid P \in \Delta_1(L)\}]\{\Delta_2/t_2\} \\ = \; & \cup_{L \in C} \cup_{P \in \Delta(L)} \{L\{P/t_1\}\}\{\Delta_2/t_2\} \\ = \; & \cup_{L \in C} \cup_{P \in \Delta_1(L)} \cup_{P' \in \Delta_2(L\{P/t_1\})} \{L\{P/t_1\}\{P'/t_2\}\}\{\Delta_2/t_2\}\} \end{aligned}$$

Definición 3.19 *Se definen los siguientes conjuntos de operadores:*

- $OP^1 = OP$
- $OP^{n+1} = \{\varphi \mid (\exists \varphi_1 \in OP)(\exists \varphi_2 \in OP^n)[\varphi = \varphi_1 \circ \varphi_2]\}$
- $\mathbb{OP}^* = \bigcup_{n \geq 1} OP^n$

Definición 3.20 *Se definen los siguientes conjuntos de operadores:*

- $OP_\Xi^1 = OP_\Xi$
- $OP_\Xi^{n+1} = \{\varphi \mid (\exists \varphi_1 \in OP_\Xi)(\exists \varphi_2 \in OP_\Xi^n)[\varphi = \varphi_1 \circ \varphi_2]\}$
- $\mathbb{OP}_\Xi^* = \bigcup_{n \geq 1} OP_\Xi^n$

Lema 3.21 *Si $\phi \in \mathbb{OP}^*$ y $C_1, C_2 \in \mathbb{C}$, entonces $(C_1 \cup C_2)\phi = C_1\phi \cup C_2\phi$.*

Demostración:
Si $\phi \in \mathbb{OP}^*$, entonces existen $\{\Delta_1/t_1\}, \ldots, \{\Delta_n/t_n\} \in OP$, con $n \geq 1$, tales que $\phi = \{\Delta_1/t_1\} \circ \{\Delta_2/t_2\} \circ \ldots \{\Delta_n/t_n\}$. Probamos el resultado por inducción en n.

($n = 1$) Por el lema 3.16, apartado 2, el resultado se tiene directamente

$$(C_1 \cup C_2)\phi = (C_1 \cup C_2)\{\Delta_1/t_1\} = C_1\{\Delta_1/t_1\} \cup C_2\{\Delta_1/t_1\} = C_1\phi \cup C_2\phi$$

($n \to n+1$) Se aplica el lema 3.16.2 y la hipótesis de inducción

$$\begin{aligned}
& (C_1 \cup C_2)\phi \\
= \; & (C_1 \cup C_2))\{\Delta_1/t_1\}\{\Delta_2/t_2\}\ldots\{\Delta_n/t_n\}\{\Delta_{n+1}/t_{n+1}\} \\
\overset{h.i.}{=} \; & C_1\{\Delta_1/t_1\}\{\Delta_2/t_2\}\ldots\{\Delta_n/t_n\} \\
& \cup \\
& C_2\{\Delta_1/t_1\}\{\Delta_2/t_2\}\ldots\{\Delta_n/t_n\})\{\Delta_{n+1}/t_{n+1}\} \\
= \; & C_1\{\Delta_1/t_1\}\{\Delta_2/t_2\}\ldots\{\Delta_n/t_n\}\{\Delta_{n+1}/t_{n+1}\} \\
& \cup \\
& C_2\{\Delta_1/t_1\}\{\Delta_2/t_2\}\ldots\{\Delta_n/t_n\}\{\Delta_{n+1}/t_{n+1}\} \\
= \; & C_1\phi \cup C_2\phi
\end{aligned}$$

⊣

Teorema 3.22 *Sean C y D dos cláusulas. Entonces existe $\phi \in \mathbb{OP}^*_\Xi$ tal que $C\phi = D$.*

Demostración:
Desglosamos la demostración en varios pasos

Paso 1: Para toda cláusula C existe un operador $\phi \in \mathbb{OP}^*_\Xi$ tal que $C\phi = \emptyset$. Si $D = \emptyset$, ese tal ϕ es el operador buscado.

Paso 2: Para toda cláusula D existe un operador $\phi \in \mathbb{OP}^*_\Xi$ tal que si aplicamos a la cláusula vacía el operador ϕ obtenemos la cláusula D, i.e., $\emptyset\phi = D$. La construcción de ese operador la haremos a su vez en dos pasos

Paso 2.1: En primer lugar, si $D = \{L_1, \ldots, L_n\}$, encontraremos una sucesión de operadores clausales que nos llevarán la cláusula vacía sobre la cláusula $\{L^{z_1}_{\alpha_1}, \ldots, L^{z_n}_{\alpha_n}\}$, donde $z_1, \ldots, z_n$ son variables distintas que no ocurren en D y $L^{z_i}_{\alpha_i}$ es un literal con el mismo símbolo de predicado y signo que L_i y todos sus argumentos son iguales a z_i

Paso 2.2: En este paso partimos de la cláusula $\{L^{z_1}_{\alpha_1}, \ldots, L^{z_n}_{\alpha_n}\}$ y encontramos una sucesión de operadores que nos lleva a la cláusula $D = \{L_1, \ldots, L_n\}$.

De manera que si tenemos dos cláusulas C y D al aplicar a C los operadores obtenidos en cada uno de estos pasos, llegamos a D. Detallamos a continuación cada uno de los pasos.

Paso 1: Sea $\Delta_0 : Lit \to \mathcal{PPN}^+$ la asignación vacía, i.e., para todo literal L, $\Delta_0(L) = \emptyset$ y sea x_0 una variable cualquiera. Entonces $C\{\Delta_0/x_0\} = \emptyset$. Si $D = \emptyset$ entonces basta tomar $\phi = \{\Delta_0/x_0\} \in \mathbb{OP}^*_\Xi$.

Paso 2.1: Si $D \neq \emptyset$, entonces sea $D = \{L_1, \ldots, L_n\}$. Consideremos n variables distintas $z_1, z_2, \ldots, z_n$ que no ocurren en D y para cada $i \in \{1, \ldots, n\}$ sea $\alpha_i = par(L_i)$ y $L^{z_i}_{\alpha_i}$ el literal con el mismo símbolo de predicado y signo que L_i y todos sus argumentos son iguales a z_i. Consideremos las siguientes asignaciones:

- Si $n = 1$, Δ_1 es la asiganción vacía, esto es, para todo literal L, $\Delta_1(L) = \emptyset$.

- Si $i \in \{2, \ldots, n\}$, entonces

$$\begin{array}{rccl} \Delta_i : & Lit & \longrightarrow & \mathcal{PPN}^+ \\ & L & \mapsto & \Delta_i(L) = \left\{ \begin{array}{ll} \{\emptyset\} & \text{si } L \in \{L^{z_1}_{\alpha_1}, \ldots, L^{z_{i-1}}_{\alpha_{i-1}}\}, \\ \emptyset & e.o.c. \end{array} \right. \end{array}$$

Veamos en primer lugar que los operadores clausales extendidos asociados a las ternas $\langle \Delta_i, \alpha_i, z_i \rangle$ llevan la cláusula vacía en la cláusula $\{L^{z_1}_{\alpha_1}, \ldots, L^{z_n}_{\alpha_n}\}$, esto es, probaremos que

$$\emptyset\{\Delta_1/z_1 + \alpha_1\}\{\Delta_2/z_2 + \alpha_2\} \ldots \{\Delta_n/z_n + \alpha_n\} = \{L^{z_1}_{\alpha_1}, \ldots, L^{z_n}_{\alpha_n}\}$$

Veamos por inducción que para todo $j \in \{1, \ldots, n\}$ se tiene que

$$\emptyset\{\Delta_1/z_1 + \alpha_1\}\{\Delta_2/z_2 + \alpha_2\} \ldots \{\Delta_j/z_j + \alpha_j\} = \{L^{z_1}_{\alpha_1}, \ldots, L^{z_j}_{\alpha_j}\}$$

Para $j = 1$ se tiene por aplicación directa de la definición

$$\emptyset\{\Delta_1/z_1 + \alpha_1\} = \emptyset\{\Delta_1/z_1\} \cup \{L^{z_1}_{\alpha_1}\} = \emptyset \cup \{L^{z_1}_{\alpha_1}\} = \{L^{z_1}_{\alpha_1}\}$$

Para el paso de inducción, hay que tener en cuenta que si $k < j$, entonces $\{L^{z_k}_{\alpha_k}\}\{\Delta_j/z_j + \alpha_j\} = \{L^{z_k}_{\alpha_k}\}$. Una vez visto esto basta aplicar la hipótesis de

inducción junto con la definición.

$$\begin{aligned}
& \emptyset\{\Delta_1/z_1+\alpha_1\}\{\Delta_2/z_2+\alpha_2\}\ldots\{\Delta_j/z_j+\alpha_j\} \\
\overset{h.i.}{=} & \{L_{\alpha_1}^{z_1},\ldots,L_{\alpha_{j-1}}^{z_{j-1}}\}\{\Delta_j/z_j+\alpha_j\} \\
= & \{L_{\alpha_1}^{z_1},\ldots,L_{\alpha_{j-1}}^{z_{j-1}}\}\{\Delta_j/z_j+\alpha_j\}\cup\{L_{\alpha_j}^{z_j}\} \\
= & [\bigcup_{k=1}^{k=j-1}\{L_{\alpha_k}^{z_k}\}\{\Delta_j/z_j+\alpha_j\}]\cup\{L_{\alpha_j}^{z_j}\} \\
= & \{L_{\alpha_1}^{z_1},\ldots,L_{\alpha_{j-1}}^{z_{j-1}}\}\cup\{L_{\alpha_j}^{z_j}\} \\
= & \{L_{\alpha_1}^{z_1},\ldots,L_{\alpha_{j-1}}^{z_{j-1}},L_{\alpha_j}^{z_j}\}
\end{aligned}$$

Paso 2.2: Veamos cómo podemos transformar la cláusula $\{L_{\alpha_1}^{z_1},\ldots,L_{\alpha_n}^{z_n}\}$ en $\{L_1,\ldots,L_n\}$ con ayuda de los operadores clausales. Para cada $i\in\{1,\ldots,n\}$, sea k_i la aridad del símbolo de predicado de L_i. Sean $\langle t_1^i,\ldots,t_{k_i}^i\rangle$ los argumentos de L_i. Para todo $i\in\{1,\ldots,n\}$ y todo $j\in\{1,\ldots,k_i\}$ consideremos el literal

$$L(i,j)=L_i\{\{j\}/z_i\}\{\{j+1\}/z_i\}\ldots\{\{k_i\}/z_i\}$$

esto es, $L(i,j)$ es un literal con el mismo símbolo de predicado y signo que L_i y cuyo argumento m–ésimo, con $m\in\{1,\ldots,j-1\}$ es igual al término t_i^m y el argumento m–ésimo, con $m\in\{j,\ldots,k_i\}$ es igual a z_i. Nótese que $L(i,1)$ es $L_{\alpha_i}^{z_i}$. Para todo $i\in\{1,\ldots,n\}$ y todo $j\in\{1,\ldots,k_i\}$ consideremos también la asignación

$$\begin{array}{rccl}
\Delta_i^j: & Lit & \longrightarrow & \mathcal{PPN}^+ \\
& L & \mapsto & \Delta_i^j(L)=\left\{\begin{array}{ll}\{\{j\}\} & \text{si } L=L(i,j)\\ \{\emptyset\} & e.o.c.\end{array}\right.
\end{array}$$

Por último, para todo $i\in\{1,\ldots,n\}$ y todo $j\in\{1,\ldots,k_i\}$ consideramos el operador $\{\Delta_i^j/t_i^j\}$. Veamos que se verifican las siguientes propiedades para estos operadores:

Sean $i,k\in\{1,\ldots,n\}$ con $i\neq k$. Entonces

(a) $\{L_{\alpha_i}^{z_i}\}\{\Delta_i^1/t_i^1\}\{\Delta_i^2/t_i^2\}\ldots\{\Delta_i^{k_i-1}/t_i^{k_i-1}\}\{\Delta_i^{k_i}/t_i^{k_i}\}=\{L_i\}$

(b) $(\forall j\in\{1,\ldots,k_i\})[\{L_{\alpha_k}^{z_k}\}\{\Delta_i^j/t_i^j\}=\{L_{\alpha_k}^{z_k}\}]$

El apartado (a) nos dice que mediante la aplicación de los k_i operadores correspondientes obtenemos el literal L_i a partir de $\{L_{\alpha_i}^{z_i}\}$. El apartado (b) nos dice que

estos operdores necesarios para la transformación de $\{L_{\alpha_i}^{z_i}\}$ en L_i deja el resto de los $\{L_{\alpha_k}^{z_k}\}$ invariantes.

Para la demostración del apartado (a) probemos por inducción que para todo $r \in \{1, \ldots, k_i\}$ se tiene que

$$\{L_{\alpha_i}^{z_i}\}\{\Delta_i^1/t_i^1\}\{\Delta_i^2/t_i^2\}\ldots\{\Delta_i^{r-1}/t_i^{r-1}\}\{\Delta_i^r/t_i^r\} = \{L(i, r+1)\}$$

Para $r = 1$ el resultado se tiene inmediatamente a partir de las definiciones

$$\{L_{\alpha_i}^{z_i}\}\{\Delta_i^1/t_i^1\} = \{L(i,1)\}\{\Delta_i^1/t_i^1\} = \{L(i,2)\}$$

Para el paso de inducción ($r-1 \rightarrow r$) sólo hay que tener en cuenta las definiciones y la hipótesis de inducción

$$\{L_{\alpha_i}^{z_i}\}\{\Delta_i^1/t_i^1\}\{\Delta_i^2/t_i^2\}\ldots\{\Delta_i^{r-1}/t_i^{r-1}\}\{\Delta_i^r/t_i^r\} \stackrel{h.i}{=} \{L(i,r)\}\{\Delta_i^r/t_i^r\} = \{L(i,r+1)\}$$

Por consiguiente se tiene que

$$\begin{aligned} & \{L_{\alpha_i}^{z_i}\}\{\Delta_i^1/t_i^1\}\{\Delta_i^2/t_i^2\}\ldots\{\Delta_i^{k_i-1}/t_i^{k_i-1}\}\{\Delta_i^{k_i}/t_i^{k_i}\} \\ = & \{L(i,1)\}\{\Delta_i^1/t_i^1\}\{\Delta_i^2/t_i^2\}\ldots\{\Delta_i^{k_i-1}/t_i^{k_i-1}\}\{\Delta_i^{k_i}/t_i^{k_i}\} \\ = & \{L(i,k_i+1)\} \\ = & \{L_i\} \end{aligned}$$

Para probar (b) sólo hay que tener en cuenta que si $i \neq k$ entonces $z_i \neq z_k$, luego para todo $j \in \{1, \ldots, k_i\}$, $L_{\alpha_k}^{z_k} \neq L(i,j)$ y por tanto, por definición del operador, $\{L_{\alpha_k}^{z_k}\}\{\Delta_i^j/t_i^j\} = \{L_{\alpha_k}^{z_k}\}$.

Por tanto hemos visto que:

- Mediante la aplicación de los k_i operadores correspondientes obtenemos el literal L_i a partir de $\{L_{\alpha_i}^{z_i}\}$.

- Los operdores necesarios para la transformación de $\{L_{\alpha_i}^{z_i}\}$ en L_i deja el resto de los $\{L_{\alpha_k}^{z_k}\}$ invariantes.

Por tanto para terminar este paso 2.2 sólo queda comprobar que la aplicación sucesiva de estos operadores a la cláusula $\{L_{\alpha_1}^{z_1}, \ldots, L_{\alpha_n}^{z_n}\}$ nos devuelve $\{L_1, \ldots, L_n\}$. Haremos esta comprobación en dos pasos.

Para simplificar la notación, para todo $i \in \{1, \ldots, n\}$, llamaremos ϕ_i a la

composición de operadores

$$\{\Delta_i^1/t_i^1\}\{\Delta_i^2/t_i^2\}\dots\{\Delta_i^{k_i-1}/t_i^{k_i-1}\}\{\Delta_i^{k_i}/t_i^{k_i}\}$$

con lo que el apartado (a) del aserto 2 se resume en $\{L_{\alpha_i}^{z_i}\}\phi_i = \{L_i\}$ para todo $i \in \{1,\dots,n\}$. Además, del apartado (b) se obtiene de manera inmediata que si $i \neq k$, $\{L_{\alpha_k}^{z_k}\}\phi_1 = \{L_{\alpha_k}^{z_k}\}$.

Veamos ahora el primer paso de esa comprobación. Veamos en primer lugar que para todo $i \in \{1,\dots,n\}$ se verifica

$$[\bigcup_{k=1}^{i-1}\{L_k\}] \cup [\bigcup_{k=i}^{n}\{L_{\alpha_k}^{z_k}\}]\phi_i = [\bigcup_{k=1}^{i}\{L_k\}] \cup [\bigcup_{k=i+1}^{n}\{L_{\alpha_k}^{z_k}\}]$$

Sólo hay que tener en cuenta el resultado que enunciamos en el lema 3.21

$$\begin{array}{cl} & [\bigcup_{k=1}^{i-1}\{L_k\}] \cup [\bigcup_{k=i}^{n}\{L_{\alpha_k}^{z_k}\}]\phi_i \\ \overset{Lema3.21}{=} & [\bigcup_{k=1}^{i-1}\{L_k\}\phi_i] \cup \{\{L_{\alpha_i}^{z_i}\}\phi_i\} \cup [\bigcup_{k=i}^{n}\{L_{\alpha_k}^{z_k}\}\phi_i] \\ = & [\bigcup_{k=1}^{i-1}\{L_k\}] \cup \{L_i\} \cup [\bigcup_{k=i}^{n}\{L_{\alpha_k}^{z_k}\}] \\ = & [\bigcup_{k=1}^{i}\{L_k\}] \cup [\bigcup_{k=i+1}^{n}\{L_{\alpha_k}^{z_k}\}] \end{array}$$

Veamos por último que

$$\{L_{\alpha_1}^{z_1},\dots\{L_{\alpha_n}^{z_n}\}\phi_1\phi_2\dots\phi_n = \{L_1,\dots,L_n\}$$

Se tiene de manera inmediata teniendo en cuenta la comprobación anterior

$$\begin{array}{cl} & \{L_{\alpha_1}^{z_1},\dots\{L_{\alpha_n}^{z_n}\}\phi_1\phi_2\dots\phi_n \\ = & (\{L_1\} \cup \{L_{\alpha_2}^{z_2},\dots\{L_{\alpha_n}^{z_n}\}\phi_2\dots\phi_n \\ = & \dots \\ = & (\{L_1,\dots,L_{n-1}\} \cup \{L_{\alpha_n}^{z_n}\})\phi_n \\ = & \{L_1,\dots,L_n\} \end{array}$$

Recapitulando, tenemos los siguientes operadores en $\mathbb{OP}_\Xi^*$:

$$\{\Delta_0/z_0+\alpha_0\},\{\Delta_1/z_1+\alpha_1\}\dots\{\Delta_n/z_n+\alpha_n\},\phi_1,\dots,\phi_n$$

Su composición también está en $\mathbb{OP}_\Xi^*$ y además hemos visto que al aplicarlos sucesivamente a C obtenemos D. Esto concluye la demostración del teorema $\dashv$

Ilustramos el teorema con el siguiente ejemplo (corresponde con el `Ejemplo 3.23` en el apéndice B):

Ejemplo 3.23 Consideremos las cláusulas

$$C = \{q(x_1, x_2), \neg r(a), p(x_1, b)\}$$
$$D = \{s(a, f(b)), r(x_3)\}$$

Veamos que podemos pasar de C a D mediante operadores clausales extendidos. Seguimos los pasos del teorema.

Paso 1: Consideremos la asignación vacía

$$\begin{array}{rccl} \Delta: & Lit & \longrightarrow & \mathcal{PP}\mathbb{N}^+ \\ & L & \mapsto & \Delta(L) = \emptyset \end{array}$$

y una variable cualquiera, por ejemplo, x_1. Entonces el operador clausal asociado lleva C en la cláusula vacía.

$$C\{\Delta/x_1\} = \emptyset$$

Paso 2a: Puesto que la cláusula $D = \{s(a, f(b)), r(x3)\}$ tiene dos literales, tomamos dos variables que no ocurran en D, por ejemplo z_1 y z_2 y consideramos los literales $s(z_1, z_1)$ y $r(z_2)$, esto es, para cada literal de la cláusula D tomamos un nuevo literal con el mismo símbolo de predicado y signo, pero con todos sus argumentos iguales a la variable nueva asociada. En este paso usamos los operadores clausales extendidos para llevar la cláusula vacía en la cláusula $\{s(z_1, z_1), r(z_2)\}$. Consideremos las asignaciones

$$\begin{array}{rccl} \Delta_1: & Lit & \longrightarrow & \mathcal{PP}\mathbb{N}^+ \\ & L & \mapsto & \Delta_1(L) = \emptyset \end{array}$$

Δ_1 es la asignación vacía y Δ_2 la asignación

$$\Delta_2(L) = \begin{cases} \{\emptyset\} & \text{si } L = s(z_1, z_1) \\ \emptyset & \text{en otro caso} \end{cases}$$

Por tanto, aplicando a la cláusula vacía el operador extendido $\{\Delta_1/z_1 + \langle s/2, \oplus\rangle\}$

tenemos

$$\begin{aligned} & \emptyset\{\Delta_1/z_1 + \langle s/2, \oplus\rangle\} \\ = \ & \emptyset\{\Delta_1/z_1\} \cup \{s(z_1, z_1)\} \\ = \ & \emptyset \cup \{s(z_1, z_1)\} \\ = \ & \{s(z_1, z_1)\} \end{aligned}$$

y si ahora aplicamos $\{\Delta_2/z_2 + \langle r/1, \oplus\rangle\}$ tenemos

$$\begin{aligned} & \emptyset\{\Delta_1/z_1 + \langle s/2, \oplus\rangle\}\{\Delta_2/z_2 + \langle r/1, \oplus\rangle\} \\ = \ & \{s(z_1, z_1)\}\{\Delta_2/z_2 + \langle r/1, \oplus\rangle\} \\ = \ & \{s(z_1, z_1)\}\{\Delta_2/z_2\} \cup \{r(z_2)\} \\ = \ & \{s(z_1, z_1)\} \cup \{r(z_2)\} \\ = \ & \{s(z_1, z_1), r(z_2)\} \end{aligned}$$

Paso 2b: Veamos ahora como transformamos la cláusula $\{s(z_1, z_1), r(z_2)\}$ en $D = \{s(a, f(b)), r(x_3)\}$ mediante operadores clausales. Siguiendo la notación del teorema, si $L_1 = s(a, f(b))$, $L_2 = r(x_3)$, $L_{\alpha_1}^{z_1} = s(z_1, z_1)$ y $L_{\alpha_2}^{z_2} = r(z_1)$ debemos considerar los siguientes literales:

$$\begin{aligned} L(1,1) &= s(z_1, z_1) \qquad & L(2,1) &= r(z_2) \\ L(1,2) &= s(a, z_1) \end{aligned}$$

Consideramos también las asignaciones siguientes

$$\Delta_1^1(L) = \begin{cases} \{\{1\}\} & \text{si } L = L(1,1) \\ \{\emptyset\} & \textit{e.o.c.} \end{cases} \qquad \Delta_2^1(L) = \begin{cases} \{\{1\}\} & \text{si } L = L(2,1) \\ \{\emptyset\} & \textit{e.o.c.} \end{cases}$$

$$\Delta_1^2(L) = \begin{cases} \{\{2\}\} & \text{si } L = L(1,2) \\ \{\emptyset\} & \textit{e.o.c.} \end{cases}$$

Veamos que en este caso

$$\begin{aligned} & \{s(z_1, z_1)\}\{\Delta_1^1/a\}\{\Delta_1^2/f(b)\} \\ = \ & \{s(a, z_1)\}\{\Delta_1^2/f(b)\} \\ = \ & \{s(a, f(b))\} \end{aligned}$$

y por otra parte

$$\begin{aligned} & \{r(z_2)\}\{\Delta_2^1/x_3\} \\ = \ & \{r(x_3)\} \end{aligned}$$

Por consiguiente, si aplicamos estos operadores a la cláusula $\{s(z_1, z_1), r(z_2)\}$ tenemos

$$\begin{aligned}
& \{s(z_1, z_1), r(z_2)\}\{\Delta_1^1/a\}\{\Delta_1^2/f(b)\}\{\Delta_2^1/x_3\} \\
= \; & \{s(z_1, z_1)\}\{\Delta_1^1/a\}\{\Delta_1^2/f(b)\}\{\Delta_2^1/x_3\} \cup \{r(z_2)\}\{\Delta_1^1/a\}\{\Delta_1^2/f(b)\}\{\Delta_2^1/x_3\} \\
= \; & \{s(a, f(b))\}\{\Delta_2^1/x_3\} \cup \{r(z_2)\}\{\Delta_2^1/x_3\} \\
= \; & \{s(a, f(b))\} \cup \{r(x_3)\} \\
= \; & \{s(a, f(b)), r(x_3)\} \\
= \; & D
\end{aligned}$$

Por tanto, tenemos que

$$C\{\Delta/x_1\}\{\Delta_1/z_1 + \langle s/2, \oplus\rangle\}\{\Delta_2/z_2 + \langle r/1, \oplus\rangle\}\{\Delta_1^1/a\}\{\Delta_1^2/f(b)\}\{\Delta_2^1/x_3\} = D$$

Nota 3.24 A partir del teorema 3.22 podemos encontrar una cota superior para el mínimo número de operadores clausales extendidos necesarios para llevar una cláusula C en otra D cuando C y D son dos cláusulas cualesquiera. Si seguimos el algoritmo de construcción de los operadores determinado en la demostración del teorema. Necesitamos un operador para llevar la cláusula C en la cláusula vacía. A continuación tantos operadores extendidos como literales ocurren en D y por último un operador por cada argumento de cada literal de D. Escrito como fórmula, la cota se expresa como

$$1 + |D| + \sum\{k \mid \pm p(t_1, \ldots, t_k) \in D\}$$

Capítulo 4

Operadores de aprendizaje para la subsunción

En el capítulo 3 hemos presentado la definición de una familia de operadores que llamamos *operadores clausales*, los cuales permiten obtener una cláusula a partir de otra mediante la inserción de un término. Esta definición, junto con su extensión, nos permite obtener cualquier cláusula a partir de otra (Teorema 3.22).

En este capítulo y los siguientes centraremos nuestro estudio en un subconjunto de estos operadores clausales, los operadores clausales ascendentes, esto es, aquellos operadores en los que la cláusula resultante de aplicar un operador es más general, respecto de algún orden, que la cláusula original. Los órdenes entre cláusulas estudiados en esta memoria son el orden de subsunción y de consecuencia.

En un sentido abstracto, para poder hablar de *generalización* sólo necesitamos una relación binaria definida en un conjunto. Podemos formalizar la idea de generalización de las siguiente manera:

Definición 4.1 *Dado un conjunto S y una relación binaria $R \subseteq S \times S$, si $a, b \in S$ diremos que a es* más general *que b si $\langle a, b \rangle \in R$*

Una vez definida esa relación binaria en un conjunto S, tiene sentido plantearse si una función $f : S \to S$ es una función de *generalización* en S. En este caso las definiciones surgen de manera natural.

Definición 4.2 *Dado un conjunto S y una relación binaria $R \subseteq S \times S$*

- *Dada una aplicación* $f : S \to S$ *y un elemento* $a \in S$, *diremos que* f *generaliza el elemento* a *en la relación* R *si* $\langle f(a), a\rangle \in R$.

- *Dada una aplicación* $f : S \to S$ *diremos que* f *es un* operador de generalización *para la relación* R *si* $(\forall a \in S)[\langle f(a), a\rangle \in R]$

En este capítulo estudiamos la existencia de operadores de generalización para la relación de subsunción sobre cláusulas. En primer lugar veremos que si C y D son cláusulas y $C \succeq D$ entonces existe un operador $\phi \in \mathbb{OP}^*$ tal que $D\phi = C$ sin necesidad de recurrir a los operadores extendidos.

En la segunda parte del capítulo caracterizaremos los operadores de generalización para la relación de subsunción entre cláusulas, que llamaremos *operadores de aprendizaje para la subsunción.*

4.1 Subsunción

La relación de subsunción que vemos a continuación, básica en esta memoria, fue definida por Plotkin en 1970 [Plo70].

Sean C y D dos cláusulas. Decimos que C subsume a D, $C \succeq D$, si existe una sustitución θ tal que $C\theta \subseteq D$. La cláusula vacía subsume cualquier cláusula.

Nótese que la sustitución θ no tiene porqué ser única. Basta considerar $C_1 = \{p(x)\}$ y $C_2 = \{p(a), p(b)\}$. Se tiene que $C_1 \succeq C_2$, pero en ningún caso podríamos discriminar entre $\theta = \{x/a\}$ y $\theta = \{x/b\}$. De ahí podemos obtener una idea intuitiva de la decidibilidad y complejidad computacional de la relación de subsunción. Dadas dos cláusulas C_1 y C_2 con $|C_1| = n$ y $|C_2| = m$ hay m^n posibles aplicaciones de C_1 en C_2. Un análisis de estas aplicaciones nos dirá si alguna de ellas corresponde o no a una sustitución. En caso afirmativo diríamos que C_1 subsume a C_2 y en caso negativo, que no se produce subsunción.

La subsunción entre cláusulas es un problema NP–completo [GJ79], p.264. Podemos encontrar una demostración de este hecho[1] en Kietz y Lübbe [KL94]. La relación de subsunción entre cláusulas da lugar de forma natural a una relación de equivalencia. Si $C \succeq D$ y $D \succeq C$ diremos que C y D son equivalentes bajo subsunción y escribiremos $C \sim D$. Puesto que si dos cláusulas son equivalentes bajo subsunción entonces son lógicamente equivalentes, los sistemas PLI

[1] Al parecer, la prueba original se debe a Baxter, en un manuscrito no publicado de 1977 [Bax77].

deben considerar las ventajas e inconvenientes de considerar únicamente un representante de cada clase de equivalencia. Este tema fue tratado en [Mah86, Sha81, PvdL92, NCdW97] y [vdLNC98] entre otros. Volveremos a este punto en la siguiente sección.

El problema de elegir un representante de cada clase de equivalencia lo solucionó Plotkin [Plo70] demostrando que en cada clase de equivalencia existe lo que él denominó *cláusula reducida*, única salvo renombramiento de variables. Una cláusula C se dice reducida si no existe un subconjunto propio D de C tal que $D \sim C$. Notaremos por $\mathbb{C}_R$ el conjunto de cláusulas reducidas de $\mathbb{C}$.

El conjunto de las cláusulas reducidas forma un retículo respecto la relación de subsunción (o equivalentemente, podemos dotar de estructura de retículo al espacio cociente que resulta en el conjunto de cláusulas al establecer la relación de equivalencia bajo subsunción), esto es, dadas dos cláusula reducidas C_1 y C_2, existe un único ínfimo $inf(C_1, C_2)$ y un único supremo $sup(C_1, C_2)$, que recibe el nombre de *menor generalización general de* C_1 *y* C_2, y que denotaremos como $mgg(C_1, C_2)$.

4.2 Operadores y sustituciones unitarias

La idea básica en los operadores clausales ascendentes relativos a la subsunción (operadores de generalización) es la descomposición de este orden en las dos relaciones básicas de las que se compone. Si $C_1 \succeq C_2$ entonces existe una sustitución θ tal que $C_1\theta \subseteq C_2$. Podemos pensar que cada sustitución θ determina en $\mathbb{C} \times \mathbb{C}$ la relación $\{\langle C, C\theta\rangle \mid C \in \mathbb{C}\}$, que junto con la relación de subconjunto $\{\langle C, D\rangle \mid C \subseteq D\}$ da lugar a la relación de subsunción entre cláusulas. Por el teorema 3.22 sabemos que si C y D son dos cláusulas cualesquiera entonces existe un operador clausal extendido ϕ tal que $D\phi = C$. En particular esto ocurre cuando C subsume a D. En la primera parte del capítulo veremos que si $C \succeq D$, entonces no necesitamos usar la extensión.

Empezamos viendo un par de lemas técnicos donde estudiamos la relación del orden de subsunción entre cláusulas con nuestros operadores. En el primero de ellos vemos que dada una sustitución elemental $\theta = \{x/t\}$ y las cláusulas C y $C\theta$ siempre podemos encontrar un operador clausal que *invierta* la sustitución, esto es, que aplicado a $C\theta$ nos devuelva C.

En el segundo lema estudiamos la relación de contención $\subseteq$ y consideramos que en esta relación C_1 es *más general que* C_2 si $C_1 \subseteq C_2$, esto es, en el mismo sentido

que la relación de subsunción cuando tomamos la sustitución vacía. Demostramos que si $C_1 \subseteq C_2$ entonces existe un operador clausal que aplicado a C_2 nos devuelve C_1.

Lema 4.3 *Sean C_1 y C_2 dos cláusulas y $\theta = \{x/t\}$ una sustitución elemental tal que $C_1\theta = C_2$. Entonces existe una asignación de posiciones Δ tal que $C_2\{\Delta/x\} = C_1$, donde la variable x de la sustitución y del operador son la misma.*

A este lema le corresponde el predicado `operador_asociado_sustitución/4` en el apéndice B.

Demostración:

En la demostración sólo hay que tener en cuenta que si L y L' son dos literales y $\theta = \{x/t\}$ es una sustitución elemental tal que $L'\theta = L$, entonces $L\{Pos(L', x)/x\} = L'$. Veamos ahora la demostración.

Para todo $L \in C_2$, sea $\theta_{C_1}^{-1}(L) = \{L' \in C_1 \mid L'\theta = L\}$. Consideremos la asignación de posiciones

$$\begin{array}{rccc} \Delta: & Lit & \longrightarrow & \mathcal{P}\mathcal{P}\mathbb{N}^+ \\ & L & \mapsto & \Delta(L) \end{array}$$

$$\Delta(L) = \begin{cases} \{Pos(L', x) \mid L' \in \theta_{C_1}^{-1}(L)\} & \text{si } L \in C_2 \\ \{\emptyset\} & \text{si } L \notin C_2 \end{cases}$$

Se tiene que si $L'\theta = L$ con $\theta = \{x/t\}$ entonces $Pos(L', x) \subseteq Pos(L, t)$. De ahí que Δ sea una asignación de posiciones, puesto que, para todo L, $\Delta(L)$ es compatible con el par $\langle L, t\rangle$. Veamos que $C_2\{\Delta/x\} = C_1$

$$\begin{aligned} C_2\{\Delta/x\} &= \bigcup_{L \in C_2} L\{\Delta(L)/x\} \\ &= \bigcup_{L \in C_2} L\{\{Pos(L', x) \mid L' \in \theta_{C_1}^{-1}(L)\}/x\} \\ &= \bigcup_{L \in C_2} \{L\{Pos(L', x)/x\} \mid L' \in \theta_{C_1}^{-1}(L)\} \\ &= \bigcup_{L \in C_2} \theta_{C_1}^{-1}(L) \\ &= C_1 \end{aligned}$$

⊣

Veamos ahora que si $C_1 \subseteq C_2$, esto es, C_1 es más general que C_2 en el orden $\subseteq$,

entonces podemos generalizar C_2 para obtener C_1 mediante un operador clausal.

Lema 4.4 *Sean C_1 y C_2 dos cláusulas tales que $C_1 \subseteq C_2$. Entonces existe una variable z y una asignación de posiciones Δ tal que $C_2\{\Delta/z\} = C_1$.*

A este lema le corresponde el predicado `operador_asociado_subconjunto/3` en el apéndice B.

Demostración:
Basta tomar una variable cualquiera z y la asignación

$$\begin{array}{rccl} \Delta: & Lit & \longrightarrow & \mathcal{PPN}^+ \\ & L & \mapsto & \Delta(L) = \begin{cases} \emptyset & \text{si } L \in C_2 - C_1 \\ \{\emptyset\} & \text{en otro caso} \end{cases} \end{array}$$

Trivialmente Δ es una asignación de posiciones. Además

$$\begin{aligned} C_2\{\Delta/z\} &= \bigcup_{L \in C_2} L\{\Delta(L)/z\} \\ &= (\bigcup_{L \in C_1} L\{\Delta(L)/z\}) \cup (\bigcup_{L \in C_2 - C_1} L\{\Delta(L)/z\}) \\ &= (\bigcup_{L \in C_1} L\{\{\emptyset\}/z\}) \cup (\bigcup_{L \in C_2 - C_1} L\{\emptyset/z\}) \\ &= (\bigcup_{L \in C_1} \{L\}) \cup \emptyset \\ &= C_1 \end{aligned}$$

⊣

Estos dos lemas nos dan los operadores elementales que van a permitir generalizar D para obtener C si $C \succeq D$. En la siguiente sección vamos a extender la idea de generalización por inversión de sustitución (lema 4.3) al caso en que la sustitución no sea elemental

4.3 Operadores clausales y subsunción entre cláusulas

La idea que desarrollamos a continuación se basa en la descomposición de la relación de subsunción en las dos relaciones que citamos en la sección 4.2, esto es, por definición, si $C_1 \succeq C_2$ entonces existe una sustitución θ tal que $C_1\theta \subseteq C_2$.

Visto de otra forma, $C_1 \succeq C_2$ si y sólo si existen una sustitución θ y una cláusula D tales que si $\mathcal{R}_\theta$, $\mathcal{R}_\subseteq$ son las relaciones binarias en $\mathbb{C}$

- $\mathcal{R}_\theta = \{\langle D_1, D_2\rangle \in \mathbb{C} \times \mathbb{C} \mid D_1\theta = D_2\}$
- $\mathcal{R}_\subseteq = \{\langle D_1, D_2\rangle \in \mathbb{C} \times \mathbb{C} \mid D_1 \subseteq D_2\}$

entonces $\langle C_1, D\rangle \in \mathcal{R}_\theta$, $\langle D, C_2\rangle \in \mathcal{R}_\subseteq$.

El lema 4.4 nos dice que si $\langle C_1, C_2\rangle \in \mathcal{R}_\subseteq$ entonces existe un operador clausal ϕ tal que $C_2\phi = C_1$. El lema 4.3 nos dice que si $\langle C_1, C_2\rangle \in \mathcal{R}_\theta$ entonces existe un operador clausal ϕ tal que $C_2\phi = C_1$, pero sólo en el caso en que θ sea elemental. Para extender este resultado a cualquier sustitución θ usaremos un resultado muy conocido, cuyo enunciado y demostración damos a continuación.

Proposición 4.5 *Sea $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ una sustitución, $x_i \neq t_i$. Entonces existen $\sigma_1, \ldots \sigma_m$ sustituciones elementales, $\sigma_i = \{z_i/s_i\}$ tales que:*

1. $(\forall i \in \{1, \ldots, n\})[x_i\sigma_1\,\sigma_2 \ldots \sigma_m = t_i]$
2. *Para todo $j \in \{1, \ldots, m\}$, z_i no ocurre en s_i.*

A esta proposición le corresponde el predicado `descompone_sustitucion/2` en el apéndice B.

Demostración:
Sea $[Var(t_1) \cup Var(t_2) \cup \cdots \cup Var(t_n)] \cap Dom(\theta) = \{y_1, \ldots, y_k\}$ y sean $z_1, \ldots, z_k$ variables distintas que no ocurran en θ. Sea $\delta = \{y_1/z_1, \ldots, y_k/z_k\}$ y consideremos

$$\begin{array}{llll} \sigma_1 = \{x_1/t_1\delta\} & \ldots \quad \ldots & \ldots & \sigma_n = \{x_n/t_n\delta\} \\ \sigma_{n+1} = \{z_1/y_1\} & \ldots \quad \ldots & \sigma_{n+k} = \{z_k/y_k\} & \end{array}$$

Consideremos ahora las sustituciones elementales σ_j. Si $j \in \{1, \ldots, n\}$, sabemos que x_j no ocurre en $t_j\delta$ ya que ninguna variable de $t_j\delta$ pertenece al $Dom(\theta)$. Si $j \in \{n+1, \ldots, n+k\}$, no hay nada que probar, z_j e y_j son variables diferentes. Por otro lado, si $x_i \in Dom(\theta)$ entonces, teniendo en cuenta que $Var(t_i\delta) \cap Dom(\theta) = \emptyset$ se tiene

$$\begin{aligned} & x_i(\sigma_1 \ldots \sigma_n\,\sigma_{n+1} \ldots \sigma_{n+k}) \\ = \; & t_i\delta(\sigma_{i+1} \ldots \sigma_n\,\sigma_{n+1} \ldots \sigma_{n+k}) \\ = \; & t_i\delta(\sigma_{n+1} \ldots \sigma_{n+k}) \\ = \; & t_i \end{aligned}$$

⊣

A partir de la proposición 4.5 y de los lemas 4.3 y 4.4 se tiene el siguiente corolario.

Corolario 4.6 *Si* $C_1 \succeq C_2$ *entonces existen operadores clausales*

$$\{\Delta_1/t_1\}, \ldots, \{\Delta_k/t_k\}$$

tales que

$$C_1 = C_2\{\Delta_k/t_k\}\{\Delta_{k-1}/t_{k-1}\}\ldots\{\Delta_1/t_1\}$$

A este corolario le corresponde el predicado `genera_operadores_clausales/5` en el apéndice B.

Demostración:

Sea θ una sustitución tal que $C\theta \subseteq D$. Por la proposición 4.5 sabemos que existen $\sigma_1, \ldots, \sigma_n$ sustituciones elementales tales que $\theta = \sigma_1 \ldots \sigma_n$, con $\sigma_j = \{x_j/t_j\}$. En la construcción de esas sustituciones $\sigma_1, \ldots, \sigma_n$, según hemos visto en la demostración de la proposición 4.5 tomamos unas variables *nuevas* que no ocurren en la sustitución θ. Para asegurar que los operadores asociados a esas sustituciones elementales verifican la tesis del corolario basta exigir que esas variables *nuevas* no ocurran en C_1.

Sean $C_0 = C$, $C_j = C\sigma_1 \ldots \sigma_j = C_{j-1}\sigma_j$, para todo $j \in \{1, \ldots, n\}$ y sea $C_n = C\theta$. Por el lema 4.4 existe un operador $\{\Delta/z\}$ tal que $D\{\Delta/z\} = C\theta$ y por el lema 4.3 para todo $j \in \{1, \ldots, n\}$ existe $\{\Delta_j/x_j\}$ tal que $C_j\{\Delta_j/x_j\} = C_{j-1}$ luego

$$C = D\{\Delta/z\}\{\Delta_n/x_n\}\{\Delta_{n-1}/x_{n-1}\}\ldots\{\Delta_1/x_1\}$$

⊣

Ilustramos el corolario con el siguiente ejemplo

Ejemplo 4.7 Consideremos las cláusulas

$$C = \{p(x_1, x_3), q(x_3)\}$$
$$D = \{p(x_2, x_3), q(x_3), r(x_1)\}$$

Si tomamos la sustitución $\theta = \{x_1/x_2\}$ entonces $C\theta \subseteq D$, luego $C \succeq D$. Las variables que ocurren en θ son $\{x_1, x_2\}$. Si tomamos dos variables nuevas *que no ocurran en* C, por ejemplo $\{x_4, x_5\}$ y aplicamos el método de descomposición de

la proposición 4.5 obtenemos $\sigma_1 = \{x_1/x_5\}$, $\sigma_2 = \{x_4/x_1\}$ y $\sigma_3 = \{x_5/x_2\}$. Sean

$$\begin{array}{lllll} C_0 = & C & = & & \{p(x_1,x_3), q(x_3)\} \\ C_1 = & C_0\sigma_1 & = & C_0\{x_1/x_5\} & = \{p(x_5,x_3), q(x_3)\} \\ C_2 = & C_1\sigma_2 & = & C_1\{x_4/x_1\} & = \{p(x_5,x_3), q(x_3)\} \\ C_3 = & C_2\sigma_3 & = & C_2\{x_5/x_2\} & = \{p(x_2,x_3), q(x_3)\} \end{array}$$

y además sabemos que $C_3 = C\theta = \{p(x_2,x_3), q(x_3)\}$.

Si aplicamos el lema 4.4 a las cláusulas

$$\begin{aligned} C_3 = C\theta &= \{p(x_2,x_3), q(x_3)\} \\ D &= \{p(x_2,x_3), q(x_3), r(x_1)\} \end{aligned}$$

con $C\theta \subseteq D$ obtenemos la asignación de posiciones

$$\begin{array}{rcl} \Delta: \mathit{Lit} & \longrightarrow & \mathcal{PPN}^+ \\ L & \mapsto & \Delta(L) = \begin{cases} \emptyset & \text{si } L = r(x_1) \\ \{\emptyset\} & \text{en otro caso} \end{cases} \end{array}$$

y si tomamos, como dice el lema, una variable cualquiera z entonces tenemos que $D\{\Delta/z\} = C_3$. Nos fijamos ahora en que $C_2\{x_5/x_2\} = C_3$. Aplicamos ahora el lema 4.3 y obtenemos la asignación de posiciones

$$\begin{array}{rcl} \Delta_1: \mathit{Lit} & \longrightarrow & \mathcal{PPN}^+ \\ L & \mapsto & \Delta(L) = \begin{cases} \{\{1\}\} & \text{si } L = p(x_2,x_3) \\ \{\emptyset\} & \text{si } L = q(x_3) \\ \{\emptyset\} & \text{en otro caso} \end{cases} \end{array}$$

Por tanto, si aplicamos a $C_3 = \{p(x_2,x_3), q(x_3)\}$ el operador clausal $\{\Delta_1/x_5\}$ obtenemos $C_2 = \{p(x_5,x_3), q(x_3)\}$.

Siguiendo el proceso, tenemos ahora que $C_2 = C_1\sigma_2 = C_1\{x_4/x_1\}$. En este ejemplo, la aplicación de esta sustitución no tiene ningún efecto, puesto que x_4 no ocurre en C_1. No obstante, seguimos el algoritmo y creamos la asignación

$$\begin{array}{rcl} \Delta_2: \mathit{Lit} & \longrightarrow & \mathcal{PPN}^+ \\ L & \mapsto & \Delta(L) = \begin{cases} \{\emptyset\} & \text{si } L = p(x_5,x_3) \\ \{\emptyset\} & \text{si } L = q(x_3) \\ \{\emptyset\} & \text{en otro caso} \end{cases} \end{array}$$

Para cualquier término t, el operador clausal $\{\Delta_2/t\}$ es la identidad en el conjunto de cláusulas, en particular $C_2\{\Delta_2/x_4\} = C_2 = C_1$. Por último, tenemos que $C_1 = C_0\sigma_1 = C_0\{x_1/x_5\}$. Aplicando de nuevo el lema, tenemos la asignación

$$\begin{array}{rccl} \Delta_3: & Lit & \longrightarrow & \mathcal{PPN}^+ \\ & L & \mapsto & \Delta(L) = \left\{ \begin{array}{ll} \{\{1\}\} & \text{si } L = p(x_5, x_3) \\ \{\emptyset\} & \text{si } L = q(x_3) \\ \{\emptyset\} & \text{en otro caso} \end{array} \right. \end{array}$$

y $C = C_0 = C_1\{\Delta_3/x_1\}$, por consiguiente

$$C = D\{\Delta/z\}\{\Delta_1/x_5\}\{\Delta_2/x_4\}\{\Delta_3/x_1\}$$

Nota 4.8 En el ejemplo 4.7 hemos seguido el algoritmo y hemos encontrado cuatro operadores clausales que aplicados a la cláusula más específica nos devuelven la cláusula original. El corolario que ilustra, el 4.6, es un resultado sobre *existencia* de esa cadena de operadores, en ningún caso se afirma que la cadena sea *minimal.* No obstante nos ofrece una *cota* para el número de operadores necesarios para llevar la cláusula D en C si $C \succeq D$.

Si observamos la descomposición de la proposición 4.5, comprobamos que obtenemos una sustitución elemental por cada enlace que ocurre en θ y una sustitución elemental por cada variable que ocurre en θ. Por tanto, podemos afirmar que si $C \succeq D$ y θ es una sustitución tal que $C\theta \subseteq D$, entonces la cadena *con menor número de operadores* que nos lleva D en C será menor o igual que $|\theta| + |Var(\theta)| + 1$, donde $|\theta|$ es el número de enlaces que ocurren en θ y $|Var(\theta)|$ es el cardinal del conjunto de variables que ocurren en θ. A la suma de estas dos cantidades hay que añadirle una unidad correspondiente al operador que generaliza la relación de subconjunto.

Siempre podremos considerar que $Dom(\theta) \subseteq Var(C)$, por tanto tenemos que $|\theta| \leq |Var(C)|$. Por otra parte, $Var(\theta) \subseteq Var(C) \cup Var(D)$, por tanto, $|Var(\theta)| \leq |Var(C)| + |Var(D)|$, de ahí que tengamos como cota superior para el mínimo número de operadores que nos lleva de C a D la cantidad

$$(2 \times |Var(C)|) + |Var(D)| + 1$$

4.4 Los operadores de aprendizaje para subsunción

En la sección anterior, hemos visto que si $C_1 \succeq C_2$, entonces existe un operador clausal ϕ tal que $C_2\phi = C_1$. En esta sección nos planteamos la pregunta en sentido inverso: ¿Qué condiciones debe cumplir un operador clausal ϕ para que, si C es una cláusula cualquiera, $C\phi$ subsuma a C?.

En esta sección vamos a definir un subconjunto de $\mathbb{OP}$ que nos permite caracterizar la relación de subsunción entre cláusulas. Antes, necesitamos algunos resultados. En el primero de ellos se establece una equivalencia que será útil más tarde. La intuición que hay detrás de la formalización es la siguiente: Tomamos un literal L, un término t que ocurra en L y un conjunto de posiciones $P \subseteq Pos(L,t)$. En las posiciones determinadas por P insertamos una variable x y al nuevo literal le aplicamos la sustitución $\{x/t\}$. De manera intuitiva podemos pensar que tras aplicar esta sustitución obtendremos de nuevo el literal L o no dependiendo de las características de L.

Una consecuencia inmediata del lema que probamos a continuación es que con estas operaciones obtenemos el literal original L, esto es, $L\{P/x\}\{x/t\} = L$ si y sólo si se da una de las siguientes condiciones

1. $x = t$

2. $x \neq t$ y $x \notin Var(L\{P/a\})$ donde a es una constante cualquiera.

Para verlo, primero caracterizaremos cuando $(L\{P/x\})\{x/t\}$ y L son distintos (lema 4.9) y a continuación el lema en el que se prueba el resultado (lema 4.10).

Lema 4.9 *Sea L un literal, t un término, x una variable tal que $x \neq t$, $P \subseteq Pos(L,t)$ y a una constante cualquiera. Son equivalentes*

1. $(L\{P/x\})\{x/t\} \neq L$

2. $x \in Var(L\{P/a\})$

Demostración:
(1) $\Rightarrow$ (2) Supongamos $(L\{P/x\})\{x/t\} \neq L$. Sabemos que para todo $v \in P$,

$$(L\{P/x\})\{x/t\}/v = t = L/v$$

Puesto que $(L\{P/x\})\{x/t\} \neq L$ debe existir una posición u tal que

$$(L\{P/x\})\{x/t\}/u \neq L/u$$

Además podemos suponer que, para todo $v \in P$, u es independiente de v ya que si u fuera prefijo de algún $v \in P$, entonces existiría una posición $u\hat{}u_0$ tal que $u\hat{}u_0$ sí es independiente de v para todo $v \in P$ y $(L\{P/x\})/u\hat{}u_0 \neq L/u\hat{}u_0$. Por tanto, supongamos que u es independiente de v para todo $v \in P$. De ahí que $L\{P/x\}/u = L/u$.

Sea t' el término tal que $t' = L\{P/x\}/u = L/u$. Se tiene que

$$t'\{x/t\} = (L\{P/x\}\{x/t\})/u \neq L/u = t'$$

Por tanto $t'\{x/t\} \neq t'$ y debe existir u_1 tal que $t'/u_1 = x$ y $L/u\hat{}u_1 = x$. Puesto que $u\hat{}u_1$ es independiente de todo $x \in P$, $x \in Var(L\{P/a\})$.
(2) $\Rightarrow$ (1) Sea $u \in Pos(L\{P/a\})$ tal que $L\{P/a\}/u = x$. puesto que para todo $v \in P$, $L\{P/a\}/v = a$, se tiene que u es independiente de v para todo $v \in P$. Por tanto $L/u = x$. Por otra parte $(L\{P/x\}/u) = x$ (P y v independientes) y $(L\{P/x\})\{x/t\}/u = t$. Por tanto $(L\{P/x\}\{x/t\} \neq L$ ya que $L/u = x \neq t = (L\{P/x\})\{x/t\}/u$ ⊣

El lema que buscamos, el 4.10, se tiene de manera inmediata del lema 4.9.

Lema 4.10 *Sean L un literal, t un término, $P \subseteq Pos(L,t)$ y a una constante. Entonces $(L\{P/x\})\{x/t\} = L$ si y sólo si se verifica una de las siguientes condiciones:*

1. $x = t$
2. $x \neq t$ y $x \notin Var(L\{P/a\})$

Demostración:
Se tiene trivialmente a partir del lema anterior.
($\Rightarrow$) Supongamos $(L\{P/x\})\{x/t\} = L$. Para ver que se tiene $x \neq t$ o se tiene $x \notin Var(L\{P/a\})$ podemos suponer que $x \neq t$ no se verifica y ver que entonces, necesariamente se tiene $x \notin Var(L\{P/a\})$. Pero esto es precisamente lo que hemos visto en el resultado anterior.
($\Leftarrow$) Si $x = t$ el resultado es trivial. El caso en el que $x \neq t$ y $x \in Var(L\{P/a\})$ se tiene por el lema anterior. ⊣

El siguiente lema es trivial.

Lema 4.11 *Sea $\mathcal{C}$ un conjunto finito de cláusulas y θ una sustitución. Entonces $\cup[\mathcal{C}\theta = [\cup\mathcal{C}]\theta$, donde $\mathcal{C}\theta = \{C\theta \mid C \in \mathcal{C}\}$*

Con el siguiente resultado empezamos a ver la relación entre los operadores clausales y la relación de subsunción. En él se establece una condición suficiente que debe cumplir un operador clausal $\{\Delta/x\}$ para que al aplicarlo a cualquier cláusula obtengamos una más general en el orden de subsunción.

Lema 4.12 *Sea $\{\Delta/x\}$ un operador clausal tal que*

$$(\forall L \in Lit)(\forall P \in \Delta(L))[x \notin Var(L\{P/a\})]$$

donde a es una constante cualquiera. Entonces, para toda cláusula C se verifica que $C\{\Delta/x\} \succeq C$.

A este lema le corresponden los `Ejemplos 4.12` en el apéndice B.

Demostración:

Consideremos los siguientes casos

Caso 1: $(\forall L \in Lit)[\{L\}\{\Delta/x\} \subseteq \{L\}]$

En este caso no es necesaria la hipótesis. Sea C una cláusula. Se tiene que

$$C\{\Delta/x\} = \bigcup_{L\in C}\{L\}\{\Delta/x\} \subseteq \bigcup_{L\in C}\{L\} = C \quad \text{y} \quad C\{\Delta/x\} \succeq C$$

Caso 2: $(\exists L \in Lit)[\{L\}\{\Delta/x\} \not\subseteq \{L\}]$

Sea L_0 un literal tal que $\{L_0\}\{\Delta/x\} \not\subseteq \{L_0\}$ y sea $L_1 \in \{L_0\}\{\Delta/x\}$ tal que $L_1 \neq L_0$. Entonces existe $P_0 \in \Delta(L_0)$, $P_0 \neq \emptyset$, tal que $L_1 = L_0\{P_0/x\}$. Sea $u \in P_0$ y $t_0 = L/u$. Puesto que $L_0 \neq L_1$, $t_0 = x$ y por ser Δ una asignación de posiciones se tiene que

$$(\forall L \in Lit)(\forall P \in \Delta(L))[P \subseteq Pos(L, t_0)]$$

Por tanto existe un término t_0, $t_0 \neq x$ tal que para toda constante a, para todo literal L y para todo $P \subseteq \Delta(L)$ se tiene que $P \subseteq Pos(L, t_0)$ y $x \notin Var(L\{P/a\})$ (esto último por hipótesis). Aplicando el lema 4.10 se tiene que para todo literal L y para todo $P \subseteq \Delta(L)$ se tiene que $(L\{P/x\})\{x/t_0\} = L$. Por tanto, para todo

literal L

$$\begin{array}{rl} [\{L\}\{\Delta/x\}]\{x/t_0\} = & [\bigcup_{P\in\Delta(L)}\{L\{P/x\}\}]\{x/t_0\} \\ \underset{Lema4.11}{=} & \bigcup_{P\in\Delta(L)}[\{L\{P/x\}\}\{x/t_0\}] \\ = & \bigcup_{P\in\Delta(L)}\{L\} \\ = & \{L\} \end{array}$$

Por consiguiente

$$\begin{array}{rl} [C\{\Delta/x\}]\{x/t_0\} = & [\bigcup_{L\in C}\{L\}\{\Delta/x\}]\{x/t_0\} \\ \underset{Lema4.11}{=} & \bigcup_{L\in C}[\{L\}\{\Delta/x\}\{x/t_0\}] \\ = & \bigcup_{L\in C}\{L\} \\ = & C \end{array}$$

Luego $C\{\Delta/x\} \succeq C$. ⊣

Con este resultado hemos encontrado una condición suficiente. Seguimos nuestro estudio para buscar una caracterización de relación de subsunción basada en operadores. La noción de incidencia será clave en nuestra definición de *operador de aprendizaje para la subsunción.* Definimos un operador clausal *incidente* de la siguiente manera:

Definición 4.13 *Decimos que el operador clausal* $\{\Delta/x\}$ *es* incidente *si existe un* único *término* t_0 *tal que*

- *Para todo literal* L, $\Delta(L)$ *es compatible con el par* $\langle L, t_0\rangle$
- x *ocurre en* t_0

A esta definición le corresponde el predicado `operador_incidente/1` en el apéndice B.

Usaremos esta definición en sentido *negativo*, esto es, nos interesa cuando un operador *no* es incidente. Estudiamos sus equivalencias en el siguiente lema.

Proposición 4.14 *Sea* $\{\Delta/x\}$ *un operador clausal. Son equivalentes:*

1. $\{\Delta/x\}$ *no es incidente*
2. *Existe un término* t *tal que* $\Delta(L)$ *es compatible con* $\langle L, t\rangle$ *para todo literal* L *y* x *no ocurre en* t.
3. *Se da una de las siguientes condiciones:*

(a) $(\forall L \in Lit)[\Delta(L) \subseteq \{\emptyset\}]$

(b) *Existen* u, P y L *tales que* $u \in P \in \Delta(L)$ y x *no ocurre en* L/u.

A esta proposición le corresponde el predicado `no_incidente/1` en el apéndice B.

Demostración:

(1) $\Rightarrow$ (3) A partir de la definición, $\{\Delta/x\}$ no es incidente si

(a) El término que permite la compatibilidad no es único, esto es, existen $t_1, t_2 \in Term$, $t_1 \neq t_2$ tales que para todo literal L y para todo $P \in \Delta(L)$, $P \subseteq Pos(L, t_1)$ y $P \subseteq Pos(L, t_2)$. Por tanto para todo literal L, $\Delta(L) \subseteq \{\emptyset\}$ ya que si existen u, P y L_0 con $u \in P \in \Delta(L_0)$ entonces $L/u = t_1 \neq t_2 = L/u$.

(b) La otra posibilidad para que $\{\Delta/x\}$ no sea incidente es que el término que permite la compatibilidad sea único y x no ocurra en ese término. Si el término que permite la compatibilidad es único entonces existe $L_0 \in Lit$ tal que$\Delta(L_0) \not\subseteq \{\emptyset\}$ ya que si para todo literal L, $\Delta(L) = \emptyset$ o bien $\Delta(L) = \{\emptyset\}$ entonces para todo término t, $\Delta(L)$ sería compatible con $\langle L, t\rangle$. Por tanto, sea L_0 un literal tal que $\Delta(L_0) \not\subseteq \{\emptyset\}$ y sean u y P tales que $u \in P \in \Delta(L_0)$. Entonces $t_0 = L/u$ es el único término que permite la compatibilidad y por hipótesis x no ocurre en t_0.

(3) $\Rightarrow$ (1) Consideremos también los dos casos

(a) Si para todo literal L se tiene que $\Delta(L) \subseteq \{\emptyset\}$ entonces, para todo término t y para todo literal L, $\Delta(L)$ es compatible con $\langle L, t\rangle$, luego el término que permite la compatibilidad no es único y $\{\Delta/x\}$ no es incidente.

(b) Si existen u, P y L_0 tales que $u \in P \in \Delta(L_0)$ y $t_0 = L_0/u$, por ser Δ una asignación de posiciones, para todo literal L, $\Delta(L)$ es compatible con $\langle L, t_0\rangle$ y si para todo literal L, $\Delta(L)$ fuera compatible con $\langle L, t_1\rangle$ entonces $t_0 = L/u = t_1$. Luego el término t_0 que permite la compatibilidad es único y por hipótesis x no ocurre en $L/u = t_0$.

(2) $\Rightarrow$ (3) Supongamos que no se tiene (a) y veamos que entonces se tiene (b). Si no se tiene (a) entonces existe un único término t_0 tal que para todo literal L, $\Delta(L)$ es compatible con $\langle L, t_0\rangle$ y existen u, P y L tales que $u \in P \in \Delta(L)$ con $L/u = t_0$ y por hipótesis x no ocurre en t_0.

(3) $\Rightarrow$ (2) En este caso también lo vemos por partes

(a) Si para todo literal L, $\Delta(L) \subseteq \{\emptyset\}$, entonces basta tomar $t = z$, con z una variable distinta de x, ya que si $\Delta(L) = \emptyset$ o bien $\Delta(L) = \{\emptyset\}$ entonces $\Delta(L)$ es compatible con $\langle L, z\rangle$ y x no ocurre en z.

(b) Si existen u, P y L_0 con $u \in P \in \Delta(L_0)$ y $t_0 = L/u$, por ser Δ una asignación de posiciones, para todo literal L, $\Delta(L)$ es compatible con $\langle L, t_0\rangle$ y por hipótesis x no ocurre en $L/u = t_0$

⊣

Llegamos a la definición que buscamos. Definimos a continuación los *operadores de aprendizaje para subsunción, OAS* con los que caracterizaremos la relación de subsunción entre cláusulas.

Definición 4.15 *Se dice que un operador clausal* $\{\Delta/x\}$ *es un* operador de aprendizaje para subsunción, OAS *si*

1. *No es incidente*

2. $(\forall L \in Lit)(\forall P \in \Delta(L))[x \notin Var(L\{P/a\})]$ *donde* a *es una constante cualquiera.*

A esta definición le corresponde el predicado `oas_soporte_finito/1` en al apéndice B.

Veamos a continuación una caracterización equivalente a la definición anterior

Teorema 4.16 *Sea* $\{\Delta/x\}$ *un operador clausal. Son equivalentes:*

1. $\{\Delta/x\}$ *es un OAS*

2. $(\forall L \in Lit)[\Delta(L) \neq \emptyset \rightarrow x \notin Var(L)]$

Demostración:
(1) $\Rightarrow$ (2) Supongamos que $\{\Delta/x\}$ es un OAS, esto es

(I) No es incidente

(II) $(\forall L \in Lit)(\forall P \in \Delta(L))[x \notin Var(L\{P/a\})]$ donde a es una constante cualquiera.

Sea L un literal tal que $\Delta(L) \neq \emptyset$. Pueden darse los siguientes casos:

Caso 1: $\emptyset \in \Delta(L)$

Puesto que $\{\Delta/x\}$ es un OAS, se verifica (II) y por tanto $(\forall P \in \Delta(L))[x \notin Var(L\{P/a\})]$ En particular, si $P = \emptyset$

$$x \notin Var(L\{P/a\}) = Var(L\{\emptyset/a\}) = Var(L)$$

Caso 2: $\emptyset \notin \Delta(L)$

Puesto que $\Delta(L) \neq \emptyset$, existe $P \neq \emptyset$ con $P \in \Delta(L)$. Sea $u \in P$ y sea $L/u = t$ el único término que permite la compatibilidad de Δ. Por hipótesis, $\{\Delta/x\}$ es un OAS, luego no incidente, y por la proposición 4.14, apdo. 3, x no ocurre en t. Por otra parte $P \subseteq Pos(L,t)$ luego

$$[Var(L) - Var(t)] \subseteq Var(L[P \leftarrow a])$$

y por ser $\{\Delta/x\}$ un OAS

$$x \notin Var(L[P \leftarrow a])$$

por tanto $x \notin Var(L) - Var(t)$ y puesto que x no ocurre en t, llegamos a que x no ocurre en L.

(1) $\Rightarrow$ (2) Para demostrar que el operador clausal $\{\Delta/x\}$ es un OAS hay que probar:

(I) No es incidente

(II) $(\forall L \in Lit)(\forall P \in \Delta(L))[x \notin Var(L\{P/a\})]$ donde a es una constante cualquiera.

Lo vemos por partes.

(I) $\{\Delta/x\}$ no es incidente.

Usaremos la proposición 4.14, apdo. 3. Tenemos que probar por tanto que se tiene una de las siguientes condiciones

(a) $(\forall L \in Lit)[\Delta(L) \subseteq \{\emptyset\}]$

(b) Existen u, P y L tales que $u \in P \in \Delta(L)$ y x no ocurre en L/u.

Supongamos que no se tiene (a), esto es, supongamos que existe un literal L tal que $\Delta(L) \not\subseteq \{\emptyset\}$. Entonces existen u y P tales que $u \in P \in \Delta(L)$. Para ver que

se tiene (b), y por tanto que $\{\Delta/x\}$ no es incidente, basta ver que x no ocurre en L/u, pero esto se tiene por hipótesis, ya que estamos suponiendo

$$(\forall L \in Lit)[\Delta(L) \neq \emptyset \rightarrow x \notin Var(L)]$$

y además $\Delta(L) \neq \emptyset$.

(II) $(\forall L \in Lit)(\forall P \in \Delta(L))[x \notin Var(L[P \leftarrow a])]$

Sea L un literal cualquiera. Si $\Delta(L) = \emptyset$ es resultado se tiene trivialmente. Si $\Delta(L) \neq \emptyset$ por hipótesis $x \notin Var(L)$ y por tanto

$$(\forall P \in \Delta(L))[x \notin Var(L[P \leftarrow a])]$$

⊣

Vemos a continuación el primer resultado que relaciona los OAS con la subsunción. El siguiente lema nos dice que los OAS son *operadores de generalización*, en el sentido de la definición 4.2 para la relación de subsunción.

Lema 4.17 *Sean C y D dos cláusulas y $\{\Delta_1/x_1\}, \ldots \{\Delta_n/x_n\}$ OAS tales que*

$$C = D\{\Delta_1/x_1\}\{\Delta_2/x_2\} \ldots \{\Delta_n/x_n\}$$

Entonces $C \succeq D$.

Demostración:

Se tiene trivialmente a partir de la definición de OAS, del lema 4.12 y la transitividad de la relación de subsunción. ⊣

4.5 Caracterización de la relación mediante OAS

El objetivo de esta sección es caracterizar unos operadores clausales, que llamaremos *operadores de aprendizaje para la subsunción, OAS* de manera que para dos cláusulas cualesquiera C y D, C subsume a D o dicho de otro modo, C es más general que D en el orden de subsunción si y sólo si existe una cadena de OAS Ψ tal que $\Psi(D) = C$. Para ello vamos a ver que con construcciones análogas a las de los lemas 4.3 y 4.4 podemos encontrar OAS que generalizan la relación de subconjunto y las sustituciones elementales.

Lema 4.18 *Sean C y D dos cláusulas tales que $C \subseteq D$. Entonces existe un OAS $\{\Delta/x\}$ tal que $D\{\Delta/x\} = C$.*

A este lema le corresponde el predicado `oas_asociado_subconjunto/2` en el apéndice B.

Demostración:

La construcción del operador es análoga a la del lema 4.4. Sólo debemos tener en cuenta que la variable que insertamos no debe pertenecer a C.

Sea x una variable tal que $x \notin Var(C)$ y sea

$$\begin{array}{rccl} \Delta: & Lit & \longrightarrow & \mathcal{P}\mathcal{P}\mathbb{N}^+ \\ & L & \mapsto & \Delta(L) = \left\{ \begin{array}{ll} \{\emptyset\} & \text{si } L \in C \\ \emptyset & \text{si } L \notin C \end{array} \right. \end{array}$$

Por la proposición 4.14, $\{\Delta/x\}$ no es incidente. Para ver que $\{\Delta/x\}$ es un OAS queda por ver que para todo literal L se tiene que

$$(\forall P \in \Delta(L))[x \notin Var(L\{P/a\})]$$

Sea L un literal

- $L \notin C$. Se tiene trivialmente.
- $L \in C$. En este caso también se verifica, ya que si $P \in \Delta(L)$, entonces $P = \emptyset$ y $L\{\emptyset/a\} = L$ y por hipótesis $x \notin Var(C)$, luego $x \in Var(L)$.

Por tanto, $\{\Delta/x\}$ es un OAS. Por último

$$\begin{array}{rcl} D\{\Delta/x\} & = & \bigcup_{L \in D}\{L\}\{\Delta/x\} \\ & = & [\bigcup_{L \in C}\{L\}\{\Delta/x\}] \cup [\bigcup_{L \in D-C}\{L\}\{\Delta/x\}] \\ & = & [\bigcup_{L \in C}\{L\}] \cup \emptyset \\ & = & C \end{array}$$

⊣

En el lema siguiente vemos cómo podemos encontrar un OAS que invierta una sustitución elemental. En este caso la construcción es análoga a la del lema 4.3. El único detalle a tener en cuenta es que si la sustitución elemental es $\theta = \{x/t\}$, la variable x no puede ocurrir en t. Esto no representa ninguna limitación para el

caso general, puesto que como vimos en la proposición 4.5 en la descomposición de una sustitución en elementales $\sigma_i = \{x_i/t_i\}$, x_i no ocurre en t_i.

Lema 4.19 *Sean C y D dos cláusulas y $\theta = \{x/t\}$ una sustitución elemental tal que x no ocurra en t. Si $C\theta = D$ entonces existe un OAS $\{\Delta/x\}$ tal que $C = D\{\Delta/x\}$*

A este lema le corresponde el predicado `oas_asociado_sustitucion/4` en el apéndice B.

Demostración:

Para todo literal L y toda sustitución θ definimos

$$\theta^{-1}(L) = \{L' \in Lit \mid L'\theta = L\}$$

Consideremos la asignación de posiciones

$$\begin{array}{rccc} \Delta: & Lit & \longrightarrow & \mathcal{P}\mathcal{P}\mathbb{N}^+ \\ & L & \mapsto & \Delta(L) \end{array}$$

$$\Delta(L) = \begin{cases} \{Pos(L', x) \mid L' \in \theta^{-1}(L) \cap C\} & \text{si } L \in D \\ \emptyset & \text{si } L \notin D \end{cases}$$

donde $\theta = \{x/t\}$. Veamos que $\{\Delta/x\}$ es un OAS.

En primer lugar, si $L'\{x/t\} = L$ entonces $Pos(L', x) \subseteq Pos(L, t)$ por tanto, para todo literal L, $\Delta(L)$ es compatible con el par $\langle L, t\rangle$. Por hipótesis x no ocurre en t, y aplicando la proposición 4.14 se tiene que $\{\Delta/x\}$ es no incidente.

Veamos que para todo literal L

$$(\forall P \in \Delta(L))[x \notin Var(L\{P/a\})]$$

- Si $L \notin D$ se tiene trivialmente
- Si $L \in D$ entonces tenemos que considerar dos casos

Caso 1: $\theta^{-1}(L) \cap C = \emptyset$

Se tiene trivialmente. En este caso, para todo literal L, $\Delta(L) = \emptyset$.

Caso 2: $\theta^{-1}(L) \cap C \neq \emptyset$

Si $L' \in \theta^{-1} \cap C$ entonces $L'\{x/t\} = L$. Por hipótesis sabemos que

x no ocurre en t, por tanto x no ocurre en L y tampoco ocurre en $L[Pos(L', x) \leftarrow a]$. Luego $\{\Delta/x\}$ es un OAS. Por último

$$\begin{aligned} D\{\Delta/x\} &= \bigcup_{L \in D} L\{\Delta(L)/x\} \\ &= \bigcup_{L \in D} L\{\{Pos(L', x) \mid L' \in \theta^{-1}(L) \cap C\}/x\} \\ &= \bigcup_{L \in D} \bigcup_{L' \in \theta^{-1}(L) \cap C} \{L\{Pos(L', x)/x\}\} \\ &= \bigcup_{L \in D} \bigcup_{L' \in \theta^{-1}(L) \cap C} \{L'\} \\ &= \bigcup_{L \in D} (\theta^{-1}(L) \cap C) \\ &= [\bigcup_{L \in D} \theta^{-1}(L)] \cap C) \\ &= C \end{aligned}$$

En esta última igualdad hemos usado que si $C\theta \subseteq D$ entonces

$$C \subseteq \bigcup_{L \in D} \theta^{-1}(L)$$

Lo vemos. Sea $L' \in C$. Entonces existe $L \in D$ tal que $L'\theta = L$, luego $L' \in \theta^{-1}(L)$ y

$$L' \in \bigcup_{L \in D} \theta^{-1}(L)$$

⊣

Teorema 4.20 *Sean C y D dos cláusulas tales que $C \succeq D$. Entonces existen OAS, $\{\Delta_0/x_0\}, \{\Delta_1/x_1\}, \ldots, \{\Delta_m/x_m\}$ tales que*

$$C = D\{\Delta_0/x_0\}\{\Delta_1/x_1\} \ldots \{\Delta_m/x_m\}$$

A este teorema le corresponde el predicado `genera_oas/3` en el apéndice B.

Demostración:

Sea θ una sustitución tal que $C\theta \subseteq D$ y $Dom(\theta) \subseteq Var(C)$. Por el lema 4.18 existe un OAS $\{\Delta_0/x_0\}$ tal que $D\{\Delta_0/x_0\} = C\theta$. Si θ es la sustitución vacía, ya hemos terminado. Si no, $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ y por la proposición 4.5 existen $\sigma_1, \ldots, \sigma_m$ sustituciones elementales $\sigma_i = \{z_i/s_i\}$ tales que

- $(\forall i \in \{1,\ldots,n\})[x_i(\sigma_1\,\sigma_2\,\ldots\,\sigma_m) = t_i]$
- Para todo j, con $1 \leq j \leq m$, z_i no ocurre en s_i.

Puesto que $\{x_1,\ldots,x_n\} \subseteq Var(C)$, basta elegir las variables *nuevas* que aparecen en los σ_i tomándolas entre las que no son variables de C. Se tiene entonces

$$C\theta = C\sigma_1\sigma_2\ldots\sigma_m$$

Sean $C_0 = C$, $C_m = C\theta$ y $C_{i+1} = C_i\sigma_{i+1}$, $i \in \{0,\ldots,m-1\}$. Gráficamente

$$C = C_0 \xrightarrow{\sigma_1} C_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_{m-1}} C_{m-1} \xrightarrow{\sigma_m} C_m = C\theta \subseteq D$$

Puesto que $\sigma_i = \{z_i/s_i\}$ y z_i no ocurre en s_i, por el lema 4.19 existen m OAS

$$\{\Delta_1/z_1\}, \{\Delta_2/z_2\}, \ldots, \{\Delta_m/z_m\}$$

tales que $C_i\{\Delta_i/z_i\} = C_{i-1}$, $i \in \{1,\ldots,m\}$, luego

$$C = D\{\Delta_1/z_1\}\{\Delta_2/z_2\}\ldots\{\Delta_m/z_m\}\{\Delta_0/x_0\}$$

⊣

Por último, combinando el teorema 4.20 y el lema 4.17 tenemos el resultado pedido.

Corolario 4.21 *Sean C y D dos cláusulas. Son equivalentes:*

1. $C \succeq D$
2. *Existe una sucesión finita n OAS $\{\Delta_1/x_1\}, \{\Delta_2/x_2\}, \ldots, \{\Delta_n/x_n\}$ tales que*

$$C = D\{\Delta_n/x_n\}\{\Delta_{n-1}/x_{n-1}\}\ldots\{\Delta_1/x_1\}$$

Demostración:
Por el teorema 4.20 y el lema 4.17. ⊣

Con el teorema 4.20 hemos probado la relación existente entre las sucesiones de OAS y la relación de subsunción entre cláusulas. Sabemos que por la aplicación sucesiva de OAS a una cláusula siempre vamos a obtener cláusulas más generales

(o equivalentes) bajo el orden de subsunción que la cláusula original. Además sabemos que si dos cláusulas están relacionadas por subsunción podemos alcanzar la más general a partir de la más específica aplicando una sucesión finita de OAS. Esta cadena finita de OAS que nos lleva una cláusula en otra puede no ser única, basta tener en cuenta que ni siquiera la sustitución necesaria para comprobar la relación de subsunción es única, y por consiguiente, podemos encontrar cadenas de OAS de diferente longitud.

Esto nos lleva a pensar en la formalización de la idea de *proximidad basada en subsunción* relacionada con esa cadena de OAS. Dedicaremos el resto del capítulo a estudiar esta cuestión.

4.6 Quasi–métricas

En Aprendizaje Automático hay una creciente necesidad de formalizar el concepto de proximidad en espacios cada vez más abstractos. En Programación Lógica Inductiva, el problema de cuantificar la proximidad entre cláusulas ha sido estudiado, entre otros, por A. Hutchinson [Hut97] y S.–H. Nienhuys–Cheng [NC97], ofreciendo distintas alternativas de solución al problema. En ambos casos se define primero una distancia entre literales y luego se usa la métrica de Hausdorff para obtener a partir de esta una distancia entre cláusulas. Esto tiene dos desventajas. Por un lado la métrica de Hausdorff depende exclusivamente de los puntos extremos (véase, p.e. [EM97]) y por otro, estos literales se consideran aislados y en ningún momento se consideran las posibles relaciones entre los literales de la misma cláusula.

En la parte final de este capítulo, proponemos una solución al problema de cuantificar la proximidad entre cláusulas, considerándolas como elementos de un entramado de relaciones vía subsunción que nos va a permitir acceder de una cláusula a otra, en cierto sentido, por el camino más corto. Esta aproximación representa una importante diferencia con [Hut97] y [NC97], que consideran los literales como elementos aislados.

La relación de subsunción, como relación binaria definida en el conjunto de cláusulas, no es una relación simétrica. Ello nos lleva a pensar que la noción de *distancia* es excesivamente restrictiva para formalizar la idea de proximidad basada en subsunción.

Las funciones de distancia no simétricas ya fueron consideradas por Hausdorff [Hau14] a principios de siglo. Wilson [Wil31] introdujo el término *quasi–metrics*

UNIVERSIDAD DE SEVILLA
FACULTAD DE MATEMÁTICAS
BIBLIOTECA

para estas funciones en 1931. A lo largo del siglo diversos investigadores han contribuido al desarrollo de las distancias no simétricas, recibiendo recientemente un nuevo empuje con los trabajos en computación teórica de Lawson [Law91] o Smyth [Smy91] entre otros.

Definición 4.22 ([Smy91]) *Una quasi–métrica sobre un conjunto X es una aplicación de $X \times X$ en los reales no negativos, incluyendo posiblemente $+\infty$ tal que*

- $(\forall x \in X)\,[d(x,x) = 0]$
- $(\forall x, y \in X)\,[d(x,y) = d(y,x) = 0 \Rightarrow x = y]$
- $(\forall x, y, z \in X)\,[d(x,z) \leq d(x,y) + d(y,z)]$

Nótese que una quasi–métrica verifica las condiciones[2] para poder desarrollar una de métrica de Fréchet, excepto la condición de simetría. Veamos algunos ejemplos (otros ejemplos más sofisticados pueden encontrarse en [Smy91]).

Ejemplo 1: Dado cualquier conjunto parcialmente ordenado $\langle P, \leq \rangle$, la *quasi–métrica discreta* se define como

$$d(x,y) = \begin{cases} 0 & \text{si } x \leq y \\ 1 & \text{e.o.c.} \end{cases}$$

Ejemplo 2: En el intervalo unidad $[0,1]$ podemos definir la quasi–métrica siguiente, cuya métrica asociada es la distancia euclídea en el conjunto.

$$d(x,y) = \begin{cases} 0 & \text{si } x \leq y \\ x - y & \text{si } y < x \end{cases}$$

[2]Fue Fréchet en su tesis doctoral en 1906 [Fré06] quien afirmó que bastaba que la función verificara

- $(\forall x \in X)\,[d(x,x) = 0]$
- $(\forall x, y \in X)\,[d(x,y) = 0 \Rightarrow x = y]$
- $(\forall x, y \in X)\,[d(x,y) = d(y,x)]$ *(Condición de simetría)*
- $(\forall x, y, z \in X)\,[d(x,z) \leq d(x,y) + d(y,z)]$ *(Propiedad triangular)*

para ser una métrica.

4.7 Una quasi–métrica sobre cláusulas

En 4.20 hemos probado que si C y D son dos cláusulas tales que $C \succeq D$ entonces existe una sucesión de OAS $\{\Delta_0/x_0\}, \{\Delta_1/x_1\}, \ldots, \{\Delta_m/x_m\}$ tales que

$$C = D\{\Delta_0/x_0\}\{\Delta_1/x_1\}\ldots\{\Delta_m/x_m\}$$

Vamos a usar esas sucesiones de OAS para formalizar la proximidad entre cláusulas. En nuestra definición, la distancia entre dos cláusulas vendrá determinada por la longitud del *camino* más corto entre ellas, considerando como camino la sucesión de cláusulas asociada a una cadena de OAS.

Definición 4.23 *Una* cadena de OAS *(o simplemente* cadena*) de longitud n de la cláusula D a la cláusulas C es una sucesión de n OAS*

$$\{\Delta_1/x_1\}, \{\Delta_2/x_2\}, \ldots, \{\Delta_n/x_n\}$$

tales que

$$C = D\{\Delta_1/x_1\}, \{\Delta_2/x_2\}, \ldots, \{\Delta_n/x_n\}$$

Si $C = D$, consideraremos que la cadena vacía*, de longitud cero, lleva C en D. Denotaremos como $\mathbf{L}(C, D)$ el conjunto de todas las cadenas de C a D y si $\mathcal{C}$ es una cadena, $|\mathcal{C}|$ denotará su longitud.*

A continuación definimos nuestra quasi–métrica. Si $D \succeq C$ entonces existe al menos una cadena de C a D, (el conjunto $\mathbf{L}(C, D)$ no es vacío) y tiene sentido considerar el mínimo del conjunto de longitudes de caminos en $\mathbf{L}(C, D)$.

Siguiendo la intuición geométrica, si consideramos esas cadenas como *caminos* de C a D, podemos definir nuestra quasi–métrica como la longitud del camino más corto de C a D. Si no existe ningún camino, esto es, si D no subsume a C, pensamos que D no puede ser alcanzado desde C mediante OAS, así que están separados por una *distancia infinita.*

Definición 4.24 *Definimos la aplicación $dc : \mathbb{C} \times \mathbb{C} \to [0, +\infty]$ de la siguiente manera*

$$dc(C, D) = \begin{cases} min\{|\mathcal{C}| : \mathcal{C} \in \mathbf{L}(C, D)\} & si\ C \succeq D \\ +\infty & e.o.c. \end{cases}$$

Teorema 4.25 *dc es una quasi–métrica*

Demostración:

(1) Puesto que para toda cláusula C se tiene que $C \succeq C$, entonces la cadena vacía lleva C en C y el mínimo de las longitudes de las cadenas es cero.

(2) Si $dc(C,D) = dc(D,C) = 0$, entonces podemos pasar de C a D (y viceversa) sin usar ningún operador, esto es, $C = D$.

(3) Tenemos que probar que $dc(C_1,C_3) \leq dc(C_1,C_2) + dc(C_2,C_3)$. Si $C_1 \not\succeq C_2$ o $C_2 \not\succeq C_3$ el resultado se tiene trivialmente, luego supongamos $C_1 \succeq C_2$ y $C_2 \succeq C_3$. Si la cadena de menor longitud de C_1 a C_2 tiene longitud n y la cadena de menor longitud de C_2 a C_3 tiene longitud m entonces, componiendo ambas cadenas obtenemos una cadena de longitud $n+m$ que va de C_1 a C_3, por tanto, la cadena de longitud mínima de C_1 a C_3 tendrá una longitud menor o igual que $n+m$, esto es

$$d([C_1],[C_3]) \leq= n+m = d([C_1],[C_2]) + d([C_2],[C_3])$$

⊣

Nota 4.26 La función dc no es simétrica, ni tan siquiera cuando las cláusulas son equivalentes bajo subsunción. Esto se debe a que, en cierto sentido, dc mide el número mínimo de transformaciones sintácticas necesarias para llevar una cláusula en otra y dos cláusulas equivalentes pueden ser sintácticamente muy diferentes. Basta considerar las siguientes cláusulas

$$\begin{aligned} C = &\ \{p(x_1,y_1), p(a,b), q(a,b,c)\} \\ D = &\ \{p(a,b), q(a,b,c), q(x_2,y_2,z_2\} \end{aligned}$$

Si $\theta_1 = \{x_1/a, y_1/b\}$ y $\theta_2 = \{x_2/a, y_2/b, z_2/c\}$ entonces $C\theta_1 \subseteq D$ y $D\theta_2 \subseteq C$, luego $C \sim D$ y un simple cálculo nos lleva a que $dc(C,D) = 2$ y $dc(D,C) = 3$.

Por tanto dc es una quasi–métrica. De esta manera, $dc(C,D)$ codifica de manera numérica suficiente información sobre la relación de subsunción entre C y D y permite un tratamiento algebraico de la relación de proximidad.

4.8 Trabajos relacionados

En la literatura puede encontrarse diversas aproximaciones al problema de cuantificar la relación de proximidad entre cláusulas. Nuestra propuesta se suma al

esfuerzo de arrojar luz sobre el problema.

En [NC97], Nienhuys-Cheng define una distancia para átomos cerrados

- $d_{nc,g}(e,e) = 0$
- $p/n \neq q/m \Rightarrow d_{nc,g}(p(s_1,\ldots,s_n), q(t_1,\ldots,t_m)) = 1$
- $d_{nc,g}(p(s_1,\ldots,s_n), p(t_1,\ldots,t_n)) = \frac{1}{2n}\sum_{i=1}^{n} d_{nc,g}(s_i,t_i)$

y luego considera la métrica de Hausdorff para trasladar esa distancia a conjuntos de átomos.

$$d_h(A,B) = \max\{\max_{a\in A}\{\min\{d_{nc,g}(a,b) \mid b \in B\}\}, \max_{b\in B}\{\min\{d_{nc,g}(a,b) \mid a \in A\}\}\}$$

El objetivo de esta distancia es definir una métrica entre interpretaciones de Herbrand, así que $d_{nc,g}$ estaba sólo definida sobre átomos cerrados. En [RB98], Ramon y Bruynooghe extendieron esta distancia a una función sobre expresiones cerradas y no cerradas:

- $d_{nc}(p(s_1,\ldots,s_n), X) = d_{nc}(X, p(s_1,\ldots,s_n)) = 1$ con X una variable.
- $d_{nc}(X,Y) = 1$ y $d_{nc}(X,X) = 0$ para todo $X \neq Y$ con X e Y variables.
- En otro caso, seguimos la definición de $d_{nc,g}$

Podemos extender fácilmente esta métrica a literales: Si A y B son átomos, consideramos $d_{nc}(\neg A, B) = d_{nc}(A, \neg B) = 0$ y $d_{nc}(\neg A, \neg B) = d_{nc}(A,B)$. Aplicando a d_{nc} la métrica de Hausdorff obtenemos una distancia sobre cláusulas, como muestra el siguiente ejemplo:

Ejemplo 4.27 Consideremos los átomos $L_1 = p(x_1)$, $L_2 = \{q(f(f(x_2)))\}$, $M_1 = p(a)$ y $M_2 = \{q(f(f(b)))\}$ y las cláusulas

$$C_1 = \{L_1, L_2\} \qquad C_2 = \{L_1\}$$
$$D_1 = \{M_1, M_2\} \qquad D_2 = \{M_1\}$$

Además se verifica que $C_1 \succeq D_1$ y $C_2 \succeq D_2$. Usando la métrica d_{nc} tenemos que

$$d_{nc}(L_1, M_1) = \tfrac{1}{2} \qquad d_{nc}(L_1, M_2) = 1$$
$$d_{nc}(L_2, M_1) = 1 \qquad d_{nc}(L_2, M_2) = \tfrac{1}{8}$$

Aplicando la métrica de Hausdorff para obtener ahora distancias entre cláusulas, tenemos que

$$d_h(C_1, D_1) = d_h(C_2, D_2) = \frac{1}{2}$$

En este caso vemos que los literales L_2 y M_2 son irrelevantes para el cálculo de la distancia entre C_1 y D_1. Sin embargo, al considerar la *proximidad* basada en la noción de subsunción, la relación entre L_2 y M_2 queda reflejada en nuestra quasi-distancia. Sean Δ_1 y Δ_2 las asignaciones

$$\begin{array}{rccl} \Delta_1: & Lit & \longrightarrow & \mathcal{PPN}^+ \\ & L & \mapsto & \Delta_1(L) = \begin{cases} \{\{\{1\}\}\} & \text{si } L = p(a) \\ \{\emptyset\} & \text{si } L = q(f(f(b))) \\ \emptyset & \text{en otro caso} \end{cases} \end{array}$$

$$\begin{array}{rccl} \Delta_1: & Lit & \longrightarrow & \mathcal{PPN}^+ \\ & L & \mapsto & \Delta_1(L) = \begin{cases} \{\{\{1 \cdot 1 \cdot 1\}\}\} & \text{si } L = q(f(f(b))) \\ \{\emptyset\} & \text{si } L = p(x_1) \\ \emptyset & \text{en otro caso} \end{cases} \end{array}$$

Entonces, tenemos que $D_1\{\Delta_1/x_1\}\{\Delta_2/x_2\} = C_1$ y $D_2\{\Delta_1/x_1\} = C_2$ y además no podemos llevar D_2 en C_2 ni D_1 en C_1 con cadenas de longitud menor, por tanto

$$dc(C_1, D_1) = 2 \qquad \text{y} \qquad dc(C_1, D_2) = 1$$

En [Hut97], Hutchinson da una pseudo-métrica sobre el conjunto de términos y la extiende al conjunto de literales. Entonces, considera la métrica de Hausdorff sobre el conjunto de cláusulas usando su pseudo–métrica sobre literales.

En su definición de distancia sobre términos, usa una función del conjunto de sustituciones sobre $\mathbb{R}$ llamada *size*. Da las condiciones que tiene que satisfacer una función para ser una *size* y da una función concreta con esas características

$$S(\theta) = \sum\{w_{f/n} \mid (\exists x)\,(x \in Var \text{ y } f/n \text{ ocurre en } x\theta)\}$$

donde $w_{f/n}$ es un peso positivo para el símbolo de función f/n. Con la métrica de Hausdorff basada en esa pseudo–métrica tenemos que para las cláusulas

$$C_1 = \{p(X, X, Y, Y)\} \qquad C_2 = \{p(U, V, U, V)\}$$

obtenemos los valores $d_h(C_1, C_1) = 0$ y $d_h(C_1, C_2) = 0$ a pesar de que C_1 y C_2 no

son ni siquiera comparables bajo subsunción.

Con nuestra función dc, al no existir ningún camino de C_1 a C_2, esa relación de inaccesibilidad se codifica con el símbolo $+\infty$.

$$dc(C_1, C_2) = dc(C_2, C_1) = +\infty$$

4.9 Cálculo efectivo de la quasi-distancia de subsunción

En esta sección[3] vamos a estudiar cómo obtener la quasi-distancia de subsunción entre dos cláusulas de manera efectiva. Para ello vamos a probar un resultado que relaciona esta quasi-distancia con la descomposición de una sustitución en sustituciones elementales

Definición 4.28 *Dada una sustitución θ se define el conjunto*

$$Desc(\theta) = \left\{ \sigma_1 \dots \sigma_n : \begin{array}{l} \sigma_i = \{x_i/t_i\} \quad x_i \notin Var(t_i) \\ (\forall z \in Dom(\theta))[z\theta = z\sigma_1 \dots \sigma_n] \end{array} \right\}$$

Si $\sigma_1 \dots \sigma_n$ es una sucesión de n sustituciones elementales, diremos que la longitud de la sucesión es n y lo representaremos por

$$long(\sigma_1 \dots \sigma_n) = n$$

A continuación definimos el peso de una sustitución.

Definición 4.29 *Sea θ una sustitución*

$$Peso(\theta) = min\{long(\Sigma) \mid \Sigma \in Desc(\theta)\}$$

Las proposición siguiente relaciona el peso de una sustitución y la quasi-distancia de subsunción.

Proposición 4.30 *Sean C y D dos cláusulas tales que $C \succeq D$. Entonces*

$$min\{Peso(\theta) \mid C\theta \subseteq D\} \leq dc(C, D)$$

[3]El predicado correspondiente al cálculo de la quasi-distancia en el apéndice B es `d_subsuncion/3`. En el mismo apéndice se encuentran los distintos auxiliares.

Demostración:
Supongamos que $dc(C, D) = n$ y sea $\mathcal{C} \in \mathbf{L}(C, D)$ tal que $dc(C, D) = |\mathcal{C}|$ esto es, $\mathcal{C}$ es una cadena de n OAS $\{\Delta_1/x_1\} \ldots \{\Delta_n/x_n\}$ tal que

$$D\{\Delta_1/x_1\} \ldots \{\Delta_n/x_n\} = C$$

Sean $D_0 = D$ y para todo $i \in \{1, \ldots, n\}$ sea $D_i = D_{i-1}\{\Delta_i/x_i\}$ con $D_n = C$. Para todo $i \in \{1, \ldots, n\}$ se tiene lo siguiente:

- Si $(\forall L \in Lit)[\Delta_i(L) \subseteq \emptyset]$ entonces $D_i \subseteq D_{i-1}$
- En otro caso, existe un único término t_i que permite la compatibilidad de Δ_i. Además, por ser $\{\Delta_i/x_i\}$ un OAS tenemos que x_i no ocurre en t_i. Sea $\sigma_i = \{x_i/t_i\}$. Tenemos que $D_i\sigma_i \subseteq D_{i-1}$.

Por consiguiente existe una sucesión $\sigma_k\sigma_{k-1} \ldots \sigma_1$ con $k \leq n$ de sustituciones elementales tales que si $\sigma_i = \{x_i/t_i\}$, entonces $x_i \notin Var(t_i)$ verificando

$$C\sigma_k\sigma_{k-1} \ldots \sigma_1 \subseteq D$$

Sea θ la composición de sustituciones elementales $\sigma_k \circ \sigma_{k-1} \circ \cdots \circ \sigma_1$. Se tiene que $\sigma_k\sigma_{k-1} \ldots \sigma_1 \in Desc(\theta)$ y $C\theta \subseteq D$, luego

$$min\{Peso(\theta) \mid C\theta \subseteq D\} \leq k \leq n = dc(C, D)$$

⊣

A continuación presentamos otra proposición que nos da la otra desigualdad en el caso en que $C \not\subseteq D$, pero antes necesitamos un lema técnico.

Lema 4.31 *Sea θ una sustitución y $\delta_1 \ldots \delta_k \in Desc(\theta)$ con $long(\delta_1 \ldots \delta_k) = k$. Sea V^* un conjunto finito de variables tal que $V^* \cap Dom(\theta) = \emptyset$. Entonces existe $\sigma_1 \ldots \sigma_k \in Desc(\theta)$ con $long(\sigma_1 \ldots \sigma_k) = k$ tal que para toda $z \in V^*$, $z\sigma_1 \ldots \sigma_k = z$.*

Demostración:
Sean $\delta_i = \{x_i/t_i\}$ y sea $z_1 \in V^*$ tal que $z_1\delta_1 \ldots \delta_k \neq z_1$. Entonces, para algún $i \in \{1 \ldots, k\}$, $\delta_i = \{z_1/t_i\}$, esto es, $x_i = z_1$. Sea i_1^1 el menor índice i tal que

$x_i = z_1$ y sea y_1^1 una variables que no ocurra en $Dom(\theta) \cup Ran(\theta) \cup V^*$. Definimos

$$\delta_i^{11} = \begin{cases} \{x_i/t_i\{z_1/y_1^1\}\} & \text{si } i < i_1^1 \\ \{y_1^1/t_{i_1^1}\} & \text{si } i = i_1^1 \\ \delta_i & \text{si } i > i_1^1 \end{cases}$$

Se tiene que $\delta_1^{11} \ldots \delta_k^{11} \in Desc(\theta)$ ya que para todo $i \in \{1, \ldots, k\}$, $Dom(\delta_i^{11}) \subseteq Ran(\delta_i^{11})$ y además, para todo $u \in Dom(\theta)$

$$\begin{aligned} & u\delta_1^{11} \ldots \delta_{i_1^1-1}^{11} \delta_{i_1^1}^{11} \delta_{i_1^1+1}^{11} \ldots \delta_k^{11} \\ = \; & (u\delta_1 \ldots \delta_{i_1^1-1})\{z_1/y_1^1\}\{y_1^1/t_{i_1^1}\}\delta_{i_1^1+1}^{11} \ldots \delta_k^{11} \\ = \; & u\delta_1 \ldots \delta_{i_1^1} \delta_{i_1^1+1} \ldots \delta_k^{11} \end{aligned}$$

Por tanto $\delta_1^{11} \ldots \delta_k^{11} \in Desc(\theta)$. Puede ocurrir que $z\delta_1^{11} \ldots \delta_k^{11} \neq z_1$. En este caso, iteramos el proceso y tomamos y_1^2 una variable que no ocurra en $Dom(\theta) \cup Ran(\theta) \cup V^* \cup \{y_1^1\}$ y sea i_1^2 el menor índice i tal que $z_1 \in Dom(\delta_i^1)$. Obviamente $i_1^1 < i_2^1$. De manera análoga al paso anterior obtenemos $\delta_1^{12} \ldots \delta_k^{12} \in Desc(\theta)$. Iterando el proceso obtenemos una sucesión de índices $i_1^1, i_1^2, i_1^3, \ldots$ Esta sucesión no puede ser infinita, luego, tras n pasos obtenemos $\delta_1^{1n} \ldots \delta_k^{1n} \in Desc(\theta)$ tal que para todo $k \in \{1, \ldots, n\}$, $z_1 \notin Dom(\delta_i^{1n})$.

Sea $V_1^* = V^* - \{z_1\}$. Si existe $z_2 \in V_1^*$ tal que $z_2\delta_1^{1n} \ldots \delta_k^{1n} \neq z_2$, iteramos todo el proceso tomando ahora las variables y_2^j de manera que no pertenezcan a

$$Dom(\theta) \cup Ran(\theta) \cup V^* \cup \{y_1^1, \ldots, y_1^n\} \cup \{y_2^1, \ldots, y_2^{j-1}\}$$

Puesto que V^* es finito, el proceso termina y finalmente obtenemos $\delta_1^{mn}, \ldots, \delta_k^{mn} \in Desc(\theta)$ con $long(\delta_1^{mn}, \ldots, \delta_k^{mn}) = k$ tal que para toda $z \in V^*$, $z\delta_1^{mn}, \ldots, \delta_k^{mn} = z$. ⊣

A continuación enunciamos y demostramos la proposición por la que obtenemos la otra desigualdad.

Proposición 4.32 *Sean C y D dos cláusulas tales que $C \succeq D$ y $C \not\subseteq D$. Entonces*

$$dc(C, D) \leq min\{Peso(\theta) \mid C\theta \subseteq D\}$$

Demostración:
Sea θ una sustitución tal que $C\theta \subseteq D$. Puesto que $C \not\subseteq D$, la sustitución θ no

es vacía. Sea $k = Peso(\theta)$. Entonces existe una sucesión $\sigma_1 \ldots \sigma_k \in Desc(\theta)$ de longitud k tal que $C\sigma_1 \ldots \sigma_k \subseteq D$.

Sean $C_0 = C$ y para todo $i \in \{1, \ldots, k\}$ sea $C_i = C_{i-1}\sigma_i$. Por el lema 4.19 sabemos que para $j \in \{1, \ldots, k-1\}$ existe un OAS tal que $C_j\{\delta_j/x_j\} = C_{j-1}$. Veamos que también existe un OAS $\{\Delta_k/x_k$ tal que $D\{\Delta_k/x_k\} = C_{k-1}$. Tenemos que $C_{k-1}\sigma_k = C_k$ y $C_k \subseteq D$. Aplicando de nuevo el lema 4.19 encontramos un operador $\{\Delta_k/x_k\}$ tal que $C_k\{\Delta_k/x_k\} = C_{k-1}$ y además la asignación de posiciones Δ_k verifica que $\Delta_k(L) = \emptyset$ si $L \notin C_k$. Por tanto ese OAS verifica que

$$\begin{aligned} & D\{\Delta_k/x_k\} \\ = \; & C\{\Delta_k/x_k\} \cup [D - C]\{\Delta_k/x_k\} \\ = \; & C\{\Delta_k/x_k\} \cup \emptyset \\ = \; & C_{k-1} \end{aligned}$$

Por tanto, si θ es una sustitución no vacía tal que $C\theta \subseteq D$ y $Peso(\theta) = k$ entonces existe $\mathcal{C} \in \mathbf{L}(C, D)$ con $|\mathcal{C}| = k$ por tanto, si $C \not\subseteq D$

$$\{Peso(\theta) \mid C\theta \subseteq D\} \subseteq \{|\mathcal{C}| : \mathcal{C} \in \mathbf{L}(C, D)\}$$

luego

$$dc(C, D) = min\{|\mathcal{C}| : \mathcal{C} \in \mathbf{L}(C, D)\} \leq min\{Peso(\theta) \mid C\theta D\}$$

⊣

En consecuencia, si tenemos en cuenta que la quasi-distancia de una cláusula a sí misma es cero y que si $C \neq D$ y $C \subseteq D$ entonces podemos obtener C aplicando un único operador, obtenemos el siguiente corolario:

Corolario 4.33

$$dc(C, D) = \begin{cases} 0 & si\ C = D \\ 1 & si\ C \neq D\ y\ C \subseteq D \\ min\{Peso(\theta) \mid C\theta \subseteq D\} & si\ C \succeq D\ y\ C \not\subseteq D \\ \infty & si\ C \not\succeq D \end{cases}$$

Dadas C y D las distintas condiciones en la definición por partes del corolario son decidibles. Además, si $C \succeq D$ sólo existen una cantidad finita de sustituciones θ tales que $C\theta \subseteq D$. Luego, para conocer de manera efectiva la distancia de subsunción sólo queda conocer el peso de una sustitución dada. Dedicaremos a

ello el resto de la sección.

En primer lugar veamos una interesante propiedad sobre las descomposiciones minimales. Sólo debemos tener en cuenta que si θ es una sustitución no vacía y $\delta_1 \ldots \delta_n \in Desc(\theta)$ es tal que $long(\delta_1 \ldots \delta_n) = Peso(\theta)$, en virtud del lema 4.31 siempre existe $\sigma_1, \ldots, \sigma_n$ con

$$long(\sigma_1, \ldots, \sigma_n) = long(\delta_1 \ldots \delta_n) = Peso(\theta)$$

tal que si V^* es un conjunto de variables finito disjunto con $Dom(\theta)$, para toda $z \in Var^*$, $z\sigma_1, \ldots, \sigma_n = z$. Por tanto no hay pérdida de generalidad en considerar una descomposición $\sigma_1, \ldots, \sigma_n$ de la sustitución θ, con $long(\sigma_1, \ldots, \sigma_n) = Peso(\theta)$ tal que $z \in V^*$, $z\sigma_1, \ldots, \sigma_n) = z$ con V^* apropiado.

Teorema 4.34 *Sea θ una sustitución no vacía y sea $\sigma_1 \ldots \sigma_n \in Desc(\theta)$ tal que $Peso(\theta) = long(\sigma_1 \ldots \sigma_n) = n$. Sea θ^* la sustitución $\sigma_1 \ldots \sigma_{n-1} \restriction_{Dom(\theta)}$. Entonces $Peso(\theta^*) = n - 1$*

Demostración:
Para todo $i \in \{1, \ldots, n-1\}$, $\sigma_i = \{x_i/t_i\}$ con $x_i \notin Var(t_i)$ y para todo $z \in Dom(\theta^*)$ se tiene que $z\theta^* = z\sigma_1 \ldots \sigma_{n-1}$. Por tanto $\sigma_1 \ldots \sigma_{n-1} \in Desc(\theta^*)$ y $long(\sigma_1 \ldots \sigma_{n-1}) = n - 1$. De ahí que $Peso(\theta^*) \leq n - 1$.

Veamos a continuación que $Peso(\theta^*) \geq n - 1$. Lo vemos por reducción al absurdo. Supongamos que $Peso(\theta^*) = k < n - 1$. Sin pérdida de generalidad podemos considerar $\delta_1 \ldots \delta_k \in Desc(\theta^*)$ con $long(\delta_1 \ldots \delta_k) = k$ tal que para toda $z \in Dom(\theta) - Dom(\theta^*)$, $z\delta_1 \ldots \delta_k = z$. Puesto que $\sigma_n = \{x_n/t_n\}$ con $x_n \notin Var(t_n)$, para ver que $\delta_1 \ldots \delta_k\sigma_n \in Desc(\theta)$ sólo hay que comprobar que para toda $z \in Dom(\theta)$ se verifica que $z\theta = z\delta_1 \ldots \delta_k\sigma_n$.

- Si $z \in Dom(\theta^*)$. Por definición $Dom(\theta^*) \subseteq Dom(\theta)$, luego

$$z\delta_1 \ldots \delta_k\sigma_n = z\theta^*\sigma_n = z\sigma_1 \ldots \sigma_{n-1}\sigma_n = z\theta$$

- Tomemos ahora $z \in Dom(\theta) - Dom(\theta^*)$. Por definición de θ^*, $z\sigma_1 \ldots \sigma_{n-1} = z$ (en otro caso, $z \in Dom(\theta^*)$) y por otra parte tenemos que $z \in Dom(\theta)$ luego $z\theta \neq z$, por tanto

$$z\sigma_1 \ldots \sigma_{n-1}\sigma_n = z\sigma_n = z\{x_n/t_n\} \neq z$$

luego $z = x_n$, por tanto, según hemos elegido $\delta_1 \ldots, \delta_k$, se tiene

$$z\delta_1 \ldots \delta_k\sigma_n = z\sigma_n = t_n = z\theta$$

Por tanto tenemos que $\delta_1 \ldots \delta_k\sigma_n \in Desc(\theta)$ verificando $long(\delta_1 \ldots \delta_k\sigma_n) = k + 1 < n = Peso(\theta)$. Contradicción. ⊣

Entrada: Una sustitución θ no vacía
Sea $\theta_0 = \theta$ y $U_O = Dom(\theta) \cup Ran(\theta)$
Paso 1:
Si θ_i es la sustitución vacía
Entonces para

En otro caso: representamos $\theta_i = \{x_1/t_1, \ldots, x_n/t_n\}$ e ir al Paso 2.
Paso 2:
Si existe $x_j \in Dom(\theta_i)$ tal que $x_j \notin Ran(\theta_i)$
Entonces para toda $k \in \{1, \ldots, j-1, j+1, \ldots, n\}$ sea t_k^* un término tal que al aplicarle $\{x_j/t_j\}$ obtengamos t_k,esto es, $t^k = t_k^*\{x_j/t_j\}$
Sean.
$\theta_{i+1} = \{x_1/t_1^*, \ldots, x_{j-1}/t_{j-1}^*, x_{j+1}/t_{j+1}^*, \ldots, x_n/t_n^*\}$
$\sigma_{i+1} = \{x_j/t_j\}$
$U_{i+1} = U_i$
asignamos a i el valor $i+1$ y volvemos al Paso 1.

En otro caso: Ir al Paso 3.
Paso 3:
En este caso sea z_i una variable que no pertenezca a U_i y sea
$U_{i+1} = U_i \cup \{z_i\}$
sea $j \in \{1, \ldots, n\}$ y sea T un subtérmino de t_j tal que T no es una variable de U_{i+1}. Entonces, para toda $k \in \{1, \ldots, n\}$ sea t_k^* un término
tal que al aplicarle $\{z/T\}$ obtengamos t_k,esto es, $t^k = t_k^*\{z/T\}$. Sean
$\theta_{i+1} = \{x_1/t_1^*, \ldots, x_n/t_n^*\}$
$\sigma_{i+1} = \{z/T\}$
asignamos a i el valor $i+1$ y volvemos al Paso 1.

Figura 4.1: Algoritmo para el cálculo de distancia de subsunción

Antes de pasar a ver cómo podemos encontrar una descomposición mínima, veamos otro lema que relaciona el dominio de θ con el dominio de las sustituciones elementales de las descomposiciones.

Lema 4.35 *Sea θ una sustitución no vacía y sea $\delta_1 \ldots \delta_k \in Desc(\theta)$ tal que*

long$(\delta_1 \ldots \delta_k) = k$. *Entonces existe* $\sigma_1 \ldots \sigma_k \in Desc(\theta)$ *con* *long*$(\sigma_1 \ldots \sigma_k) = k$ *y* $\sigma_i = \{x_i/t_i\}$ *tal que para todo* $z \in Dom(\theta)$ *existe un único índice* i *tal que* $x_i = z$.

Demostración:
Trivialmente se tiene que si $\sigma_1 \ldots \sigma_k \in Desc(\theta)$ y $z \in Dom(\theta)$ existe un $i \in \{1, \ldots, k\}$ tal que $x_i = z$. Tenemos entonces que probar que existe una descomposición $\sigma_1 \ldots \sigma_k$ donde para cada $z \in Dom(\theta)$ existe un *único* índice i con $x_i = z$.

Sea $\delta_1 \ldots \delta_k \in Desc(\theta)$ con $\delta_j = \{y_j/s_j\}$ y sea $z^* \in Dom(\theta)$ tal que existen $i, j \in \{1, \ldots, k\}$ con $i \neq j$ tales que $y_i = y_j = z^*$. Sea j_0 el mayor índice tal que $y_{j_0} = z^*$ y sea j_1 es mayor índice de $\{1, \ldots, j_0 - 1, j_0 + 1, \ldots, k\}$ tal que $y_{j_1} = z^*$. Sea x_0 una variable que no ocurra en θ ni en ningún δ_i. Definimos

$$\delta_i^1 = \begin{cases} \delta_i & \text{si } i \leq j_1 \\ \{y_1/s_i\{z^*/x_0\}\} & \text{si } j_1 < i < j_0 \\ \{x_0/s_{j_0}\} & \text{si } i = j_0 \\ \delta_i & \text{si } i > j_0 \end{cases}$$

Cada sustitución elemental δ_k^1 verifica que $Dom(\delta_k^1) \not\subseteq Ran(\delta_k^1)$ y además, para todo $z \in Dom(\theta)$

$$\begin{aligned} & z\delta_1^1 \ldots \delta_k^1 \\ = \; & z\delta_1 \ldots \delta_{j_0-1}\{z/x_0\}\{x_0/s_{j_0}\}\delta_{j_0+1} \ldots \delta_k \\ = \; & z\delta_1 \ldots \delta_{j_0-1}\{x_{j_0}/s_{j_0}\}\delta_{j_0+1} \ldots \delta_k \end{aligned}$$

luego $\delta_1^1 \ldots \delta_k^1 \in Desc(\theta)$, *long*$(\delta_1^1 \ldots \delta_k^1) = k$ y z^* ha disminuido en una unidad sus ocurrencias en los dominios de las sustituciones elementales. Iterando el proceso obtenemos una descomposición en la que cada variable del dominio ocurra una única vez en el dominio de una sustitución elemental. ⊣

La figura 4.1 representa el algoritmo que vamos a seguir para obtener la descomposición de una sustitución. Se trata de un algoritmo no determinista. Cada una de las posibles elecciones nos lleva a una descomposición. Para calcular el peso tomaremos la descomposición más corta entre todas las posibles.

Proposición 4.36 *El* ALGORITMO *termina*

Demostración:
Para ver que el algoritmo termina sólo hay que tener en cuenta que si x_j/t_j es

un enlace de θ_i y x_j/t_j^* es un enlace de θ_{i+1} entonces $Pos(t_j^*) \subseteq Pos(t_j)$ ya que en ambos pasos del algoritmo existe una sustitución γ tal que $t^j = t_j^*\gamma$. Teniendo esto en cuenta definimos

$$A_i = \left\{ \langle x, f, p\rangle \mid \begin{array}{l} x/t \text{ es un enlace de } \theta_i,\ p \in Pos(t) \\ \text{y } f \text{ es el símbolo asociado a } t/p \end{array} \right\}$$

Sea $K_i = A_i \cap A_0$. Para ver que el algoritmo termina vamos a ver que $|K_{i+1}| < |K_i|$. Sea $\theta_i = \{x_1/t_1, \ldots, x_n/t_n\}$ y supongamos que existe $x_j \in Dom(\theta_i)$ tal que $x_j \notin Ran(\theta_i)$ y sea

$$\theta_{i+1} = \{x_1/t_1^*, \ldots, x_{j-1}/t_{j-1}^*, x_{j+1}/t_{j+1}^*, \ldots, x_n/t_n^*\}$$

Puesto que para todo j, $Pos(t_j^*) \subseteq Pos(t_j)$, tenemos que $K_{i+1} \subseteq K_i$ y además, si $\sigma_{i+1} = \{x_{j_0}/t_{j_0}\}$ entonces en θ_{i+1} no hay ningún enlace x_{j_0}/s_{j_0}, luego $K_{i+1} \subseteq K_i$. En el caso en que $Dom(\theta_i) \subseteq Ran(\theta_i)$, aplicamos el paso 2. En este caso también tenemos que para todo j, $Pos(t_j^*) \subseteq Pos(t_j)$, tenemos que $K_{i+1} \subseteq K_i$ y si T es un subtérmino de t_{j_0} que no sea variable nueva y $u \in Pos(t_j)$ verificando $t_{j_0}/u = T$ entonces $\langle x_{j_0}, f, u\rangle \in K_i$ y $\langle x_{j_0}, f, u\rangle \notin K_{i+1}$. ⊣

El ALGORITMO es un algoritmo no determinista que recibe una sustitución y calcula un conjunto de descomposiciones. Nuestro objetivo ahora es probar que una de ellas es de longitud mínima. De este modo completaríamos el cálculo efectivo de la quasi-distancia de subsunción. Lo vemos en el teorema final. Antes, necesitamos un lema.

Lema 4.37 *Si toda sucesión $\sigma_1 \ldots \sigma_n \in Desc(\theta)$ con $Peso(\theta) = long(\sigma_1 \ldots \sigma_n) = n$ verifica que $x_n \notin Dom(\theta)$ (donde $\sigma_i = \{x_i/t_i\}$) entonces $Dom(\theta) \subseteq Ran(\theta)$.*

Demostración:
Lo vemos por reducción al absurdo. Supongamos $Dom(\theta) \not\subseteq Ran(\theta)$ y sea $\sigma_1\sigma_n \in Desc(\theta)$ con $Peso(\sigma_1 \ldots \sigma_n) = long(\sigma_1 \ldots \sigma_n) = n$. No hay pérdida de generalidad si suponemos que par toda variable el dominio z existe un único índice i tal que $z = x_i$.

Puesto que $Dom(\theta) \not\subseteq Ran(\theta)$ existe $z \in Dom(\theta)$ tal que $z \notin Ran(\theta)$. Sea i_0 el único índice tal que $z = x_{i_0}$ y consideremos las sustituciones $\sigma_{i_0} = \{z/t_{i_0}\}$ y

$\sigma_{i_0+1} = \{x_{i_0+1}/t_{i_0+1}\}$. Su composición es

$$\sigma_{i_0}\sigma_{i_0+1} = \{z/t_{i_0}\sigma_{i_0+1}, x_{i_0+1}/t_{i_0+1}\}$$

Sea $\delta_{i_0} = \{x_{i_0+1}/t_{i_0+1}\}$ y $\delta_{i_0+1} = \{z/t_{i_0}\sigma_{i_0+1}\}$. Para ver que $\delta_{i_0}\delta_{i_0+1} = \sigma_{i_0}\sigma_{i_0+1}$ basta ver que $x \notin Var(t_{i_0+1})$, pero esto se tiene. ⊣

Lema 4.38 *Si existe una sucesión de sustituciones $\theta_0, \theta_1, \ldots, \theta_n$ con θ_n la sustitución vacía generada mediante los distintos paso del* ALGORITMO *a partir de $\theta = \theta_0$ verificando que para todo i, $Dom(\theta_i) \not\subseteq Ran(\theta_i)$ entonces $Peso(\theta) = |\theta|$.*

Demostración:
Trivial, ya que en este caso siempre podemos aplicar el Paso 1 del algoritmo, con lo que en cada iteración $|\theta_{i-1}| = |\theta| - 1$. ⊣

Nota 4.39 Usaremos el resultado del lema anterior en sentido contrario al que lo hemos enunciado, esto es, consideraremos que si $Peso(\theta) \neq |\theta|$, entonces para toda sucesión de sustituciones $\theta_0, \theta_1, \ldots, \theta_n$ con θ_n la sustitución vacía generada mediante los distintos paso del ALGORITMO a partir de $\theta = \theta_0$, existe i tal que $Dom(\theta_i) \subseteq Ran(\theta_i)$.

Teorema 4.40 *Sea θ una sustitución no vacía y $Peso(\theta) = n$. Entonces, en un paso el* ALGORITMO *genera dos sustituciones θ^* y σ verificando que*

(1) Si $\sigma = \{x/t\}$, x no ocurre en t.

(2) Para toda $z \in Dom(\theta)$, $z\theta = z\theta^*\sigma$.

(3) $Peso(\theta^*) = n - 1$

Demostración:
Los apartados (1) y (2) se tienen trivialmente. Veamos el apartado(3).

Por el teorema 4.34 si $\sigma_1 \ldots \sigma_n \in Desc(\theta)$ y $Peso(\theta) = long(\sigma_1 \ldots \sigma_n) = n$, entonces $Peso(\sigma_1 \ldots \sigma_{n-1} \restriction_{Dom(\theta)}) = n - 1$. Para ver que se tiene el resultado tenemos que comprobar que existe una descomposición $\sigma_1 \ldots \sigma_n \in Desc(\theta)$ y $Peso(\theta) = long(\sigma_1 \ldots \sigma_n) = n$, tal que el algoritmo encuentra $\sigma = \sigma_n$ y $\theta^* = \sigma_1 \ldots \sigma_{n-1} \restriction_{Dom(\theta)}$.

Sea $\sigma_1 \ldots \sigma_n \in Desc(\theta)$ y $Peso(\theta) = long(\sigma_1 \ldots \sigma_n) = n$ con $\sigma_i = \{x_n/t_n\}$. Sin pérdida de generalidad podemos suponer que para todo $z \in Dom(\theta)$ existe un único índice i tal que $x_i = z$. Distinguimos dos casos:

Caso 1: $x_n \in Dom(\theta)$

Veamos en primer lugar que si $x_n \in Dom(\theta)$ entonces $x_n \notin Ran(\theta)$. Nótese que por definición de $Desc(\theta)$, para todo i, $x_i \notin Var(t_i)$, en particular, $x_n \notin Var(t_n)$. Sea $z \in Dom(\theta)$.

- Si $x_n \in z\sigma_1 \ldots \sigma_{n-1}$ entonces $x_n \notin (z\sigma_1 \ldots \sigma_{n-1})\sigma_n$ ya que $x_n \notin Var(t_n)$ y $z\theta = z\sigma_1 \ldots \sigma_n$
- Si $x_n \notin z\sigma_1 \ldots \sigma_{n-1}$ entonces $x_n \notin z\sigma_1 \ldots \sigma_{n-1} = z\sigma_1 \ldots \sigma_{n-1}\sigma_n = z\theta$.

Por consiguiente, tenemos que $x_n \in Dom(\theta)$ y $x_n \notin Ran(\theta)$. El Paso 2 del ALGORITMO encuentra $x_n \in Dom(\theta)$. Sabemos que para todo $z \in Dom(\theta)$ existe un único índice $i \in \{1, \ldots, n\}$ tal que $x_i = z$, por tanto, para todo $j < n$ $x_j \neq x_n$. De ahí que

$$x_n\theta = x_n\sigma_1 \ldots \sigma_n = x_n\sigma_n = t_n$$

Por tanto $\sigma = \sigma_n = \{x_n/t_n\}$.

Para completar la tesis del teorema, en este caso, tenemos que probar que si θ_{i+1} es una sustitución generada en el Paso 2 a partir de $\theta_i = \theta$ entonces $\sigma_1 \ldots \sigma_{n-1} \restriction_{Dom(\theta)}$.

Se tiene que

$$Dom(\theta_{i+1} = Dom(\sigma_1 \ldots \sigma_{n-1} \restriction_{Dom(\theta)}) = \{x_1, \ldots, x_{n-1}\}$$

luego, para ver que ambas sustituciones coinciden, basta ver que para todo $k \in \{1, \ldots, n-1\}$

$$x_k\sigma_1 \ldots \sigma_{n-1} \restriction_{Dom(\theta)} = x_k\theta_{i+1}$$

sabemos que $(x_k\sigma_1 \ldots \sigma_{n-1} \restriction_{Dom(\theta)})\{x_n/t_n\} = t_k$. Por tanto $x_k\sigma_1 \ldots \sigma_{n-1} \restriction_{Dom(\theta)}$ es un determinado t_k^* y el algoritmo lo encuentra.

Caso 2: $x_n \notin Dom(\theta)$

Aquí tenemos que contemplar dos posibilidades:

Caso 2.1: $Dom(\theta) \not\subseteq Ran(\theta)$

En virtud del lema 4.37 si $Dom(\theta) \not\subseteq Ran(\theta)$, entonces existe $\delta_1 \ldots \delta_n \in Desc(\theta)$ con $Peso(\theta) = long(\delta_1 \ldots \delta_n) = n$ tal que $\delta_n = \{y_n/s_n\}$ verificando $y_n \in Dom(\theta)$. En este caso, como hemos visto en el caso anterior, el algoritmo encuentra un θ^*

y σ relacionados con $\delta_1 \ldots \delta_n$.

Caso 2.2: $Dom(\theta) \subseteq Ran(\theta)$

En este caso estamos en el paso 3 del ALGORITMO. Sea z la variable *nueva* y sean, para todo $i \in \{1, \ldots, n-1\}$, $\gamma_i = \{x_i/t_i\{x_n/z\}\}$ y

$\theta^* = \gamma_1 \ldots \gamma_{n-1} \restriction_{Dom(\theta)}$

$\sigma = \{z/t_n\}$

Para concluir el teorema queda por ver:

- $\gamma_1 \ldots \gamma_{n-1}\sigma \in Desc(\theta)$
  En primer lugar, $x_i \notin Var(t_i\{x_n/z\})$, para todo $i \in \{1, \ldots, n-1\}$ y $z \notin Var(t_n)$. Por otra parte, sea $u \in Dom(\theta)$.

  - $u \neq x_n$

    $$u\gamma_1 \ldots \gamma_{n-1}\{z/t_n\} = u\sigma_1 \ldots \sigma_{n-1}\{x_n/z\}\{z/t_n\} = u\sigma_1 \ldots \sigma_n = U\theta$$

  - $u = x_n$

    $$\begin{aligned} & x_n\gamma_1 \ldots \gamma_{n-1}\{z/t_n\} \\ = \; & x_n\sigma_1 \ldots \sigma_{n-1}\{x_n/z\}\{z/t_n\} \\ = \; & x_n\{x_n/z\}\{z/t_n\} = t_n \\ = \; & \overset{(*)}{=} x_n\sigma_1 \ldots \sigma_n \\ = \; & x_n\theta \end{aligned}$$

    donde $(*)$ se tiene porque si $j < n$, entonces $x_n \neq x_j$.

- Existe $z \in Dom(\theta)$ tal que t_n es un subtérmino de $z\theta$.
  Trivialmente se comprueba que si t_n no es subtérmino de ningún $z\theta$ entonces $\sigma_1 \ldots \sigma_{n-1} \in Desc(\theta)$ y $Peso(\theta) < n$. Contradicción

⊣

A partir del teorema se tiene de manera inmediata el siguiente corolario.

Corolario 4.41 *Sea θ una sustitución no vacía. Entonces $Peso(\theta)$ coincide con la longitud de la descomposición más corta obtenida por el* ALGORITMO 1.

Ejemplo 4.42 Para todo $n \geq 0$, consideremos las cláusulas

$$\begin{aligned} C_n &\equiv sum(s^{n+1}(x_1), s^n(y_1), s^{n+1}(z_1)) \leftarrow sum(s^n(x_1), s^n(y_1), s^n(z_1)) \\ D_n &\equiv sum(s^{2n+1}(x_2), s^{2n}(y_2), s^{2n+1}(z_2)) \leftarrow sum(s^{2n}(x_2), s^{2n}(y_2), s^{2n}(z_2)) \end{aligned}$$

N	$dc(C_n, D_n)$		$d_h(C_n, D_n)$	
	Seg	Q–dist	Seg	Dist
64	0.02	3	0.11	$\sim 2.7\,10^{-20}$
128	0.06	3	0.21	$\sim 1.4\,10^{-39}$
256	0.1	3	0.43	$\sim 4.3\,10^{-78}$
512	0.26	3	0.93	$\sim 3.7\,10^{-155}$
1024	0.67	3	2.03	$\sim 2.7\,10^{-309}$

Figura 4.2: Tabla comparativa

y la sustitución $\theta_n = \{x_1/s^n(x_2), y_1/s^n(y_2), x_3/s^n(y_3)\}$. Entonces $C_n\theta_n = D_n$ para todo n y de ahí, $C_n \succeq D_n$. La tabla 4.2 muestra los valores obtenidos para la quasi-métrica $dc(C_n, D_n)$ y la distancia de Hausdorff $d_h(C_n, D_n)$ para varios valores de N así como el tiempo de computación en un Pentium III 800 Mhz. en una implementación en SWI-Prolog 4.0.11. Se prueba fácilmente que para todo $n \geq 0$, $dc(C_n, D_n) = 3$. Si usamos la métrica de Hausdorff d_h basada en d_{nc} tenemos que, para todo $n \geq 0$

$$d_h(C_n, D_n) = \frac{1}{2^{n+1}}$$

que tiende a cero a pesar de que la relación de subsunción se verifica para todo n.

4.10 Cotas

En la sección anterior hemos visto un método para computar dc, pero decidir si dos cláusulas están relacionadas por subsunción es un problema NP-completo [GJ79], así que, desde un punto de vista práctico necesitamos una estimación rápida de la quasi-métrica antes de decidir la subsunción.

Terminamos este capítulo con un resultado que determina una cota superior e inferior para dc.

Teorema 4.43 *Sean C_1 y C_2 dos cláusulas tales que $C_1 \not\subseteq C_2$. Si $C_1 \succeq C_2$ entonces*

$$|Var(C_1) - Var(C_2)| \leq dc(C_2, C_1) \leq min\{2 \cdot |Var(C_1)|\,,\ |Var(C_1)| + |Var(C_2)|\}$$

Demostración:
Para cada θ tal que $C_1\theta \subseteq C_2$, θ tiene al menos $|Var(C_1) - Var(C_2)|$ enlaces y necesitamos al menos un OAS para cada enlace, de ahí la primera desigualdad. Para la segunda, si $C_1\theta \subseteq C_2$ entonces podemos encontrar n sustituciones $\sigma_1, \ldots, \sigma_n$ con $\sigma_1 = \{x_i/t_i\}$ y $x_i \notin Var(t_i)$ tales que $C_1\sigma_1 \ldots \sigma_2 \subseteq C_2$ verificando $n = |\theta| + |Ran(\theta) \cap Dom(\theta)|$. La desigualdad se tiene puesto que $Ran(\theta) \subseteq Var(C_2)$, $Dom(\theta) \subseteq Var(C_1)$ y $|\theta| \leq Var(C_1)$. ⊣

El siguiente ejemplo muestra que estas cotas no pueden mejorarse.

Ejemplo 4.44 Consideremos las cláusulas

$$C_1 = \{p(x_1, x_2)\}$$
$$C_2 = \{p(a, b)\}$$
$$C_3 = \{p(f(x_1, x_2), f(x_2, x_1))\}$$

entonces se tiene que

$$dc(C_2, C_1) = |Var(C_1) - Var(C_2)| = 2$$
$$dc(C_3, C_1) = min\{2 \cdot |Var(C_1)|, |Var(C_1)| + |Var(C_2)|\} = 4$$

Capítulo 5

Operadores clausales y el orden de consecuencia

En este capítulo estudiamos la relación entre los operadores clausales y el orden de consecuencia. En el capítulo 4 veíamos que si dos cláusulas C y D estaban ligadas por la relación de subsunción, $C \succeq D$, entonces podíamos encontrar una serie de operadores clausales de un determinado tipo (OAS) que *generalizaban* D para obtener C. Estudiamos ahora el orden de consecuencia $\models$ definido como una relación binaria en el conjunto de cláusulas $\mathbb{C}$. Según vimos en el teorema 3.22, dadas dos cláusulas C y D siempre podemos encontrar un operador $\phi \in \mathbb{OP}^*_{\Xi}$ tal que $C\phi = D$. En particular, esto ocurre cuando $C \models D$. Vamos a estudiar cuáles son esos operadores de $\mathbb{OP}^*_{\Xi}$ que nos permiten obtener C a partir de D cuando $C \vDash D$.

Para ello, nos apoyaremos en un corolario del teorema de subsunción (5.7) que hemos llamado corolario de interpolación (5.12) mediante el cual caracterizamos la relación $\models$ definida en $\mathbb{C}$ con la composición de dos relaciones binarias: La relación de derivación $\vdash_r$ (Definición 5.5) y la relación de subsunción $\succeq$.

También veremos que con un tipo especial de operadores que llamaremos *operadores de inversión sesgados (OIS)* podemos pasar de la cláusula D a la cláusula C cuando C es más general que D en el orden de derivación $\vdash_r$. Además, como ocurría con el orden de subsunción, para ello no necesitamos los operadores extendidos (Def. 3.18), sino que basta buscar en $\mathbb{OP}^*$. Combinando adecuadamente los operadores de aprendizaje para subsunción (OAS) y los operadores de inversión sesgados (OIS) podremos invertir la relación de consecuencia $\models$. Nuestro objetivo es probar el siguiente resultado

Teorema 5.1 *Si $C_1 \models C_2$ y C_2 no es una tautología entonces existe $\phi \in \mathbb{OP}^*$ tal que $C_1 = C_2\phi$*

Los operadores necesarios para invertir el orden de consecuencia (OIS) tienen propiedades distintas a los que usamos para invertir el orden de subsunción (OAS). Sin embargo, ambos órdenes, subsunción y consecuencia, están relacionados por el lema de Gottlob (Lema 5.8).

Tomando como punto de partida ambos tipos de operadores, OAS y OIS, nos planteamos la búsqueda de un nuevo conjunto de operadores capaces de generar tanto operadores de inversión sesgados (OIS) como operadores de aprendizaje para subsunción (OAS): Los *operadores de generalización minimales (OGM)*, a los que dedicaremos la parte final del capítulo. Estos operadores son capaces de generar tanto los Operadores de Aprendizaje para Subsunción como los Operadores de Inversión Sesgados y representan las unidades mínimas que permiten operar con ambos órdenes.

En este capítulo no consideraremos el caso de generalización de la relación de consecuencia de tautologías.

5.1 Motivación

El punto de partida para poder encontrar operadores clausales que generalicen la cláusula D para obtener C en el caso en que $C \models D$ es el teorema de subsunción (teorema 5.7). Ese teorema ha tenido distintas formulaciones a lo largo de la historia[1]. Nosotros seguimos a Nienhuys-Cheng y de Wolf en [NCdW97]. Antes de presentar su enunciado, recordamos algunas definiciones y fijamos las notaciones correspondientes.

Definición 5.2 *Sea C una cláusula y $L_1, \ldots, L_n$ $(n \geq 1)$ literales unificables de C y sea θ un umg de $L_1, \ldots, L_n$. Entonces la cláusula $C\theta$ se llama* factor *de C.*

Definición 5.3 *Sean $C_1 = \{L_1, \ldots, L_m\}$ y $C_2 = \{M_1, \ldots, M_n\}$ dos cláusulas que no tienen variables en común. Si la sustitución θ es un umg de L_i y $\neg M_j$ entonces la cláusula*

$$(\{L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_m\} \cup \{M_1, \ldots, M_{j-1}, M_{j+1}, \ldots, M_n\})\theta$$

[1] Ver [NCdW96].

se llama resolvente binaria *de* C_1 *y* C_2. *Si* C_1 *y* C_2 *tuvieran variables en común, podemos usar una variante* C_2' *de* C_2 *de manera que* C_1 *y* C_2' *no tuvieran variables en común.*

Definición 5.4 *Sean* C_1 *y* C_2 *dos cláusulas. Una resolvente* C *de* C_1 *y* C_2 *es una resolvente binaria de un factor de* C_1 *y un factor de* C_2.

Definición 5.5 *Sea* Σ *un conjunto de cláusulas y* C *una cláusula. Una derivación de* C *a partir de* Σ *es una sucesión finita de cláusulas* $R_1, \ldots, R_n$ *donde* $R_n = C$ *tal que cada* R_i *o bien pertenece a* Σ *o bien es una resolvente de dos cláusulas en* $\{R_1, \ldots, R_{i-1}\}$. *En este caso diremos que* R_n *ha sido obtenida en una derivación de longitud* n. *Si existe una derivación de* C *a partir de* Σ *lo representaremos por* $\Sigma \vdash_r C$.

Definición 5.6 *Sea* Σ *un conjunto de cláusulas y* C *una cláusula. Se dice que existe una deducción de* C *a partir de* Σ *y lo representamos por* $\Sigma \vdash_d C$ *si* C *es una tautología o si existe una cláusula* D *tal que* $\Sigma \vdash_r D$ *y* $D \succeq C$.

Teorema 5.7 (El teorema de subsunción, 5.17 en [NCdW97]) *Sea* Σ *un conjunto de cláusulas y* C *una cláusula. Entonces* $\Sigma \models C$ *si y sólo si* $\Sigma \vdash_d C$.

La demostración del teorema 5.1 la podemos obtener a partir del teorema de subsunción, pero resulta más natural obtenerla a partir de uno de sus corolarios más importantes: El lema de Gottlob. Este resultado fue probado por Gottlob en [Got87], aunque como apuntan Nienhuys-Cheng y de Wolf en [NCdW97], puede obtenerse como corolario del teorema 5.7.

Lema 5.8 (Lema de Gottlob) *Sean* C *y* D *dos cláusulas no tautológicas y* C^{pos} *y* C^{neg} *los conjuntos de literales (resp.) positivos y negativos de* C. *Análogamente, sean* D^{pos} *y* D^{neg} *los conjuntos (resp.) de literales positivos y negativos de* D. *Si* $C \models D$, *entonces* $C^{pos} \succeq D^{pos}$ *y* $C^{neg} \succeq D^{neg}$.

Ilustramos el lema de Gottlob con un ejemplo clásico.

Ejemplo 5.9 Consideremos las siguientes cláusulas[2]. Sean C y D las cláusulas

$$C = \{lista\_de\_n([x_1|x_2]), \neg lista\_de\_n(x_2), \neg n(x_1)\}$$
$$D = \{lista\_de\_n([z_1, z_2|z_3]), \neg lista\_de\_n(z_3), \neg n(z_1), \neg n(z_2)\}$$

[2]Usaremos la notación Prolog $[X|Y]$ en lugar de la notación $.(X, Y)$.

Es fácil ver que $C \models D$ y que C no subsume a D ya que no existe una sustitución θ tal que $C\theta \subseteq D$. En cambio podemos encontrar dos sustituciones $\delta_1 = \{x_1/z_1, x_2/[z_2|z_3]\}$ y $\delta_2 = \{x_1/z_1, x_2/z_3\}$ tales que

$$\begin{aligned} & C^{pos}\delta_1 \\ = \; & \{lista\_de\_n([x_1|x_2])\}\delta_1 \\ = \; & \{lista\_de\_n([z_1, z_2|z_3])\} \\ \subseteq \; & D^{pos} \end{aligned}$$

$$\begin{aligned} & C^{neg}\delta_2 \\ = \; & \{\neg lista\_de\_n(x_2), \neg n(x_1)\}\delta_2 \\ = \; & \{\neg lista\_de\_n(z_3), \neg n(z_1)\} \\ \subseteq \; & \{\neg lista\_de\_n(z_3), \neg n(z_1), \neg n(z_2)\} \\ = \; & D^{neg} \end{aligned}$$

Luego $C^{pos} \succeq D^{pos}$ y $C^{neg} \succeq D^{neg}$.

Nota 5.10 Es trivial comprobar que el recíproco del lema de Gottlob es falso. Basta tomar

$$C = \{suma(s(x), y, s(z)), \neg suma(x, y, z)\}$$
$$D = \{suma(s(0), y, s(s(0))), \neg suma(0, y, 0)\}$$

y las sustituciones $\theta_p = \{x/0, z/s(s(0))\}$ y $\theta_n = \{x/0, z/0\}$, entonces

$$\begin{aligned} suma(s(x), y, s(z))\theta_p &= suma(s(0), y, s(s(0))) \\ \neg suma(x, y, z)\theta_n &= \neg suma(0, y, 0) \end{aligned}$$

luego $C^{pos} \succeq D^{pos}$ y $C^{neg} \succeq D^{neg}$, pero en cambio $C \not\succeq D$.

Por tanto, tenemos que si $C \succeq D$, entonces existe una *única* sustitución θ tal que $C^{pos}\theta \subseteq D^{pos}$ y $C^{neg}\theta \subseteq D^{neg}$. Si $C \models D$ entonces existen *dos* sustituciones θ_p y θ_n tales que $C^{pos}\theta_p \subseteq D^{pos}$ y $C^{neg}\theta_n \subseteq D^{neg}$. Por consiguiente, para adaptar nuestro estudio del orden de subsunción al orden de implicación tendremos que encontrar operadores clausales *sesgados*, esto es, que transformen únicamente los literales positivos de una cláusula y dejen igual los negativos o viceversa. La existencia de tales operadores la estudiamos en el siguiente lema.

Lema 5.11 (Lema del *dribbling*) *Si C y D son dos cláusulas tales que D sólo tiene literales positivos (resp. negativos) y $C \succeq D$ entonces existe un operador $\phi \in \mathbb{OP}^*$ tal que*

1. $C = D\phi$

2. *Si L es un literal negativo (resp. positivo) entonces $\{L\}\phi = \{L\}$*

Demostración:
En el corolario 4.6 vimos que si $C \succeq D$ entonces podemos encontrar $\phi \in \mathbb{OP}^*$ tal que $C = D\phi$. El operador ϕ era la composición de operadores clausales construidos según los lemas 4.3 y 4.4. Para ver que le resultado se tiene, basta ver que si C y D son dos cláusulas formadas por literales positivos y $\{\Delta/x\}$ es un operador clausal construido según los lemas 4.3 ó 4.4 tal que $D\{\Delta/x\} = C$ entonces $\{L\}\{\Delta/x\} = \{L\}$ para todo literal negativo L.

Si observamos la asignación de posiciones construida según el lema 4.3, vemos que a todo literal que no pertenezca a la cláusula C_2 la imagen por la asignación es $\{\emptyset\}$, en particular, esta es la imagen de todo literal negativo, luego, si L es negativo

$$\{L\}\{\Delta/x\} = L\{\Delta(L)/x\} = L\{\{\emptyset\}/x\} = L$$

De manera análoga, vemos que en la asignación de variables construida según el lema 4.4 tenemos que la imagen de un literal es $\emptyset$ sólo si el literal pertenece a C_2 y no pertenece a C_1. En cualquier otro caso, en particular, si L es negativo, se tiene que $\Delta(L) = \{\emptyset\}$, por tanto, también en este caso

$$\{L\}\{\Delta/x\} = L\{\Delta(L)/x\} = L\{\{\emptyset\}/x\} = L$$

⊣

Podemos demostrar ahora el teorema 5.1:

Demostración [Teorema 5.1]: La demostración se tiene de manera natural a partir de los lemas de Gottlob (5.8) y del lema del *dribbling* (5.11). Si C_1 y C_2 son cláusulas tales que $C_1 \vDash C_2$ y C_2 no es tautología, entonces C_1 tampoco lo es y estamos en las condiciones del lema de Gottlob, luego $C_1^{pos} \succeq C_2^{pos}$ y $C_1^{neg} \succeq C_2^{neg}$. Por el lema del *dribbling* existen dos operadores $\phi_p, \phi_n \in \mathbb{OP}^*$ tales que

- $C_2^{pos}\phi_p = C_1^{pos} \qquad C_2^{neg}\phi_n = C_1^{neg}$
- Para todo literal L positivo, $\{L\}\phi_n = \{L\}$
- Para todo literal L negativo, $\{L\}\phi_p = \{L\}$

De ahí que $C_2^{pos}\phi_n = C_2^{pos}$ y $C_1^{neg}\phi_p = C_1^{neg}$ Por tanto aplicando el lema 3.16 se tiene que

$$\begin{aligned} C_2\phi_n\phi_p &= (C_2^{neg} \cup C_2^{pos})\phi_n\phi_p \\ &= C_2^{neg}\phi_n\phi_p \cup C_2^{pos}\phi_n\phi_p \\ &= C_1^{neg}\phi_p \cup C_2^{pos}\phi_p \\ &= C_1^{neg} \cup C_1^{pos} \\ &= C_1 \end{aligned}$$

⊣

Por tanto ya tenemos que su $C_1 \models C_2$ (y C_2 no es tautología), podemos llegar a C_1 partiendo de C_2 mediante operadores clausales. Estos operadores son operadores de generalización para la relación de consecuencia. Dedicaremos las secciones siguientes a estudiar la naturaleza de estos operadores.

5.2 El orden de derivación

En las sección 5.1 hemos estudiado la relación entre el orden de implicación entre cláusulas y los operadores clausales y hemos visto que si $C \models D$ y D no es tautología entonces podemos encontrar un operador $\phi \in \mathbb{OP}^*$ tal que *generalice* D para obtener C. Los operadores clausales que componen ϕ no son OAS en general, ya que por el lema 4.17 si existiera una cadena de OAS de D a C entonces $C \succeq D$ y esto no ocurre en general. En esta sección nos preguntamos por la naturaleza de los operadores empleados en la generalización de la relación de implicación. Para su estudio consideraremos una versión restringida del teorema de subsunción: El caso en que el conjunto donde realizamos resolución conste de una única cláusula. Enunciamos el resultado como un corolario del teorema de subsunción.

Corolario 5.12 (Corolario de interpolación) *Sean C_1 y C_2 dos cláusulas. Se tiene que $C_1 \models C_2$ si y sólo si existe una cláusula D tal que $C_1 \vdash_r D$ y $D \succeq C_2$.*

Por el teorema 4.20 se tiene que si $C \succeq D$ entonces existe un operador ϕ en $\mathbb{OP}^*$, resultado de la composición de OAS, tal que $D\phi = C$, por tanto, en virtud del corolario de interpolación (5.12) para conocer la naturaleza de los operadores que invierten la relación de consecuencia $\models$ basta conocer los operadores que invierten la relación de derivación $\vdash_r$.

La clave de estos operadores consiste en mantener invariantes los literales de un signo mientras que sólo se permiten transformaciones en los literales del otro

signo. Llamaremos a estos operadores de inversión *sesgados.*

Definición 5.13 *Dada un cláusula C, una sustitución θ y un signo $s \in \{\oplus, \ominus\}$ se define el* operador de inversión sesgado OIS *asociado al par $\langle C, \theta \rangle$ relativo al signo s como la aplicación $\{\theta_C^{-1}/s\}$*

$$\begin{array}{rccl} \{\theta_C^{-1}/\oplus\}: & \mathbb{C} & \longrightarrow & \mathbb{C} \\ & D & \mapsto & D\{\theta_C^{-1}/\oplus\} = \bigcup_{L \in D^{pos}} \{L' \in C \mid L'\theta = L\} \cup D^{neg} \end{array}$$

$$\begin{array}{rccl} \{\theta_C^{-1}/\ominus\}: & \mathbb{C} & \longrightarrow & \mathbb{C} \\ & D & \mapsto & D\{\theta_C^{-1}/\ominus\} = \bigcup_{L \in D^{neg}} \{L' \in C \mid L'\theta = L\} \cup D^{pos} \end{array}$$

Nota 5.14 Nótese que una descripción equivalente de estos operadores consiste en definirlos de la siguiente manera

$$D\{\theta_C^{-1}/\oplus\} = D^{neg} \cup [C^{pos} \cap (\bigcup_{L \in D} \{L' \mid L'\theta = L\})]$$

$$D\{\theta_C^{-1}/\ominus\} = D^{pos} \cup [C^{neg} \cap (\bigcup_{L \in D} \{L' \mid L'\theta = L\})]$$

Además, para cualesquiera C y θ

- Si D sólo tiene literales positivos, $D\{\theta_C^{-1}/\ominus\} = D$.
- Si D sólo tiene literales negativos, $D\{\theta_C^{-1}/\oplus\} = D$.

Por otra parte, se tiene que para cualesquiera cláusulas D_1, D_2 y C y cualquier sustitución θ y $s \in \{\oplus, \ominus\}$ se tiene que

$$(D_1 \cup D_2)\{\theta_C^{-1}/s\} = D_1\{\theta_C^{-1}/s\} \cup D_2\{\theta_C^{-1}/s\}$$

Por consiguiente, para cualesquiera C, D y θ

$$D\{\theta_C^{-1}/\ominus\}\{\theta_C^{-1}/\oplus\} = D\{\theta_C^{-1}/\oplus\}\{\theta_C^{-1}/\ominus\}$$

Estos operadores de inversión sesgados son los que nos van a permitir invertir el orden de derivación $\vdash_r$. Tal como están definidos, no es inmediato ver que los OIS sean elementos de $\mathbb{OP}^*$. Veremos en la próxima sección que en realidad todo OIS puede expresarse como composición de operadores clausales.

El siguiente lema relaciona los OIS con la relación de derivación entre dos cláusulas y es el resultado clave en el que nos apoyaremos para el teorema principal de esta sección.

La idea que guía el lema es la siguiente: Si D es una resolvente de C entonces existen dos sustituciones σ_1 y σ_2 tales que $C_1\sigma_1$ y $C_2\sigma_2$ son factores de C. Existe un renombramiento δ tal que $C_2\sigma_2\delta$ no tiene variables en común con $C_1\sigma_1$ y además existen unos literales $L_i \in C_1\sigma_1$ y $M_j \in C_2\sigma_2\delta$ y un umg γ de $\overline{L_i}$ y M_j (con $\overline{L_i}$ el literal complementario de L_i) tales que

$$D = (C_1\sigma_1 - \{L_i\})\gamma \cup (C_2\sigma_2\delta - \{M_j\})\gamma$$

Si llamamos $\theta_1 = \sigma_1\gamma$ y $\theta_2 = \sigma_2\delta\gamma$ y φ_p es el OIS$^{\oplus}$ asociado al par $\langle C, \theta_2\rangle$ y φ_n el OIS$^{\ominus}$ asociado al par $\langle C, \theta_1\rangle$, entonces $C = D\varphi_n\varphi_p$.

Lema 5.15 *Si D es una resolvente de C, entonces existen un OIS$^{\oplus}$ φ_p y un OIS$^{\ominus}$ φ_n tales que*

$$C = D\varphi_n\varphi_p$$

Demostración:
Si D es una resolvente de C entonces existen dos conjuntos de literales $E^{pos} \subseteq C^{pos}$, $E^{neg} \subseteq C^{neg}$, dos sustituciones θ_1, θ_2 y un átomo A tales que

(I) $E^{pos}\theta_1 = \{A\}$, $E^{neg}\theta_2 = \{\neg A\}$

(II) $C\theta_1 = \{A\} \cup [C^{pos} - E^{pos}]\theta_1 \cup C^{neg}\theta_1$

(III) $C\theta_2 = C^{pos}\theta_2 \cup \{\neg A\} \cup [C^{neg} - E^{neg}]\theta_2$

(IV) $D = [C^{pos} - E^{pos}]\theta_1 \cup C^{neg}\theta_1 \cup C^{pos}\theta_2 \cup [C^{neg} - E^{neg}]\theta_2$, esto es, tenemos que

$$D^{pos} = C^{pos}\theta_2 \cup [C^{pos} - E^{pos}]\theta_1$$
$$D^{neg} = C^{neg}\theta_1 \cup [C^{neg} - E^{neg}]\theta_2$$

Sean φ_p el OIS$^{\oplus}$ asociado al par $\langle C, \theta_2\rangle$ y φ_n el OIS$^{\ominus}$ asociado al par $\langle C, \theta_1\rangle$. Por la definición de φ_p y φ_n se tiene que

$$D^{pos}\,\varphi_p = D^{pos}\,\{\theta_2^{-1}/\oplus\} = \bigcup_{L \in D^{pos}} \{L' \in C \mid L'\theta = L\}$$

$$D^{neg}\,\varphi_n = D^{neg}\,\{\theta_1^{-1}/\ominus\} = \bigcup_{L \in D^{neg}} \{L' \in C \mid L'\theta = L\}$$

Por consiguiente

$$\begin{aligned} & D\varphi_n\varphi_p \\ = \; & (D^{pos} \cup D^{neg})\varphi_n\varphi_p \\ = \; & D^{pos}\varphi_n\varphi_p \cup D^{neg}\varphi_n\varphi_p \\ = \; & D^{pos}\varphi_p \cup [\bigcup_{L\in D^{neg}} \{L' \in C \mid L'\theta_1 = L\}]\varphi_p \\ = \; & [\bigcup_{L\in D^{pos}} \{L' \in C \mid L'\theta_2 = L\}] \cup [\bigcup_{L\in D^{neg}} \{L' \in C \mid L'\theta_1 = L\}] \end{aligned}$$

luego

$$(D\varphi_n\varphi_p)^{pos} = \bigcup_{L\in D^{pos}} \{L' \in C \mid L'\theta_2 = L\} \subseteq C^{pos}$$
$$(D\varphi_n\varphi_p)^{neg} = \bigcup_{L\in D^{neg}} \{L' \in C \mid L'\theta_1 = L\} \subseteq C^{neg}$$

Por tanto $D\varphi_n\varphi_p \subseteq C$. Para ver $C \subseteq D\varphi_n\varphi_p$ hay que ver que

$$C^{pos} \subseteq [D\varphi_n\varphi_p]^{pos}] = \bigcup_{L\in D^{pos}} \{L' \in C \mid L'\theta_2 = L\}$$
$$C^{neg} \subseteq [D\varphi_n\varphi_p]^{neg}] = \bigcup_{L\in D^{neg}} \{L' \in C \mid L'\theta_1 = L\}$$

o lo que es lo mismo, $C^{pos}\theta_2 \subseteq D^{pos}$ y $C^{neg}\theta_1 \subseteq D^{neg}$, pero esto se tiene por **(IV)**, luego

$$C = D\varphi_n\varphi_p$$

⊣

Ilustraremos el teorema con un ejemplo.

Ejemplo 5.16 Consideremos las siguientes cláusulas

$$C = \{q(f(x_2)), p(x_2, f(x_1)), \neg p(f(x_3), x_1), \neg r(x_3)\}$$

$$D = \{q(f(f(x_5))), q(f(x_2)), p(x_2, f(f(x_4))), \neg p(f(x_6), x_4), \neg r(x_6), \neg r(x_5)\}$$

donde separamos los literales positivos y negativos

$$C^{pos} = \{q(f(x_2)), p(x_2, f(x_1))\}$$
$$C^{neg} = \{\neg p(f(x_3), x_1), \neg r(x_3)\}$$

$$D^{pos} = \{q(f(f(x_5))), q(f(x_2)), p(x_2, f(f(x_4)))\}$$
$$D^{neg} = \{\neg p(f(x_6), x_4), \neg r(x_6), \neg r(x_5)\}$$

Tomemos ahora $E^{pos} \subseteq C^{pos}$ y $E^{neg} \subseteq C^{neg}$

$$E^{pos} = \{p(x_2, f(x_1))\} \qquad E^{neg} = \{\neg p(f(x_3), x_1)\}$$

y sean $A = p(f(x_5), f(x_4)$ y θ_1, θ_2 las sustituciones

$$\theta_1 = \{x_1/x_4, x_2/f(x_5), x_3/x_6\} \qquad \theta_2 = \{x_1/f(x_4), x_3/x_5\}$$

Si consideramos ahora las cláusulas

$$C\theta_1 = \{\{q(f(f(x_5))), p(f(x_5), f(x_4)), \neg p(f(x_6), x_4), \neg r(x_6)\}$$
$$C\theta_2 = \{\{q(f(x_2)), p(x_2, f(f(x_4))), \neg p(f(x_5), f(x_4)), \neg r(x_5)\}$$

haciendo resolución sobre el literal A obtenemos la cláusula D. Sean ahora φ_p y φ_n, respectivamente, el OIS$^{\oplus}$ asociado al par $\langle C, \theta_2\rangle$ y el OIS$^{\ominus}$ asociado al par $\langle C, \theta_1\rangle$. Si aplicamos estos operadores según la caracterización de la nota 5.14 tenemos:

$$\begin{aligned} D\varphi_n &= D^{neg} \cup [C^{pos} \cap (\bigcup_{L \in D} \{L' \mid L'\theta_2 = L\})] \\ &= D^{neg} \cup \{q(f(x_2), p(x_2, f(x_1))\} \\ &= \{q(f(x_2), p(x_2, f(x_1)), \neg p(f(x_6), x_4), \neg r(x_6), \neg r(x_5)\} \end{aligned}$$

Aplicando ahora el operador φ_n a la cláusula $D\varphi_p$ tenemos

$$\begin{aligned} D\varphi_p\varphi_n &= [D\varphi_p]^{pos} \cup [C^{neg} \cap (\bigcup_{L \in D\varphi_p} \{L' \mid L'\theta_1 = L\})] \\ &= [D\varphi_p]^{pos} \cup \{\neg p(f(x_3), x_1), \neg r(x_3)\} \\ &= \{q(f(x_2)), p(x_2, f(x_1)), \neg p(f(x_3), x_1), \neg r(x_3)\} \\ &= C \end{aligned}$$

El lema que acabamos de ver nos dice que podemos obtener D a partir de C si $C \vdash_r D$ en el caso particular en que D se obtenga a partir de C en un paso de resolución. El teorema siguiente presenta el caso general. $Res^n(C)$ denotará el conjunto de cláusulas que podemos obtener a partir de C mediante derivaciones de longitud menor o igual que n.

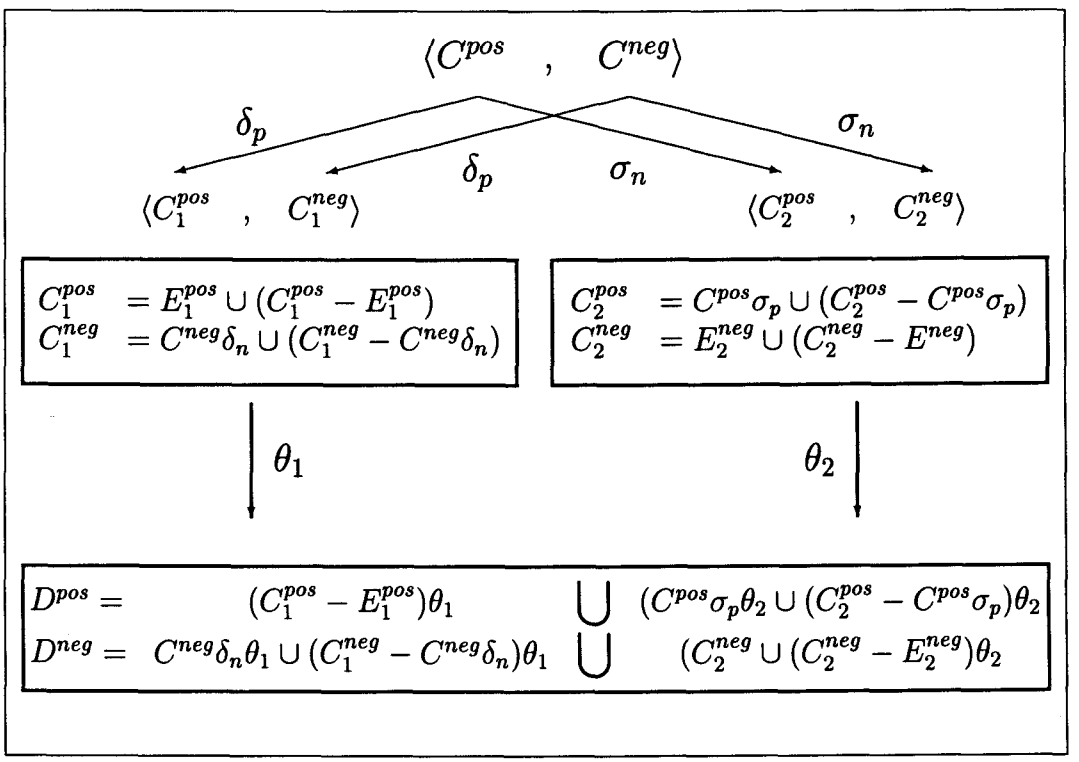


Figura 5.1: OIS y resolución

Teorema 5.17 *Sean C y D dos cláusulas tales que $C \vdash_r D$. Entonces existen un $OIS^{\oplus}$, φ_p y un $OIS^{\ominus}$, φ_n tales que*

$$C = D\varphi_n\varphi_p$$

Demostración:

Sean $C \vdash_r D$, entonces, por definición, o bien $C = D$, o bien existen dos cláusulas $C_1, C_2 \in Res^{n-1}(C)$ para algún $n \geq 1$ tales que D es la resolvente de C_1 y C_2.

Caso 1: $C = D$.

En este caso basta tomar la sustitución vacía ϵ y $\varphi_p = \{\epsilon_D^{-1}/\oplus\}$ y $\varphi_n = \{\epsilon_D^{-1}/\ominus\}$. Así $D\{\epsilon_D^{-1}/\ominus\}\{\epsilon_D^{-1}/\oplus\} = D = C$.

Caso 2: Existen dos cláusulas $C_1, C_2 \in Res^{n-1}(C)$ para algún $n \geq 1$ tales que D es la resolvente de C_1 y C_2.

Se tiene que $C \vdash_r C_1$ y $C \vdash_r C_2$ y de ahí que $C \models C_1$ y $C \models C_2$. Por el lema de Gottlob, tenemos que $C^{pos} \succeq C_1^{pos}$, $C^{neg} \succeq C_1^{neg}$, $C^{pos} \succeq C_2^{pos}$ y $C^{neg} \succeq C_2^{neg}$. En particular existen dos sustituciones σ_p y δ_n tales que $C^{pos}\sigma_p \subseteq C_2^{pos}$ y $C^{neg}\delta_n \subseteq C_1^{neg}$.

En la figura 5.1 podemos encontrar un esquema de la situación.

Por un razonamiento análogo al del lema 5.15, si D es la resolvente de C_1 y C_2 entonces existen $E_1^{pos} \subseteq C_1^{pos}$, $E_2^{neg} \subseteq C_2^{neg}$, dos sustituciones θ_1 y θ_2 y un átomo A tales que

(I) $E_1^{pos}\theta_1 = \{A\}$, $E_2^{neg}\theta_2 = \{\neg A\}$

(II) $C\theta_1 = \{A\} \cup [C^{pos} - E^{pos}]\theta_1 \cup C^{neg}\theta_1$

(III) $C\theta_2 = C^{pos}\theta_2 \cup \{\neg A\} \cup [C^{neg} - E^{neg}]\theta_2$

(IV) $D = [C^{pos} - E^{pos}]\theta_1 \cup C^{neg}\theta_1 \cup C^{pos}\theta_2 \cup [C^{neg} - E^{neg}]\theta_2$, esto es, tenemos que

$$D^{pos} = C_2^{pos}\theta_2 \cup [C_1^{pos} - E_1^{pos}]\theta_1$$
$$D^{neg} = C_1^{neg}\theta_1 \cup [C_2^{neg} - E_2^{neg}]\theta_2$$

Sea φ_p el OIS$^{\oplus}$ asociado al par $\langle C, \sigma_p\theta_2\rangle$ y φ_n el OIS$^{\ominus}$ asociado al par $\langle C, \delta_n\theta_1\rangle$. Se tiene que

$$D^{pos}\varphi_p = \bigcup_{L \in D^{pos}} \{L' \in C \mid L'\sigma_p\theta_2 = L\}$$
$$D^{neg}\varphi_n = \bigcup_{L \in D^{neg}} \{L' \in C \mid L'\delta_n\theta_1 = L\}$$

Por consiguiente

$$\begin{aligned} & D\varphi_n\varphi_p \\ = \; & \textstyle\bigcup_{L \in D^{pos}} \{L' \in C \mid L'\sigma_p\theta_2 = L\} \cup \\ & \textstyle\bigcup_{L \in D^{neg}} \{L' \in C \mid L'\delta_n\theta_1 = L\} \\ \subseteq \; & C^{pos} \cup C^{neg} \\ = \; & C \end{aligned}$$

Para tener el resultado basta ver que

$$C^{pos} \subseteq \textstyle\bigcup_{L \in D^{pos}} \{L' \in C \mid L'\sigma_p\theta_2 = L\}$$
$$C^{neg} \subseteq \textstyle\bigcup_{L \in D^{neg}} \{L' \in C \mid L'\delta_n\theta_1 = L\}$$

esto es, $C^{pos}\sigma_p\theta_2 \subseteq D^{pos}$ y $C^{neg}\delta_n\theta_1 \subseteq D^{neg}$. Pero esto se tiene ya que sabemos que $C^{pos}\sigma_p \subseteq C_2^{pos}$ y por **(IV)** y $C_2^{pos}\theta_2 \subseteq D^{pos}$, luego $C^{pos}\sigma_p\theta_2 \subseteq^{pos}$. Análogamente, tenemos que $C^{neg}\delta_n \subseteq C_1^{neg}$ y por **(IV)**, $C_1^{neg}\theta_1 \subseteq D^{neg}$. De ahí que $C^{neg}\delta_n\theta_1 \subseteq D^{neg}$ ⊣

En virtud del teorema de subsunción (teorema 5.7), o más exactamente, del corolario de interpolación (corolario 5.12) y del teorema anterior, tenemos el siguiente resultado.

Corolario 5.18 *Sean C y D dos cláusulas tales que $C \models D$. Entonces existe ϕ composición de OAS y dos OIS φ_p y φ_n tales que*

$$C = D\phi\varphi_n\varphi_p$$

Este corolario nos da información sobre la naturaleza de los operadores necesarios para obtener C a partir de D cuando $C \vDash D$. Son los operadores de aprendizaje para la subsunción y los operadores de inversión sesgados. Ambos tipos de operadores son de naturaleza muy distinta. De hecho, aún no hemos probado que los OIS se puedan expresar como composición de operadores clausales.

En la sección siguiente no sólo expresamos los OIS como elementos de $\mathbb{OP}^*$, sino que definiremos un nuevo tipo de operadores: los *operadores de generalización minimales (OGM)*. Estos operadores son operadores clausales y tienen una propiedad muy interesante: Cualquier operador de inversión sesgado puede expresarse como composición de OGM, con lo que tendremos que todo OIS pertenece a $\mathbb{OP}^*$, pero también todo OAS puede expresarse como composición de OGM. Por tanto, esta clase de operadores permite generalizar las tres relaciones binarias necesarias para el proceso de aprendizaje basado en cláusulas (Corolario 5.28).

5.3 Operadores de generalización minimales

Estudiemos los operadores mínimos capaces de generar OAS y OIS.

Definición 5.19 *Una aplicación $\{\Delta^{\oplus}/x\} : \mathbb{C} \longrightarrow \mathbb{C}$ se dice que es un* operador de generalización minimal, OGM *positivo si existe un OAS $\{\Delta/x\}$ tal que para*

toda cláusula C se verifica

$$C\{\Delta^{\oplus}/x\} = C^{pos}\{\Delta/x\} \cup C^{neg}$$

De manera análoga se definen los OGM negativos $\{\Delta^{\ominus}/x\} : \mathbb{C} \longrightarrow \mathbb{C}$.

$$C\{\Delta^{\ominus}/x\} = C^{neg}\{\Delta/x\} \cup C^{pos}$$

Dado el OAS $\{\Delta/x\}$ *los operadores* $\{\Delta^{\oplus}/x\}$ *y* $\{\Delta^{\ominus}/x\}$ *se dice que son los* operadores de generalización minimal *(OGM) asociados al OAS* $\{\Delta/x\}$.

Hemos usado la notación de operador clausal para los OGM aunque su definición no sigue las pautas con las que definimos los operadores clausales, esto es, necesitamos encontrar una asignación de literales asociada al OGM para ver que realmente se trata de un operador clausal. La proposición siguiente afirma que los operadores que acabamos de definir son operadores clausales y otras propiedades interesantes.

Proposición 5.20 *Se verifican las siguientes propiedades:*

(1) Los OGM son operadores clausales

(2) Los OGM no *son OAS*

(3) Si $\{\Delta/x\}$ *es un OAS y* $\{\Delta^{\oplus}/x\}$, $\{\Delta^{\ominus}/x\}$ *son los OGM asociados, entonces, para toda cláusula* C

$$C\{\Delta/x\} = C\{\Delta^{\oplus}/x\}\{\Delta^{\ominus}/x\}$$

(4) Sean $\{\Delta_1^{\oplus}/x_1\}$, $\{\Delta_2^{\ominus}/x_2\}$ *dos OGM y* C *una cláusula cualquiera. Entonces*

$$C\{\Delta_1^{\oplus}/x_1\}\{\Delta_2^{\ominus}/x_2\} = C\{\Delta_2^{\ominus}/x_2\}\{\Delta_1^{\oplus}/x_1\}$$

Demostración:
Lo vemos punto por punto.

(1) Los OGM son operadores clausales.

Sea $\{\Delta^{\oplus}/x\}$ el OGM positivo asociado al OAS $\{\Delta/x\}$. Para ver que $\{\Delta^{\oplus}/x\}$ es un operador clausal, basta encontrar una asignación de posi-

ciones Δ_0 tal que $(\forall C \in \mathbb{C})[C\{\Delta^{\oplus}/x\} = C\{\Delta_0/x\}$. Sea Δ_0 la aplicación

$$\begin{array}{rccl} \Delta_0: & Lit & \longrightarrow & \mathcal{PP}\mathbb{N}^+ \\ & L & \mapsto & \Delta(L) = \left\{ \begin{array}{ll} \Delta(L) & \text{si } L \text{ es positivo} \\ \{\emptyset\} & \text{si } L \text{ es negativo} \end{array} \right. \end{array}$$

Puesto que Δ es una asignación de posiciones, Δ_0 también lo es, ya que si L es un literal y t un término tales que $\Delta(L)$ es compatible con $\langle L, t\rangle$ entonces, $\Delta_0(L)$ también es compatible con $\langle L, t\rangle$. Por tanto $\{\Delta_0/L\}$ es un operador clausal. Llamaremos a Δ_0 la *asignación de posiciones sesgada positiva* asociada a Δ. Además

$$\begin{aligned} C\{\Delta_0/x\} &= \bigcup_{L\in C} L\{\Delta_0(L)/x\} \\ &= [\bigcup_{L\in C^{pos}} L\{\Delta_0(L)/x\}] \cup [\bigcup_{L\in C^{neg}} L\{\Delta_0(L)/x\}] \\ &= [\bigcup_{L\in C^{pos}} L\{\Delta(L)/x\}] \cup [\bigcup_{L\in C^{neg}} L\{\{\emptyset\}/x\}] \\ &= C^{pos}\{\Delta/x\} \cup C^{neg} \end{aligned}$$

(2) Como hemos visto en el apartado anterior, si $\{\Delta^{\oplus}/x\}$ es el OGM positivo asociado a $\{\Delta/x\}$ se tiene que

$$\begin{array}{rccl} \Delta^{\oplus}: & Lit & \longrightarrow & \mathcal{PP}\mathbb{N}^+ \\ & L & \mapsto & \Delta^{\oplus}(L) = \left\{ \begin{array}{ll} \Delta(L) & \text{si } L \text{ es positivo} \\ \{\emptyset\} & \text{si } L \text{ es negativo} \end{array} \right. \end{array}$$

Por tanto para todo literal L negativo L en el que ocurra x se tiene que $\Delta^{\oplus}(L) \neq \emptyset$ y $x \in Var(L)$. Por tanto, por el teorema 4.16, $\{\Delta^{\oplus}/x\}$ no es un OAS. El mismo resultado se tiene para los OGM negativos.

(3) Si $\{\Delta/x\}$ es un OAS y $\langle\{\Delta^{\oplus}/x\}, \{\Delta^{\ominus}/x\}\rangle$ son los OGM asociados, entonces, para toda cláusula C se tiene que $C\{\Delta/x\} = C(\{\Delta^{\oplus}/x\} \circ \{\Delta^{\ominus}/x\})$

$$\begin{aligned} & C(\{\Delta^{\oplus}/x\} \circ \{\Delta^{\ominus}/x\}) \\ = \; & (C\{\Delta^{\oplus}/x\})\{\Delta^{\ominus}/x\} \\ = \; & (C^{pos}\{\Delta/x\} \cup C^{neg})\{\Delta^{\ominus}/x\} \\ = \; & C^{pos}\{\Delta/x\} \cup C^{neg}\{\Delta/x\} \\ = \; & C\{\Delta/x\} \end{aligned}$$

(4) Sean $\{\Delta_1^{\oplus}/x_1\}$, $\{\Delta_2^{\ominus}/x_2\}$ dos OGM y C una cláusula cualquiera. Entonces $C\{\Delta_1^{\oplus}/x_1\}\{\Delta_2^{\ominus}/x_2\} = C\{\Delta_2^{\ominus}/x_2\}\{\Delta_1^{\oplus}/x_1\}$

$$\begin{aligned} & C\{\Delta_1^{\oplus}/x_1\}\{\Delta_2^{\ominus}/x_2\} \\ = \; & (C^{pos}\{\Delta_1/x_1\} \cup C^{neg})\{\Delta_2^{\ominus}/x_2\} \\ = \; & C^{pos}\{\Delta_1/x_1\} \cup C^{neg}\{\Delta_2/x_2\} \\ = \; & (C^{neg}\{\Delta_2/x_2\} \cup C^{pos})\{\Delta_1^{\oplus}/x_1\} \\ = \; & C\{\Delta_2^{\ominus}/x_2\}\{\Delta_1^{\oplus}/x_1\} \end{aligned}$$

⊣

Por consiguiente, la notación elegida para los OGM, $\{\Delta^{\oplus}/x\}$ y $\{\Delta^{\ominus}/x\}$ es coherente con la notación para operadores clausales, representando $\Delta^{\oplus}$ y $\Delta^{\ominus}$ las asignaciones de posiciones sesgadas asociadas a la asignación Δ.

Nota 5.21 Nótese que

- En general, dos OGM del mismo signo no conmutan.

  Basta considerar, por ejemplo, las asignaciones de posiciones

$$\Delta_1(L) = \begin{cases} \{\{\{1\}\}\} & \text{si } L = p(a) \\ \emptyset & \text{en otro caso} \end{cases}$$

$$\Delta_2(L) = \begin{cases} \{\emptyset\} & \text{si } L = p(x_1) \\ \emptyset & \text{en otro caso} \end{cases}$$

$$\{p(a)\}\{\Delta_1^{\oplus}/x_1\}\{\Delta_2^{\oplus}/x_2\} = \{p(x_1)\}\{\Delta_2^{\oplus}/x_2\} = \{p(x_1)\}$$
$$\{p(a)\}\{\Delta_2^{\oplus}/x_2\}\{\Delta_1^{\oplus}/x_1\} = \emptyset$$

- Si tenemos la composición de varios OGM podemos reagruparlos poniendo juntos los del mismo signo respetando el orden entre ellos (por la proposición 5.20, apdo. 4).

Veamos a continuación que si podemos pasar de D_1 a D_2 mediante un OIS entonces también podemos pasar de D_1 a D_2 mediante una cadena de OGM.

Lema 5.22 *Sea* $\{\theta_c^{-1}/\oplus\}$ *un OIS*$^{\ominus}$ *y sean* D_1 *y* D_2 *dos cláusulas tales que* $D_1\{\theta_c^{-1}/\oplus\} = D_2$. *Entonces existen unos OGM,* $\{\Delta_1^{\oplus}/x_1\}, \{\Delta_2^{\oplus}/x_2\}, \ldots, \{\Delta_n^{\oplus}/x_n\}$ *tales que*

$$D_1\{\Delta_1^{\oplus}/x_1\}\{\Delta_2^{\oplus}/x_2\}\ldots\{\Delta_n^{\oplus}/x_n\} = D_2$$

Demostración:
A partir de la definición de $\{\theta_c^{-1}/\oplus\}$ se tiene que

$$D_2^{pos} = \bigcup_{L \in D_1^{pos}} \{L' \in C \mid L'\theta = L\}$$

por tanto tenemos que $D^{pos_2}\theta \subseteq D_1^{pos}$, luego $D_2^{pos} \succeq D_1^{pos}$ y por el teorema 4.20 existe una serie de OAS $\{\Delta_1/x_1\}, \{\Delta_2/x_2\}, \ldots, \{\Delta_n/x_n\}$ tales que

$$D_1^{pos}\{\Delta_1/x_1\}\{\Delta_2/x_2\}\ldots\{\Delta_n/x_n\} = D_2^{pos}$$

Sean $\{\Delta_1^{\oplus}/x_1\}, \{\Delta_2^{\oplus}/x_2\}\ldots\{\Delta_n^{\oplus}/x_n\}$ los OGM$^{\oplus}$ asociados a los OAS. Se tiene lo siguiente:

$$\begin{aligned}
& D_1\{\theta_c^{-1}/\oplus\} \\
= \; & (D_1^{pos} \cup D_1^{neg})\{\theta_c^{-1}/\oplus\} \\
= \; & D_1^{pos}\{\theta_c^{-1}/\oplus\}D_1^{neg}\{\theta_c^{-1}/\oplus\} \\
= \; & D_1^{pos}\{\theta_c^{-1}/\oplus\} \cup D_1^{neg} \\
= \; & D_2^{pos} \cup D_1^{neg} \\
= \; & D_1^{pos}\{\Delta_1/x_1\}\{\Delta_2/x_2\}\ldots\{\Delta_n/x_n\} \cup D_1^{neg} \\
= \; & D_1^{pos}\{\Delta_1^{\oplus}/x_1\}\{\Delta_2^{\oplus}/x_2\}\ldots\{\Delta_n^{\oplus}/x_n\} \cup D_1^{neg} \\
= \; & D_1\{\Delta_1^{\oplus}/x_1\}\{\Delta_2^{\oplus}/x_2\}\ldots\{\Delta_n^{\oplus}/x_n\}
\end{aligned}$$

⊣

El siguiente lema afirma que todo OGM puede verse localmente como un OIS. Enunciamos el lema para operadores positivos. En el caso negativo el resultado es análogo.

Lema 5.23 *Supongamos que* $D_1\{\Delta^{\oplus}/x\} = D_2$. *Entonces existen* θ *y* C *tales que* $D_1\{\theta_C^{-1}/\oplus\} = D_2$

Demostración:
Sea $\{\Delta/x\}$ el OAS asociado al OGM$^{\oplus}$ $\{\Delta^{\oplus}/x\}$. Distinguimos dos casos.

Caso 1: Para todo literal positivo de D_1, se tiene que $\Delta(L) \subseteq \{\emptyset\}$

En este caso $D_2^{pos} \subseteq D_1^{pos}$ y basta tomar $C = D_2$ y $\theta = \epsilon$. Entonces

$$\begin{aligned} & D_1\{\theta_C^{-1}/\oplus\} \\ = & \bigcup_{L \in D_1^{pos}} \{L' \in D_2 \mid L' = L\} \cup D_1^{neg} \\ = & D_2^{pos} \cup D_1^{neg} \\ = & D_2^{pos} \cup D_2^{neg} \\ = & D_2 \end{aligned}$$

Caso 2: Existe un literal positivo L_0 en D_1 tal que $\Delta(L_0) \not\subseteq \{\emptyset\}$.

En este caso, sea t el término que permite la compatibilidad de Δ y sean $\theta = \{x/t\}$ y $C = D_2$. Se tiene lo siguiente

$$\begin{aligned} & D_1\{\theta_C^{-1}/\oplus\} \\ = & \bigcup_{L \in D_1^{pos}} \{L' \in D_2 \mid L'\{x/t\} = L\} \cup D_1^{neg} \\ = & D_2^{pos} \cup D_1^{neg} \\ = & D_2^{pos} \cup D_2^{neg} \\ = & D_2 \end{aligned}$$

⊣

Por último, veamos un resultado que nos dice que si podemos generalizar D_1 para obtener D_2 mediante una serie de OGM positivos, entonces también lo podemos generalizar mediante un OIS.

Lema 5.24 *Sean* $\{\Delta_1^{\oplus}/x_1\}\{\Delta_2^{\oplus}/x_2\}\ldots\{\Delta_n^{\oplus}/x_n\}$ *una sucesión de OGM*$^{\oplus}$ *y* D_1 *y* D_2 *dos cláusulas tales que*

$$D_1\{\Delta_1^{\oplus}/x_1\}\{\Delta_2^{\oplus}/x_2\}\ldots\{\Delta_n^{\oplus}/x_n\} = D_2$$

Entonces existen θ *y* C *tales que* $D_1\{\theta_C^{-1}/\oplus\} = D_2$.

Demostración:
Si $D_1\{\Delta_1^{\oplus}/x_1\}\{\Delta_2^{\oplus}/x_2\}\ldots\{\Delta_n^{\oplus}/x_n\} = D_2$, entonces, por el lema 4.17 $D_2^{pos} \succeq$

D_1^{pos} y existe θ tal que $D_2^{pos}\theta \subseteq D_1^{pos}$. Tomemos $C = D_2$. Entonces

$$\begin{aligned} & D_1\{\theta_{D_2}^{-1}/\oplus\} \\ = & \bigcup_{L\in D_1}\{L' \in D_2 \mid L'\theta = L\} \cup D_1^{neg} \\ = & D_2^{pos} \cup D_2^{neg} \\ = & D_2 \end{aligned}$$

⊣

Por el teorema 5.17 tenemos que si $C \vdash_r D$ entonces existen dos OIS φ_n y φ_p tales que

$$C = D\varphi_n\varphi_p$$

Si combinamos este resultado con el lema 5.22 obtenemos el siguiente corolario.

Corolario 5.25 *Si C y D son dos cláusulas tales que $C \vdash_r D$ entonces existen dos cadenas de OAS*

$$\{\Delta_1^{\ominus}/x_1\}.....\{\Delta_n^{\ominus}/x_n\}$$
$$\{\Delta_{n+1}^{\oplus}/x_{n+1}\}.....\{\Delta_{n+m}^{\oplus}/x_{n+m}\}$$

tales que $D\{\Delta_1^{\ominus}/x_1\}\ldots\{\Delta_n^{\ominus}/x_n\}\{\Delta_{n+1}^{\oplus}/x_{n+1}\}\ldots\{\Delta_{n+m}^{\oplus}/x_{n+m}\} = C$.

Finalmente, por el apartado (3) de la proposición 5.20 tenemos que todo OAS podemos expresarlo como composición de sus OGM asociados, esto es, para toda cláusula C y todo OAS $\{\Delta/x\}$ se tiene que

$$C\{\Delta/x\} = C\{\Delta^{\oplus}/x\}\{\Delta^{\ominus}/x\}$$

Por tanto, a partir del corolario 5.18 tenemos el siguiente resultado.

Corolario 5.26 *Si $C \models D$ y D no es tautología, entonces existen n OGM, $\Gamma_1,\ldots,\Gamma_n$ tales que*

$$C = D\Gamma_1\ldots\Gamma_n$$

Demostración:
Por 5.18, $C = \phi\varphi_n\varphi_p$ con ϕ la composición de varios OAS y φ_n y φ_p dos OIS. La cadena de OGM se obtiene mediante la descomposición de cada uno de estos operadores en OGM. ⊣

Nota 5.27 La otra implicación no es cierta. Si volvemos a las cláusulas de la nota 5.10

$$C = \{suma(s(x), y, s(z)), \neg suma(x, y, z)\}$$
$$D = \{suma(s(0), y, s(s(0))), \neg suma(0, y, 0)\}$$

y consideramos los OAS $\{\Delta_1/x\}$, $\{\Delta_2/z\}$, $\{\Delta_3/x\}$ y $\{\Delta_4/z\}$ con

$$\Delta_1(L) = \begin{cases} \{\{1 \cdot 1\}\} & \text{si } L = suma(s(0), y, s(s(0))) \\ \emptyset & \text{en otro caso} \end{cases}$$

$$\Delta_2(L) = \begin{cases} \{\{3 \cdot 1\}\} & \text{si } L = suma(s(x), y, s(s(0))) \\ \emptyset & \text{en otro caso} \end{cases}$$

$$\Delta_3(L) = \begin{cases} \{\{1\}\} & \text{si } L = \neg suma(0, y, 0) \\ \emptyset & \text{en otro caso} \end{cases}$$

$$\Delta_4(L) = \begin{cases} \{\{3\}\} & \text{si } L = \neg suma(x, y, 0) \\ \emptyset & \text{en otro caso} \end{cases}$$

entonces lo OGM asociados $\{\Delta_1^{\oplus}/x\}$, $\{\Delta_2^{\oplus}/z\}$, $\{\Delta_3^{\ominus}/x\}$ y $\{\Delta_4^{\ominus}/z\}$ verifican que

$$D\{\Delta_1^{\oplus}/x\}\{\Delta_2^{\oplus}/z\}\{\Delta_3^{\ominus}/x\}\{\Delta_4^{\ominus}/z\} = C$$

y sin embargo $C \nvdash D$.

Enunciamos, como resultado final del capítulo, que los OGM son los operadores básicos para la generalización en las tres relaciones básicas en el aprendizaje clausal. El resultado se tiene a partir del corolario 5.26 y del apartado (4) de la proposición 5.20

Corolario 5.28 *Sean C y D dos cláusulas tales que se verifica una de las siguientes condiciones*

- $C \succeq D$
- $C \vdash_r D$
- $C \models D$ *y D no es una tautología*

entonces existen dos operadores en $\mathbb{OP}^*$, Γ_p^*, composición de OGM positivos y

Γ_n^*, *composición de OGM negativos, tales que*

$$D\Gamma_p^*\Gamma_n^* = D\Gamma_n^*\Gamma_p^* = C$$

Nota 5.29 De manera análoga a como razonamos en el capítulo anterior también podemos plantearnos el problema de asignar un valor cuantitativo a la relación de consecuencia entre cláusulas mediante una función que a cada par de cláusulas C y D le asignara la longitud de la cadena de OGM más corta que nos llevara D en C si $C \models D$ e infinito si $C \not\models D$, pero esta definición no tendría ningún sentido práctico puesto que la relación de implicación entre cláusulas no es decidible [SS88].

Capítulo 6

Subsunción entre programas

En este capítulo extendemos nuestro estudio sobre el aprendizaje y los operadores clausales a programas. En la primera parte definimos una relación binaria sobre el conjunto de programas: la subsunción entre programas basada en la relación de subsunción entre cláusulas.

Nuestra definición y resultados extienden y precisan la noción de equivalencia entre programas basada en subsunción (véase [Mah88]). En su artículo, M.J. Maher define una equivalencia entre programas basada en subsunción, pero no la relación de subsunción entre programas. No obstante, a partir del análisis de sus demostraciones se deduce que la definición de subsunción implícita en el artículo coincide con nuestra propuesta.

En este capítulo relacionamos la subsunción entre programas con nuestros operadores. Para ellos definimos unos operadores, los *OAS compuestos*, a partir de los Operadores de Aprendizaje para la Subsunción (OAS) del capítulo 4 y que actúan sobre programas con ayuda de unas funciones auxiliares que llamaremos *aplicadores*.

De manera análoga a como ocurría con los OAS y la subsunción entre cláusulas (Corolario 4.21), los OAS compuestos también representan una caracterización de la relación de subsunción entre programas: El programa P_1 es más general que P_2 en el orden de subsunción si y sólo si existe una cadena de OAS compuestos $\Theta_1, \ldots, \Theta_n$ tales que $P_2\Theta_1 \ldots \Theta_n \subseteq P_1$ (Corolario 6.12).

Esta caracterización de la relación de subsunción entre programas mediante cadenas de OAS compuestos nos lleva a considerar las cadenas de OAS compuestos de menor longitud entre programas y a cuantificar mediante una pseudo-quasi-distancia la subsunción entre programas. En la sección 6.4 establecemos la

relación entre estas cadenas mínimas de OAS compuestos y la quasi-distancia *dc* entre cláusulas y damos un método de cálculo para la pseudo-quasi-distancia.

Como hemos visto, la definición de subsunción que presentamos apareció de manera implícita en un artículo de Maher y este artículo nos brinda la clave para relacionar la subsunción entre programas con la semántica de los programas definidos: El operador *consecuencia inmediata* T_P de Van Emden y Kowalski [VK76]. En la parte final del capítulo nos apoyaremos en estos resultados para relacionar nuestros operadores clausales con el aprendizaje clausal de la Programación Lógica Inductiva.

Como vimos en el capítulo 2, la idea central del aprendizaje de conceptos se basa en definir un concepto como el conjunto de sus instancias. Por consiguiente, si usamos una representación clausal para aproximarnos al problema del Aprendizaje de Conceptos, un concepto será un conjunto de átomos cerrados que comparten el mismo símbolo de predicado, o visto de otro modo, todo subconjunto I de la base de Herbrand de un lenguaje admite una partición en *conceptos*, donde cada concepto está formado por los elementos de I con el mismo símbolo de predicado.

Si el concepto es finito, podremos conocerlo por enumeración de sus elementos. Si es infinito, sólo podremos definirlo mediante una fórmula (programa lógico en nuestro caso), por tanto, todo programa definido P determina varios conceptos: tantos como símbolos de predicado tenga el lenguaje. Cada uno de esos conceptos estará formado por los elementos del menor modelo de Herbrand de P, M_P, asociados al correspondiente símbolo de predicado.

En este contexto, tiene sentido preguntarse *qué es aprender.* Dado un símbolo de predicado p y un programa definido P, consideramos la definición actual de nuestro concepto como el conjunto de átomos de M_P con el símbolo de predicado p. Si descubrimos que existe un átomo A que debe pertenecer a nuestro concepto, pero $P \not\models A$ (esto es, $A \notin M_P$), nuestro concepto debe ser *generalizado* para *cubrir* el átomo A, luego nuestra definición informal de *aprendizaje por generalización* en programas definidos es la siguiente: *Aprender es el proceso que nos permite pasar del programa definido* P_1 *al programa definido* P_2 *cumpliendo que exista un átomo cerrado* A *tal que* $P_1 \not\models A$ *y* $P_2 \models P_1 \cup \{A\}$.

Distinguiremos entre *aprendizaje* y *síntesis de conocimiento.* En esta memoria diremos que se ha producido *aprendizaje por generalización* cuando pasamos del programa P_1 al programa P_2 y el menor modelo de Herbrand de P_2 contiene en sentido estricto al menor modelo de Herbrand de P_1.

El caso en que $M_{P_1} = M_{P_2}$ también resulta interesante cuando para alguna medida apropiada μ, $\mu(P_2) < \mu(P_1)$. En este caso P_1 y P_2 cubren las mismas observaciones pero P_2 necesita menos recursos. En este caso hablaremos de *síntesis del conocimiento.*

6.1 Subsunción

Comenzamos nuestro estudio extendiendo la relación de subsunción a programas. La subsunción entre programas es la extensión natural a programas de la relación de subsunción entre cláusulas. En [Mah88], Maher define una relación de equivalencia entre programas basada en la relación de subsunción entre cláusulas. No define la relación de subsunción entre programas, pero está implícita en sus resultados.

En este capítulo veremos que la generalización natural de los OAS a programas, que denominamos *OAS compuestos*, representan una caracterización mediante operadores de la relación de subsunción entre programas. Veamos en primer lugar cómo se define la relación de subsunción entre programas.

Definición 6.1 *Sea P_1 un programa. Diremos que el programa P_2 es más general que P_1 bajo subsunción, $P_2 \succeq P_1$, si para toda cláusula $C \in P_1$ existe una cláusula $D \in P_2$ tal que $D \succeq C$, esto es,*

$$P_2 \succeq P_1 \Leftrightarrow (\exists F : P_1 \to P_2)(\forall C \in P_1)[F(C) \succeq C]$$

La equivalencia entre programas se tiene de manera natural.

Definición 6.2 *Diremos que los programas P_1 y P_2 son equivalentes bajo subsunción y lo representaremos por $P_1 \sim P_2$ si $P_1 \succeq P_2$ y $P_2 \succeq P_1$.*

Proposición 6.3 *La relación de subsunción entre programas tiene las siguientes propiedades*

1. *Es un quasi-orden sobre el conjunto de programas.*
2. *Si $P_1 = \{C_1\}$ y $P_2 = \{C_2\}$ entonces $P_2 \succeq P_1$ si y sólo si $C_2 \succeq C_1$*
3. *Si $P_1 = P \cup \{C_1\}$, $P_2 = P \cup \{C_2\}$ y $C_2 \succeq C_1$ entonces $P_2 \succeq P_1$.*
4. *Si $P_1 \subseteq P_2$, entonces $P_2 \succeq P_1$.*

Demostración:
Veamos cada una de las propiedades

1. *Reflexiva:* Basta tomar la identidad como aplicación $Id : P \to P$ y $(\forall C \in P)[Id(C) = C \succeq C]$.
   Transitiva:

$$\left.\begin{array}{l} P_2 \succeq P_1 \Leftrightarrow (\exists F_1 : P_1 \to P_2)(\forall C \in P_1)[F_1(C) \succeq C] \\ P_3 \succeq P_2 \Leftrightarrow (\exists F_2 : P_2 \to P_3)(\forall C \in P_2)[F_2(C) \succeq C] \end{array}\right\} \Longrightarrow$$
$$(\exists F' = F_2 \circ F_1 : P_1 \to P_3)(\forall C \in P_1)[F'(C) = F_2(F_1(C)) \succeq F_1(C) \succeq C]$$

   luego $P_3 \succeq P_1$

2. $P_2 \succeq P_1 \Leftrightarrow (\exists F : P_1 \to P_2)(\forall C \in P_1)(F(C) \succeq C) \Leftrightarrow C_2 \succeq C_1$.

3. Basta tomar

$$\begin{array}{rccl} F: & P_1 & \longrightarrow & P_2 \\ & C & \mapsto & F(C) = \left\{\begin{array}{ll} C_2 & \text{si } C = C_1 \\ C & \text{si } C \neq C_1 \end{array}\right. \end{array}$$

4. Basta tomar

$$\begin{array}{rccl} F: & P_1 & \longrightarrow & P_2 \\ & C & \mapsto & F(C) = C \end{array}$$

⊣

6.2 OAS compuestos

En esta sección vamos a estudiar la relación entre la subsunción entre programas y nuestros operadores. Para ello definiremos un operador que actúe sobre programas.

Definición 6.4 *Un* OAS compuesto *es un conjunto finito de pares* Δ_i/x_i

$$\Theta = \{\Delta_1/x_1, \ldots, \Delta_n/x_n\}$$

donde $\{\Delta_i/x_i\}$ *es un OAS para todo* $i \in \{1, \ldots, n\}$.

La aplicación de un OAS compuesto a un programa debe realizarse sin ambigüedades, por lo que tendremos que determinar qué OAS aplicamos a cada una de las cláusulas del programa. Esto lo hacemos mediante una función que llamamos *aplicador.*

Definición 6.5 *Dado un programa P y un OAS compuesto Θ*

- *Un* aplicador *es una función $a : P \to \Theta$ tal que para toda cláusula de P, la cláusula $C\,\{a(C)\}$ no es la cláusula vacía.*

- *Definimos el programa*

$$P_a\Theta = \{C\,\{a(C)\} \mid C \in P\}$$

como el programa resultante de aplicar a P el OAS compuesto Θ mediante el aplicador a.

Para simplificar la notación escribiremos $P\Theta$ en lugar de $P_a\Theta$ cuando no haya ambigüedad sobre el aplicador.

Ilustramos estas definiciones con el siguiente ejemplo[1]:

Ejemplo 6.6 Sea $P = \{C_1, C_2, C_3\}$ con

$$\begin{aligned}
C_1 &= suma(0, s(s(0)), s(s(0))) \leftarrow \\
&\qquad suma(0, s(0), s(0)) \\
C_2 &= suma(s(x), s(0), s(z)) \leftarrow \\
&\qquad suma(s(0), 0, s(0)), \\
&\qquad suma(x, s(0), z), \\
C_3 &= suma(s(y), y, s(z)) \leftarrow \\
&\qquad suma(0, y, y), \\
&\qquad suma(y, y, z)
\end{aligned}$$

[1] A lo largo de este capítulo adoptaremos la notación Prolog $A \leftarrow B_1, \ldots, B_n$ en lugar de $\{A, \neg B_1, \ldots, \neg B_n\}$ para cláusulas definidas.

y $\Theta = \{\Delta_1/x,\ \Delta_2/y,\ \Delta_3/x\}$ con

$$\Delta_1(L) = \begin{cases} \{\{2,\ 3\}\} & \text{si } L = suma(0, s(s(0)), s(s(0))) \\ \emptyset & \text{en otro caso} \end{cases}$$

$$\Delta_2(L) = \begin{cases} \{\{2\}\} & \text{si } L \in \left\{ \begin{array}{l} suma(s(x), s(0), s(z)) \\ \neg suma(x, s(0), z) \end{array} \right\} \\ \emptyset & \text{en otro caso} \end{cases}$$

$$\Delta_3(L) = \begin{cases} \{\{1 \cdot 1\}\} & \text{si } L = suma(s(y), y, s(z)) \\ \{\{1\}\} & \text{si } L = \neg suma(y, y, z) \\ \emptyset & \text{en otro caso} \end{cases}$$

Consideremos el aplicador $a : P \to \Theta$ tal que $a(C_1) = \Delta_1/x$, $a(C_2) = \Delta_2/y$, $a(C_3) = \Delta_3/x$. Entonces

$$\begin{array}{lcll} C_1\{a(C_1)\} & = & C_1\{\Delta_1/x\} & \equiv \ suma(0, x, x) \leftarrow \\ C_2\{a(C_2)\} & = & C_2\{\Delta_2/y\} & \equiv \ suma(s(x), y, s(z)) \leftarrow \\ & & & \qquad suma(x, y, z), \\ C_3\{a(C_2)\} & = & C_3\{\Delta_3/x\} & \equiv \ suma(s(x), y, s(z)) \leftarrow \\ & & & \qquad suma(x, y, z), \end{array}$$

Por consiguiente

$$P_a\Theta = \left\{ \begin{array}{l} suma(0, x, x) \leftarrow \\ suma(s(x), y, s(z)) \leftarrow \\ \quad suma(x, y, z), \end{array} \right\}$$

A continuación empezamos nuestro estudio que relaciona los OAS compuestos con la relación de subsunción entre cláusulas. Llegaremos a que nuestros operadores para programas también representan una *caracterización* de la relación de subsunción entre programas de manera análoga a cómo se producía con la relación de subsunción entre cláusulas.

Teorema 6.7 *Sean P y P^* dos programas y Θ un OAS compuesto. Si $P\Theta \subseteq P^*$ entonces $P^* \succeq P$, donde aplicamos Θ a P mediante un aplicador $a : P \to \Theta$.*

Demostración:
Basta considerar $F : P \to P^*$ tal que $F(C) = C\{a(C)\}$ ya que por ser $\{a(C)\}$ un OAS se tiene que $C\{a(C)\}$ subsume a C. ⊣

A partir de este resultado se tiene de manera inmediata el siguiente corolario

Corolario 6.8 *Sea P y P^* dos programas y $\Theta_1, \ldots, \Theta_n$ una cadena de OAS compuestos. Si $P\Theta_1 \ldots \Theta_n \subseteq P^*$, donde cada Θ_i ha sido aplicado mediante un aplicador correspondiente, entonces $P^* \succeq P$.*

Por tanto, si aplicamos a un programa P una serie de OAS compuestos $\Theta_1, \ldots, \Theta_n$ obtenemos un programa más general en el orden de subsunción entre programas. En realidad, cualquier superconjunto de $P\Theta_1 \ldots \Theta_n$ es más general que P.

A continuación probaremos el recíproco de 6.8, con lo que tendremos una caracterización de la relación de subsunción entre programas basada en OAS compuestos. Veamos antes un lema previo.

Lema 6.9 *Para toda cláusula C existe un OAS $\{\Delta/x\}$ tal que $C\{\Delta/x\} = C$.*

Demostración:
Basta tomar x una variable que no ocurra en C y

$$\Delta(L) = \begin{cases} \{\emptyset\} & \text{si } L \in C \\ \emptyset & \text{si } L \notin C \end{cases}$$

⊣

Teorema 6.10 *Sean P_0 y P_1 dos programas. Si $P_0 \succeq P_1$ entonces existe una cadena de OAS compuestos $\Theta_1, \ldots, \Theta_m$ tales que $P_1\Theta_1 \ldots \Theta_m \subseteq P_0$ donde cada OAS compuesto se ha aplicado mediante un aplicador apropiado.*

Demostración:
Si $P_0 \succeq P_1$ entonces existe una aplicación $F : P_1 \to P_0$ tal que para todo $C^i \in P_1$ se tiene que $F(C^i) \succeq C^i$ y por tanto, para cada $C^i \in P_1$ existe una cadena $\{\Delta_1^i/x_1^i\}, \ldots, \{\Delta_{n_i}^i/x_{n_i}^i\}$ de OAS tales que

$$C^i\{\Delta_1^i/x_1^i\} \ldots \{\Delta_{n_i}^i/x_{n_i}^i\} = F(C^i)$$

Sea $m = max\{n_i \mid C^i \in P_1\}$ y para todo $j \in \{1, \ldots, m\}$ se define

$$\Theta_j = \{\, \Delta_j^i/x_j^i \mid C^i \in P \wedge j \leq n_i\} \bigcup \{I_{F(C_i)} \mid C_i \in P \wedge j > n_i\}$$

donde $\{I_{F(C_i)}\}$ es un OAS que deja invariante $F(C_i)$. Tal OAS existe según hemos visto en el lema 6.9. Consideremos también los aplicadores

$$\begin{array}{rlll} a_j: & P_1\Theta_1 \ldots \Theta_{j-1} & \rightarrow & \Theta_j \\ & C^i & \mapsto & a_j(C^i) = \begin{cases} \Delta^i_j/x^i_j & \text{si } j \leq n_i \\ I_{F(C^i)} & \text{si } j > n_i \end{cases} \end{array}$$

entonces $\Theta_1, \ldots \Theta_m$ es una cadena de OAS compuestos tal que

$$P_1\Theta_1 \ldots \Theta_m = \{F(C) \mid C \in P_1\} \subseteq P_0$$

donde cada OAS compuesto se ha aplicado con el aplicador correspondiente.
⊣

Nota 6.11 En la demostración anterior hemos tomado una función $F : P_1 \rightarrow P_0$ y para cada $C_i \in P_1$ encontramos una cadena $\{\Delta^i_1/x^i_1\}, \ldots, \{\Delta^i_{n_i}/x^i_{n_i}\}$ de OAS tales que

$$C^i\{\Delta^i_1/x^i_1\} \ldots \{\Delta^i_{n_i}/x^i_{n_i}\} = F(C^i)$$

y finalmente encontramos una cadena de OAS compuestos $\Theta_1, \ldots, \Theta_m$ tal que

$$P_1\Theta_1 \ldots \Theta_m = \{F(C) \mid C \in P_1\} \subseteq P_0$$

donde $m = max\{n_i \mid C^i \in P_1\}$. Si para cada $C^i \in P_1$ tomamos una cadena *minimal*, esto es, una cadena de OAS $\{\Delta^i_1/x^i_1\}, \ldots, \{\Delta^i_{n_i}/x^i_{n_i}\}$ tal que $n_i = dc(C^i, F(C^i))$ entonces obtendríamos una cadena de OAS compuestos de longitud $m^* = max\{dc(F(C^i), C^i) \mid C^i \in P_1\}$. Nótese que m^* sólo depende de P_1 y F.

Por último, a partir del corolario 6.8 y del teorema 6.10, enunciamos en el siguiente resultado que los OAS compuestos representan una caracterización de la relación de subsunción entre programas.

Corolario 6.12 *Sean P_1 y P_2 dos programas. Entonces $P_1 \succeq P_2$ si y sólo si existe una cadena de OAS compuestos $\Theta_1, \ldots, \Theta_n$ tales que $P_2\Theta_1 \ldots \Theta_n \subseteq P_1$ donde cada OAS compuesto se ha aplicado mediante un aplicador apropiado.*

6.3 Un dominio cuantitativo

En las sección anterior hemos visto que $P_0 \succeq P$ si y sólo si $P\Theta_1 \ldots \Theta_n \subseteq P_0$, esto es, si podemos generalizar P mediante OAS compuestos para obtener un subconjunto de P_0. De manera análoga a como ocurría con las cláusulas, si $P_0 \succeq P$, sabemos que podemos obtener al menos una cadena de OAS compuestos $\Theta_1, \ldots, \Theta_n$ que lleve P en un subconjunto de P_0, pero esta cadena $\Theta_1, \ldots, \Theta_n$ no tiene por qué ser única, ni tener todas las cadenas la misma longitud, como muestra el siguiente ejemplo.

Ejemplo 6.13 Consideremos los programas $P = \{D\}$ y $P_0 = \{C_1, C_2\}$ con

$$\begin{array}{rcl} D & \equiv & p(a,b) \leftarrow q(a,b), r(b) \\ C_1 & \equiv & p(x,y) \leftarrow r(y) \\ C_2 & \equiv & p(x,b) \leftarrow q(x,b) \end{array}$$

Entonces $P_0 \succeq P$ ya que $F_1 : P \to P_0$ con $C_1 = F_1(D) \succeq D$ ya que si $\theta_1 = \{x/a, y/b\}$ entonces $C_1\theta_1 \subseteq D$. Además, si tomamos

$$\Delta_1(L) = \begin{cases} \{\{1\}\} & \text{si } L = p(a,b) \\ \{\emptyset\} & \text{si } L = \neg r(b) \\ \emptyset & \text{en otro caso} \end{cases}$$

$$\Delta_2(L) = \begin{cases} \{\{2\}\} & \text{si } L = p(x,b) \\ \{\{1\}\} & \text{si } L = \neg r(b) \\ \emptyset & \text{en otro caso} \end{cases}$$

entonces $D\{\Delta_1/x\}\{\Delta_2/y\} = \{p(x,b), \neg r(b)\}\{\Delta_2/y\} \equiv p(x,y) \leftarrow r(y) \equiv C_1$. Por consiguiente podemos considerar los OAS compuestos $\Theta_1 = \{\Delta_1/x\}$ y $\Theta_2 = \{\Delta_2/y\}$ junto los aplicadores

$$\begin{array}{rcl} a_1 : P_1 & \longrightarrow & \Theta_1 \\ D & \mapsto & \Delta_1/x \end{array}$$

$$\begin{array}{rcl} a_2 : P_{a_1}\Theta_1 & \longrightarrow & \Theta_2 \\ D\{\Delta_1/x\} & \mapsto & \Delta_2/y \end{array}$$

con lo que tendremos que $P\Theta_1\Theta_2 \subseteq P_2$ aplicando los OAS compuestos con los

aplicadores a_1 y a_2.

Por otra parte, también tenemos que mediante $\theta_2 = \{x/a\}$ se tiene que $C_2\theta_2 \subseteq D$ y tomando

$$\Delta_3(L) = \begin{cases} \{\{1\}\} & \text{si } L \in \{p(a,b), \neg q(a,b)\} \\ \emptyset & \text{en otro caso} \end{cases}$$

entonces $D\{\Delta_3/x\} = C_2$. Si tomamos el OAS compuesto $\Theta_3 = \{\Delta_3/x\}$ y el aplicador

$$\begin{array}{rcccl} a_3 & : & P & \longrightarrow & \Theta_3 \\ & & D & \mapsto & \Delta_3/x \end{array}$$

se tiene que $P_{a_3}\Theta_3 \subseteq P_2$.

De manera análoga a como ocurría con las cadenas de OAS entre cláusulas, en este caso también podemos cuantificar la relación de subsunción entre programas basándonos en el número mínimo de OAS compuestos que nos lleva un programa en otro.

Definición 6.14 *Sean P_1 y P_2 dos programas tales que $P_1 \succeq P_2$. Diremos que*

- *$\mathcal{C} = \langle\langle\Theta_1, a_1\rangle, \ldots, \langle\Theta_n, a_n\rangle\rangle$ es una cadena de P_1 a P_2 si*
  - *$\Theta_1, \ldots, \Theta_n$ son OAS compuestos*
  - *Para todo $i \in \{1, \ldots, n\}$, $a_i : P_2\Theta_1 \ldots \Theta_{i-1} \to \Theta_i$ es un aplicador*
  - *$P_2\Theta_1 \ldots \Theta_n \subseteq P_1$ donde los OAS compuestos se han aplicado mediante los a_i correspondientes.*

  en ese caso diremos que $\mathcal{C}$ es una cadena de longitud n y lo denotaremos por $|\mathcal{C}| = n$. Si $P_2 \subseteq P_1$, entonces diremos que la cadena vacía, de longitud cero, es una cadena de P_1 a P_2.

- *Denotaremos por $\mathbf{L}(P_1, P_2)$ el conjunto de cadenas de P_1 a P_2.*

Si $P_2 \succeq P_1$ el conjunto $\mathbf{L}(P_1, P_2)$ no es vacío, luego tiene sentido la siguiente definición.

Definición 6.15 *Definimos la aplicación $dp : \mathbb{P} \times \mathbb{P} \to [0, +\infty]$ de la siguiente manera*

$$dp(P_1, P_2) = \begin{cases} min\{|\mathcal{C}| : \mathcal{C} \in \mathbf{L}(P_1, P_2)\} & si\ P_1 \succeq P_2 \\ +\infty & e.o.c. \end{cases}$$

UNIVERSIDAD DE SEVILLA
FACULTAD DE MATEMATICAS
BIBLIOTECA

Proposición 6.16 *La función dp tiene las siguientes propiedades*

- $P_1 \subseteq P_2 \Leftrightarrow dp(P_2, P_1) = 0$, *en particular* $dp(P, P) = 0$.
- $dp(P_1, P_2) \leq dp(P_1, P_0) + dp(P_0, P_2)$

Demostración:
La primera equivalencia, $P_1 \subseteq P_2 \Leftrightarrow dp(P_2, P_1) = 0$, se tiene trivialmente ya que la cadena vacía es una cadena de P_2 a P_1.

Para la propiedad triangular, si $dp(P_1, P_0) = +\infty$ o $dp(P_0, P_2) = +\infty$ no hay nada que probar, luego supongamos que $P_1 \succeq P_0$ y $P_0 \succeq P_2$. Si $dp(P_1, P_0) = n$ y $dp(P_0, P_2) = m$ entonces existen dos cadenas de OAS compuestos $\Theta_1, \ldots, \Theta_m$ y $\Theta_{m+1}, \ldots, \Theta_{m+n}$ tales que $P_2\Theta_1 \ldots \Theta_n \subseteq P_0$ y $P_0\Theta_{n+1} \ldots \Theta_{n+m} \subseteq P_1$ donde cada Θ_i se ha aplicado con un aplicador apropiado. Ajustando los aplicadores tenemos que

$$P_2\Theta_1 \ldots \Theta_n\Theta_{n+1} \ldots \Theta_{n+m} \subseteq P_1$$

con lo que $dp(P_1, P_2) \leq m + n = dp(P_1, P_0) + dp(P_0, P_2)$ ⊣

Luego, la función *dp* es una función que pertenece al conjunto conocido como *distancias débiles*, más concretamente, *dp* es una *pseudo-quasi-distancia* y el conjunto de programas $\mathbb{P}$ junto con la función *dp* forman un *dominio cuantitativo* (ver [Win00]).

Nota 6.17 La función $dp(P_1, P_2)$ mide el número mínimo de operadores que tenemos que aplicar a P_2 para obtener un subconjunto de P_1. Es una función no simétrica, incluso en el caso en que P_1 y P_2 sean equivalentes bajo subsunción. Esto es debido a que programas equivalentes bajo subsunción pueden ser sintácticamente muy diferentes.

Ilustramos este hecho con un ejemplo adaptado de [Mah88].

Consideremos los programas siguientes

$$P_1 \equiv \left\{ C_0 \equiv p(x_0) \leftarrow q(x_0, y_0), r(y_0) \right.$$

$$P_2 \equiv \left\{ \begin{array}{l} C_1 \equiv p(c) \leftarrow q(c, y_1), r(y_1) \\ C_2 \equiv p(x_2) \leftarrow q(x_2, y_2), r(y_2), q(z_2, y_2) \\ C_3 \equiv p(x_3) \leftarrow q(x_3, y_3), r(y_3), s(y_3, x_3) \end{array} \right.$$

se tiene que

- $P_1 \succeq P_2$ puesto que $C_0 \succeq C_1$, $C_0 \succeq C_2$ y $C_0 \succeq C_3$.
- $P_2 \succeq P_1$ puesto que $C_2 \succeq C_0$.

por tanto $P_1 \succeq P_2$ y $P_2 \succeq P_1$. No obstante

$$\begin{array}{lll} dc(C_0, C_1) = 2 & dc(C_0, C_2) = 2 & dc(C_0, C_3) = 2 \\ dc(C_1, C_0) = +\infty & dc(C_2, C_0) = 3 & dc(C_3, C_0) = +\infty \end{array}$$

de ahí que $dp(P_1, P_2) = 2$ y $dp(P_2, P_1) = 3$.

6.4 Cálculo efectivo

La función *dp* se define como la longitud de la cadena mínima de OAS compuestos entre dos programas. El número de OAS compuestos necesarios para llevar un programa en un subconjunto del otro depende de la quasi-distancia entre cláusulas, para la cual tenemos un algoritmo que la calcula.

En esta sección proporcionamos un método para cuantificar la proximidad entre programas.

Teorema 6.18 *Sean P_1 y P_2 dos programas, entonces*

$$dp(P_1, P_2) = \max_{D \in P_2} \left\{ \min_{C \in P_1} \{dc(C, D)\} \right\}$$

A este teorema le corresponde el predicado `pq_dist/3` en el apéndice B.

Demostración:

Veamos primero la equivalencia en el caso en que P_1 no subsume a P_2

$$\begin{array}{ll} & dp(P_1, P_2) = +\infty \\ \Leftrightarrow & \neg(P_1 \succeq P_2) \\ \Leftrightarrow & \neg\,(\forall D \in P_2)\,(\exists C \in P_1)\,(C \succeq D) \\ \Leftrightarrow & (\exists D \in P_2)\,(\forall C \in P_1)\,\neg(D \succeq C) \\ \Leftrightarrow & (\exists D \in P_2)\,(\forall C \in P_1)\,(dc(C, D) = +\infty) \\ \Leftrightarrow & (\exists D \in P_2)\,[\min_{C \in P_1}\{dc(C, D)\} = +\infty] \\ \Leftrightarrow & \max_{D \in P_2}\{\min_{C \in P_1}\{dc(C, D)\}\} = +\infty \end{array}$$

Veamos a continuación el caso en que P_1 subsume a P_2. Desglosamos la demostración en las dos desigualdades.

- $\max_{D\in P_2}\{\min_{C\in P_1}\{dc(C,D)\}\} \leq dp(P_1,P_2)$

  Supongamos que $dp(P_1,P_2) = n < +\infty$. Entonces existe una cadena de OAS compuestos $\Theta_1\ldots,\Theta_n$ tal que $P_2\Theta_1\ldots\Theta_n \subseteq P_1$ y para todo $D^i \in P_2$ existe una cadena de OAS $\{\Delta_1^i/x_1^i\},\ldots\{\Delta_{n_i}^i/x_{n_i}^i\}$ tal que

  $$D^i\{\Delta_1^i/x_1^i\},\ldots\{\Delta_{n_i}^i/x_{n_i}^i\} \in P_1$$

  luego, para toda cláusula $D \in P_2$ existe $C \in P_1$ tal que $C \succeq D$ y $dc(C,D) \leq n$, luego $(\forall D \in P_2)\,[\min_{C\in P_1}\{d(C,D)\} \leq n]$ y

  $$\max_{D\in P_2}\{\min_{C\in P_1}\{dc(C,D)\}\} \leq n = d(P_1,P_2)$$

- $d(P_1,P_2) \leq \max_{D\in P_2}\{\min_{C\in P_1}\{dc(C,D)\}\}$

  Por reducción al absurdo, supongamos que

  $$dp(P_1,P_2) = n > m = \max_{D\in P_2}\{\min_{C\in P_1}\{dc(C,D)\}\}$$

  Sea $F : P_2 \to P_1$ una aplicación tal que para todo $D \in P_2$ se verifique

  $$dc(F(D),D) = \min_{C\in P_1}\{dc(C,D)\}$$

  entonces, por la nota 6.11 sabemos que existe una cadena de OAS compuestos de P_1 a P_2 de longitud

  $$max\{dc(F(D),D) \mid D \in P_2\} = \max_{D\in P_2}\{\min_{C\in P_1}\{dc(C,D)\}\} = m$$

  Contradicción con que $dp(P,P_1) = n > m$.

⊣

Puesto que los programas son conjuntos finitos de cláusulas, el teorema anterior nos proporciona un método para calcular de manera efectiva $dp(P_1,P_2)$.

Nota 6.19 Nótese que la función

$$dp^*(P_1,P_2) = \max\{dp(P_1,P_2), dp(P_2,P_1)\}$$

es una distancia entre programas. La función dp^* es la distancia de Hausdorff

basada en OAS entre dos programas vistos como conjuntos de cláusulas.

6.5 Trabajos relacionados

En [Mah88], Maher define una relación de equivalencia entre programas basada en la relación de subsunción entre cláusulas mediante la idea de *programas en forma canónica*: Dado un programa P establecemos una partición en P mediante la relación de equivalencia por subsunción entre sus cláusulas. La forma canónica de P, el programa P^*, se forma tomando una cláusula reducida de cada una de las clases de equivalencia y eliminando aquellas cláusulas que están subsumidas por otras.

Es trivial ver que la relación de equivalencia definida por Maher y la de la definición 6.2 coinciden. En la nota 6.17 vimos un ejemplo de programas equivalentes. En ese ejemplo P_1 es la forma canónica de P_2.

En el citado artículo, se relaciona la relación de subsunción entre programas con el operador consecuencia inmediata de Kowalski. Recogemos en la proposición 6.21 algunos de los resultados de Maher. Veamos antes la siguiente definición.

Definición 6.20 *Sean C_1 y C_2 dos cláusulas definidas y P_1 y P_2 dos programas definidos.*

- $T_{C_1} \geq T_{C_2}$ *si y sólo si* $(\forall I)[T_{C_2}(I) \subseteq T_{C_1}(I)]$
- $T_{P_1} \geq T_{P_2}$ *si y sólo si* $(\forall I)[T_{P_2}(I) \subseteq T_{P_1}(I)]$

Veamos ahora los resultados. Hemos adaptado la notación original a la usada en este capítulo.

Proposición 6.21 ([Mah88]) *Sean C_1 y C_2 dos cláusulas definidas y P_1 y P_2 dos programas*

- $P_1 \succeq P_2$ *si y sólo si* $T_{P_1} \geq T_{P_2}$.
- $P_1 \sim P_2$ *si y sólo si* $T_{P_1} = T_{P_2}$

Estos resultados de Maher relacionan la semántica para programas definidos basados en el operador de Kowalski T_P con la subsunción entre programas.

Podemos por tanto relacionar ahora la semántica de los programas definidos con nuestros operadores de la siguiente manera:

Corolario 6.22 *Sean P_1 y P_2 dos programas definidos. Son equivalentes:*

- $T_{P_1} \geq T_{P_2}$
- $P_1 \succeq P_2$
- *Existe una cadena de OAS compuestos $\Theta_1, \ldots, \Theta_n$ tales que $P_2\Theta_1 \ldots \Theta_n \subseteq P_1$ donde cada OAS compuesto se ha aplicado mediante un aplicador apropiado.*
- $dp(P_1, P_2) < +\infty$

Demostración:
Se tiene a partir del corolario 6.12, de la proposición 6.21 y de la definición de *dp* (Def. 6.15). ⊣

Nota 6.23 Nótese que para todo programa P y todo OAS compuesto Θ se tiene que $T_{P\Theta} \geq T_P$ (donde aplicamos Θ mediante un aplicador apropiado) ya que $P\Theta \succeq P$.

6.6 Aprendizaje

Como veíamos en la introducción, podemos considerar un programa definido como la definición de varios conceptos: Tantos como símbolos de predicado diferentes podemos encontrar en las cláusulas. Con esta representación clausal, el universo sobre el que realizamos aprendizaje es la base de Herbrand del lenguaje y dado un programa P su menor modelo de Herbrand representa la unión de todos los conceptos definidos por P.

Ejemplo 6.24 Consideremos el programa P

$$P \equiv \begin{cases} padre(juan, alberto) \\ padre(juan, andres) \\ padre(jose, juan) \\ abuelo(X, Y) \leftarrow padre(X, Z), padre(Z, Y) \end{cases}$$

que nos define dos conceptos:

$$I^P_{padre} = \{padre(juan, alberto), padre(juan, andres), padre(jose, juan)\}$$
$$I^P_{abuelo} = \{abuelo(jose, alberto), abuelo(jose, andres)\}$$

y obviamente $M_P = I^P_{padre} \cup I^P_{abuelo}$.

Puesto que consideramos un concepto como el conjunto de todas las instancias (Sección 2.8) e identificamos el universo de instancias con la base de Herbrand de un lenguaje $\mathcal{L}$, el orden de aprendizaje entre programas lo definiremos a partir de la relación de subconjunto entre los menores modelos de Herbrand.

Definición 6.25 *Sean P_1 y P_2 dos programas definidos. Diremos que P_2 es más general que P_1 en el orden de aprendizaje si $M_{P_1} \subsetneq M_{P_2}$. Si $M_{P_1} = M_{P_2}$ entonces P_1 y P_2 son definiciones de los mismos conceptos. Si $M_{P_1} \subseteq M_{P_2}$ diremos que P_2 es más general o igual que P_1 en el orden de aprendizaje.*

El proceso de aprendizaje suele centrarse en un único concepto y los ejemplos son instancias cerradas correspondientes a un único símbolo de predicado.

Dado un programa P y un símbolo de predicado p consideraremos que la definición de p en P la forman aquellas cláusulas de P que tienen el símbolo de predicado p en la cabeza. En los procesos de aprendizaje del concepto asociado a un símbolo de predicado tomaremos el conocimiento básico fijo y sólo consideraremos cambios en la definición de p.

Definición 6.26 *Sea p un símbolo de predicado del lenguaje $\mathcal{L}$*

- *Un programa definido CB se dice que es un* conocimiento básico *para p si p no ocurre en CB.*

- *Un programa definido DEF es una definición para p si p es el símbolo de predicado de la cabeza de todas las cláusulas de DEF.*

Ilustramos esta definición con un ejemplo.

Ejemplo 6.27 Consideremos el siguiente programa CB

$$CB \equiv \begin{cases} arista(a_1, a_2) \leftarrow \\ arista(a_1, a_3) \leftarrow \\ arista(a_2, a_4) \leftarrow \\ arista(a_4, a_5) \leftarrow \\ arista(a_5, a_6) \leftarrow \end{cases}$$

y las siguientes definiciones del predicado *conectado*/2

$$DEF_1 \equiv \left\{ \begin{array}{l} C_1 = conectado(a_1, a_2) \leftarrow \\ \qquad arista(a_1, a_2) \\ C_2 = conectado(a_1, a_3) \leftarrow \\ \qquad arista(a_1, a_3) \\ C_3 = conectado(W, a_6) \leftarrow \\ \qquad arista(W, a_4), \\ \qquad conectado(a_4, a_6), \\ \qquad conectado(a_2, a_4), \\ \qquad arista(a_5, a_6) \end{array} \right.$$

$$DEF_2 \equiv \left\{ \begin{array}{l} D_1 = conectado(X, Y) \leftarrow arista(X, Y) \\ D_2 = conectado(X, Z) \leftarrow arista(X, Y), conectado(Y, Z) \end{array} \right.$$

con $P_1 = CB \cup DEF_1$, $P_2 = CB \cup DEF_2$. Nótese que DEF_1 y DEF_2 son dos definiciones del predicado *conectado*/2 que necesitan del conocimiento base CB. Además P_2 es más general que P_1 en el orden de subsunción ya que la aplicación $F : P_1 \rightarrow P_2$ con $F(C_1) = D_1$, $F(C_2) = D_1$, $F(C_3) = D_3$ y $F(C) = C$ si $C \in CB$ verifica que $(\forall C \in P_1)[F(C) \succeq C]$. Se tiene que $M_{CB} = \{arista(a_1, a_2), arista(a_1, a_3), arista(a_2, a_4), arista(a_4, a_5), arista(a_5, a_6)\}$ y

$$M_{P_1} = M_{CB} \cup \{conectado(a_1, a_2), conectado(a_1, a_3)\}$$

$$M_{P_2} = M_{CB} \cup \left\{ \begin{array}{lll} conectado(a_1, a_2), & conectado(a_1, a_3), & conectado(a_1, a_4), \\ conectado(a_1, a_5), & conectado(a_1, a_6), & conectado(a_2, a_4), \\ conectado(a_2, a_5), & conectado(a_2, a_6), & conectado(a_4, a_5), \\ conectado(a_4, a_6), & conectado(a_5, a_6) & \end{array} \right\}$$

por tanto $M_{P_1} \subsetneq M_{P_2}$ y P_2 es más general que P_1 en el orden de aprendizaje.

Definición 6.28 *Dado un lenguaje $\mathcal{L}$ y un símbolo de predicado p de $\mathcal{L}$ denotaremos por $[p]$ al conjunto de todos los átomos cerrados del lenguaje que tienen a p como símbolo de predicado.*

Si el objetivo del proceso de aprendizaje es generalizar un concepto asociado al símbolo de predicado p, podemos adaptar la definición 6.25 de la siguiente manera:

Definición 6.29 *Sea p un símbolo de predicado, CB un conocimiento básico para p y DEF_1 y DEF_2 dos definiciones para p. Sean $P_1 = CB \cup DEF_1$ y $P_2 = CB \cup DEF_2$. Diremos que P_1 es más general que P_2 en el orden de aprendizaje si $M_{P_2} \subsetneq M_{P_1}$*

En el caso en que $M_{P_2} = M_{P_1}$ ambos programas definen los mismos conceptos. En el caso en que P_1 represente una *mejora* respecto a P_2 mediante algún sistema de medida, se dice que el paso de P_2 a P_1 representa una *síntesis del conocimiento.*

Puesto que el menor modelo de Herbrand de un programa definido puede definirse en términos del operador de Kowalski T_P, podemos relacionar nuestros OAS compuestos con el aprendizaje clausal.

Proposición 6.30 *Dados dos programas definidos P_1 y P_2, si existe una cadena de OAS compuestos $\Theta_1, \ldots, \Theta_n$ tales que $P_2\Theta_1 \ldots \Theta_n \subseteq P_1$ (donde cada OAS compuesto se ha aplicado mediante un aplicador apropiado), entonces P_1 es más general (o igual) que P_2 en el orden de aprendizaje.*

Demostración:
Es muy simple a partir de la relación entre el menor modelo de Herbrand de un programa definido y el operador consecuencia de Kowalski. Basta tener en cuenta que por el corolario 6.22, si $P_2\Theta_1 \ldots \Theta_n \subseteq P_1$ entonces $T_{P_1} \geq T_{P_2}$ y por tanto

$$M_{P_2} = \bigcup_{n \geq 1} T^n_{P_2}(\emptyset) \subseteq \bigcup_{n \geq 1} T^n_{P_1}(\emptyset) = M_{P_1}$$

⊣

El recíproco no es cierto, como muestran los siguientes ejemplos

Ejemplo 6.31 Consideremos el lenguaje $\mathcal{L}$ que tiene como único símbolo de predicado $p/2$ y cuyos únicos símbolos de función son las constantes a y b. La base de Herbrand de $\mathcal{L}$ es

$$B_{\mathcal{L}} = \{p(a,a), p(a,b), p(b,a), p(b,b)\}$$

Consideremos los programas

$$P_1 \equiv \begin{cases} p(x,a) & \leftarrow \\ p(x,b) & \leftarrow \end{cases} \qquad P_2 \equiv \begin{cases} p(a,x) & \leftarrow \\ p(x,b) & \leftarrow \end{cases}$$

Tenemos en este caso $M_{P_1} = B_{\mathcal{L}}$ y $M_{P_2} = \{p(a,a), p(a,b), p(b,b)\}$. Tenemos que $M_{P_2} \not\subseteq M_{P_1}$, pero $p(a,x) \leftarrow$ y $p(x,a) \leftarrow$ son incompatibles por subsunción.

Ejemplo 6.32 Sea P el programa

$$P \equiv \begin{cases} p(0) \leftarrow \\ p(s^2(0)) \leftarrow \\ p(s^3(0)) \leftarrow \end{cases}$$

y las cláusulas $C_1 \equiv p(s^6(0)) \leftarrow$ y $C_2 \equiv p(s^6(x)) : -p(x)$. Sean $P_1 = P \cup \{C_1\}$ y $P_2 = P \cup \{C_2\}$. Entonces $M_{P_1} \subsetneq M_{P_2}$ y no existe ningún programa P_3 tal que $M_{P_3} = M_{P_2}$ y P_3 se haya obtenido generalizando alguna cláusula de P_1, ya que, si así fuera, $p(s^7(0)) \in M_{P_3}$ y $p(s^7(0)) \notin M_{P_2}$

Como resultado final de esta sección vemos que la equivalencia por subsunción entre programas implica la igualdad de conceptos, aunque el inverso no es cierto, como ilustra el ejemplo 6.34.

Corolario 6.33 *Si $P_1 \sim P_2$ entonces P_1 y P_2 son iguales en el orden de aprendizaje (i.e. $M_{P_1} = M_{P_2}$).*

Demostración:
A partir de 6.22 y 6.30. ⊣

El inverso no es cierto.

Ejemplo 6.34 Consideremos los programas

$$P_1 \equiv \begin{cases} nat(0) \\ nat(s(X)) \leftarrow nat(X) \end{cases}$$

$$P_2 \equiv \begin{cases} nat(0) \\ nat(s(0)) \\ nat(s(s(X))) \leftarrow nat(X) \end{cases}$$

con $C_1 \equiv nat(s(X)) \leftarrow nat(X) \in P_1$ y $C_2 \equiv nat(s(s(X))) \leftarrow nat(X) \in P_2$. No hay ninguna cláusula C en P_2 tal que $C \succeq C_1$ y no hay ninguna cláusula D en P_1 tal que $D \succeq C_2$. Luego ni $P_1 \succeq P_2$ ni $P_2 \succeq P_1$, sin embargo $M_{P_1} = M_{P_2} = \{s^n(0) : n \geq 0\}$.

6.7 Procesos infinitos

El estudio del aprendizaje como proceso de generalización entre programas basado en subsunción plantea interrogantes muy sugerentes y abre puertas hacia una profundización sobre la estructura interna del conjunto de programas. En esta sección nos preguntamos por la existencia de procesos infinitos de aprendizaje basado en operadores clausales. Si nuestro lenguaje tiene símbolos de función entonces la base de Herbrand es infinita y podemos encontrar una sucesión infinita de conceptos $S_0 = S$ y $S_{n+1} = S_n \cup \{e_{n+1}\}$ donde $S \subset M_P$ y $\{e_n\}_{n\geq 0}$ es una sucesión de ejemplos del predicado p, i.e. $\{e_n\}_{n\geq 0} \subseteq [p]$.

Nos preguntamos si podemos encontrar un programa P_0 y una sucesión de ejemplos $\{e_n\}_{n\geq 0}$ tales que para todo $n \geq 1$ $M_{P_{n+1}} = M_{P_n} \cup \{e_{n+1}\}$ y donde el paso de P_n a P_{n+1} se produce mediante OAS compuestos.

El siguiente ejemplo da una respuesta positiva a esta cuestión e ilustra que podemos encontrar procesos infinitos en el aprendizaje clausal basado en subsunción.

Ejemplo 6.35 Consideremos la sucesión de cláusulas

- $C_0 = \{q(x_0), \neg p(a,a), \neg p(a,x_0)\}$.
- $C_{2n+1} = [C_{2n} - \{\neg p(a,x_{2n})\}] \cup \{\neg p(x_{2n+1},a), \neg p(x_{2n+1},x_{2n})\}$
- $C_{2n+2} = [C_{2n+1} - \{\neg p(x_{2n+1},a)\}] \cup \{\neg p(a,x_{2n+2}), \neg p(a,x_{2n+1},x_{2n+2})\}$

las primeras cláusulas de la sucesión son

$$
\begin{array}{lllllll}
C_0 = \{ & q(x_0), & \neg p(a,a), & \neg p(a,x_0)\} & & & \\
C_1 = \{ & q(x_0), & \neg p(a,a), & \neg p(x_1,x_0), & \neg p(x_1,a)\} & & \\
C_2 = \{ & q(x_0), & \neg p(a,a), & \neg p(x_1,x_0), & \neg p(x_1,x_2), & \neg p(a,x_2)\} & \\
C_3 = \{ & q(x_0), & \neg p(a,a), & \neg p(x_1,x_0), & \neg p(x_1,x_2), & \neg p(x_3,x_2), & \neg p(x_3,a)\} \\
C_4 = \{ & q(x_0), & \neg p(a,a), & \neg p(x_1,x_0), & \neg p(x_1,x_2), & \neg p(x_3,x_2), & \neg p(x_3,x_4), \quad \neg p(a,x_4)\} \\
\ldots & & & & & & &
\end{array}
$$

Se tiene que[2]

- $C_{n+1} \succeq C_n$

[2]La demostración de que C_{n+1} subsume estrictamente a C_n es análoga a la de los apartados correspondientes en el teorema 7.8.

- $C_n \not\succeq C_{n+1}$

Consideremos el siguiente programa

$$CB \equiv \left\{ \begin{array}{l} p(a,a) \leftarrow \\ p(x,s(x)) \leftarrow \\ p(s(x),x) \leftarrow \end{array} \right.$$

Definimos la sucesión de programas $P_n = CB \cup \{C_n\}$.

Veamos algunas propiedades de estos programas.

Proposición 6.36 *Para toda* $n \geq 0$, $P_{n+1} \succeq P_n$ *y* $P_n \not\succeq P_{n+1}$

Demostración:
Para ver $P_{n+1} \succeq P_n$ basta considerar $F_n : P_n \to P_{n+1}$ con

$$F_n(C) = \left\{ \begin{array}{ll} C_{n+1} & \text{si } C = C_n \\ C & \text{si } C \in CB \end{array} \right.$$

Para ver $P_n \not\succeq P_{n+1}$ sólo hay que tener en cuenta que $C_n \not\succeq C_{n+1}$ y que si $C \in CB$ entonces $C \not\succeq C_{n+1}$. ⊣

A continuación vamos a estudiar los menores modelos de Herbrand de estos programas. En primer lugar, el menor modelo de Herbrand de CB es

$$\begin{array}{rll} M_{CB} = \{p(a,a)\} & \cup & \{p(s^n(a), s^{n+1}(a)) \mid n \geq 0\} \\ & \cup & \{p(s^{n+1}(a), s^n(a)) \mid n \geq 0\} \end{array}$$

Proposición 6.37 *Para todo* $n \geq 0$

$$M_{P_n} = M_{CB} \cup \{q(s^i(a)) \mid i \in \{0, 1, \ldots, n+1\}\}$$

Demostración:
Descomponemos la demostración en los siguientes asertos:

Aserto 1: $M_{P_n} = T_{P_n} \uparrow 1$

Demostración: Se tiene trivialmente, ya que para todo n el programa P_n sólo tiene una regla y el símbolo de predicado de la cabeza no ocurre en el cuerpo de la cláusula.

Aserto 2: $M_{P_n} \subseteq M_{P_{n+1}}$
Demostración:

$$M_{P_n} = T_{P_n} \uparrow 1 = M_{CB} \cup \{A \mid A \leftarrow \vec{B} \in Bas(C_n) \wedge \vec{B} \subseteq M_{CB}\}$$

A partir de la igualdad anterior, para probar que $M_{P_n} \subseteq M_{P_{n+1}}$ basta ver que $Bas(C_n) \subseteq Bas(C_{n+1})$. Sea $D \in Bas(C_n)$ y sea σ tal que $C_n\sigma = D$. Se tiene que $C_{n+1}\{x_{n+1}/a\} = C_n$, luego $C_{n+1}\{x_{n+1}/a\}\sigma = D$ y $D \in Bas(C_{n+1})$.

Aserto 3: $q(s^{n+1}(a)) \in M_{P_n}$
Demostración: Sea $\sigma = \{x_0/s^{n+1}(a), x_1/s^n(a), \ldots, x_n/s(a)\}$ $C_n\sigma$ es una cláusula básica cuya cabeza es $q(s^{n+1}(a))$. Para ver que $q(s^{n+1}(a)) \in M_{P_n}$ basta ver que

$$B = \{p(a,a), p(s^n(a), s^{n+1}(a)), p(s^n(a), s^{n-1}(a)), \ldots, p(s(a), a)\} \subseteq M_{CB}$$

pero esto se tiene, ya que para todo literal de B distinto de $P(a,a)$ la diferencia entre el número de ocurrencias del símbolo s en los dos argumentos es siempre uno.

Aserto 4: $M_{P_0} = M_{CB} \cup \{q(s^n(a)) \mid n \in \{0,1\}\}$
Demostración: Es inmediato

$$\begin{aligned} T_{P_0} \uparrow 0 &= M_{CB} \\ T_{P_0} \uparrow 1 &= M_{P_0} = M_{CB} \cup \{q(a), q(s(a))\} \end{aligned}$$

A partir de estos cuatro asertos se tiene que

$$M_{CB} \cup \{q(s^i(a)) : i \in \{0, \ldots, n+1\}\} \subseteq M_{P_n}$$

Para tener el resultado falta ver la otra contención. Está claro que los únicos elementos de $[p]$ que están en M_{P_n} son los de M_{CB}. Para tener la contención sólo tenemos que probar el siguiente aserto.

Aserto 5: Si $k > n+1$ entonces $q(s^k(a)) \notin M_{P_n}$
Demostración: Lo vemos por reducción al absurdo. Supongamos que $q(s^k(a)) \in M_{P_n}$. Entonces debe existir una sustitución cerrada θ tal que $x_0\theta = s^k(a)$ y si B es el conjunto de literales del cuerpo de C_n, entonces $B\theta \subseteq_{;CB}$. En particular $p(x_1, x_0)\theta \in M_{CB}$ y puesto que $x_0\theta = s^k(a)$ entonces $x_1\theta \in \{s^{k+1}(a), s^{k-1}(a)\}$. Análogamente, $P(x_1, x_2) \in M_{CB}$ y puesto que $x_1\theta \in \{s^{k+1}(a), s^{k-1}(a)\}$ tenemos que $x_2\theta \in \{s^{k+2}(a), s^k(a), s^{k-2}(a)\}$. Un razonamiento análogo nos lleva a que $x_3\theta \in \{s^{k+3}(a), s^{k+1}(a), s^{k-1}(a), s^{k-3}(a)\}$ e iterando el proceso llegamos a que

- Si n es par

  $$x_n\theta \in \{s^{k+n}(a), s^{k+n-2}(a), \ldots, s^{k+2}(a), s^k(a), s^{k-2}(a), \ldots s^{k-(n-2)}(a), s^{k-n}(a)\}$$

  y $p(a, x_n\theta) \in M_{CB}$. Pero esto es imposible, ya que si $p(a, x_n\theta) \in M_{CB}$ entonces $x_n\theta = s^0(a)$ ó $x_n\theta = s^1(a)$ y

  $$\{0, 1\} \cap \{k + n, k + n - 2, \ldots, k + 2, k, k - 2, \ldots, k - (n - 2), k - n\} = \emptyset$$

  puesto que $k > n + 1$.

- Si n es impar

  $$x_n\theta \in \{s^{k+n}(a), s^{k+n-2}(a), \ldots, s^{k+1}(a), s^{k-1}(a), \ldots s^{k-(n-2)}(a), s^{k-n}(a)\}$$

  y $p(x_n\theta, a) \in M_{CB}$. Un razonamiento análogo al del caso anterior nos lleva a contradicción, puesto que estamos suponiendo $k > n + 1$.

⊣

El estudio de procesos infinitos en aprendizaje abre la puerta a nuevas cuestiones sobre los límites del aprendizaje y la existencia de cadenas transfinitas de cláusulas que nos lleva a plantearnos cuestiones sobre la naturaleza extremadamente rica del conjunto de cláusulas. Dedicaremos a estas y otras cuestiones el siguiente capítulo.

Capítulo 7

Propiedades transfinitas

El uso de la aritmética transfinita es una herramienta importante para comprender el comportamiento de algoritmos en el límite. Un ejemplo claro lo tenemos en Programación Lógica con el operador consecuencia inmediata T_P de van Emdem y Kowalski [VK76]. Al aplicar a una interpretación I de manera iterada el operador correspondiente a un programa P vamos obteniendo la sucesión de interpretaciones $\{T_P^n(I)\}_{n\in\omega}$. Una vez formada la sucesión podemos considerar el *paso al límite* y obtener una nueva interpretación que, en general, no se puede alcanzar a partir de I aplicando el operador una cantidad finita de veces. El resultado de este salto al infinito no es más que otro elemento del retículo, al que podemos volver a aplicar el operador. En el caso del operador consecuencia inmediata el retículo considerado está formado por todas las interpretaciones de Herbrand de un lenguaje ordenado por la relación de subconjunto, pero este proceso es general en la teoría de retículos.

En el presente capítulo estudiamos propiedades transfinitas correspondientes a otro retículo. El retículo que consideramos es el conjunto de cláusulas de un lenguaje ordenadas por subsunción. Hasta donde conocemos, es la primera vez que se estudian propiedades transfinitas en el orden de subsunción.

Consideraremos sucesiones descendentes de cláusulas, esto es, sucesiones de cláusulas donde cada una de ellas es más general que la anterior y tomaremos cotas inferiores de estas sucesiones. Cualquiera de estas cotas puede ser a su vez la primera cláusula de una nueva sucesión descendente y así sucesivamente.

El diámetro de un orden es el mayor ordinal que podemos sumergir dentro de ese orden parcial y en cierto sentido responde a la pregunta de cuántas cadenas de longitud ω podemos enlazar en el orden de subsunción. Dicho diámetro dependerá

del lenguaje $\mathcal{L}$ en el que expresemos las cláusulas. En este capítulo demostramos que basta que el lenguaje tenga un símbolo de predicado binario para que el diámetro del orden de subsunción sea mayor o igual que ω^2. Informalmente, eso significa que podemos encontrar, para cualquier número natural n, n sucesiones donde la primera cláusula de cada sucesión es más general que todas las cláusulas de la sucesión anterior.

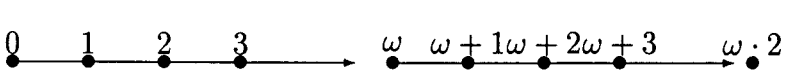
7.1 Ordinales

A continuación recordamos algunos conceptos básicos de ordinales. Una descripción detallada puede encontrarse en [HJ84].

El primer ordinal 0 se define como $\emptyset$. A continuación se definen $1 = \{\emptyset\}$, $2 = \{\emptyset, \{\emptyset\}\} = \{0, 1\}$, $3 = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} = \{0, 1, 2\}$, y así sucesivamente. Estos son los ordinales finitos. El primer ordinal infinito es ω que se define como el conjunto de todos los ordinales finitos, $\omega = \{0, 1, 2, \dots\}$. Si se consideran los ordinales $\{0, 1, 2, \dots\}$ como una sucesión de puntos, ω puede verse como el límite de la sucesión.

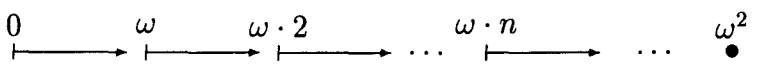
El orden natural $<$ entre ordinales se define como $\alpha < \beta \Leftrightarrow \alpha \in \beta$, por ejemplo $\alpha < \omega$ para todo ordinal finito α. Si α es un ordinal, entonces el ordinal *sucesor* de α es el ordinal $\alpha + 1 = \alpha \cup \{\alpha\}$. Se dice que un ordinal es *límite* si no es un ordinal sucesor y no es el cero. El menor ordinal límite es ω. Los siguientes ordinales después de ω son $\omega + 1 = \omega \cup \{\omega\}$, $\omega + 2 = (\omega + 1) + 1$, $\omega + 3$, ... El siguiente ordinal límite es $\omega + \omega$, que representamos como $\omega \cdot 2$.

Si vemos $\{0, 1, 2, \dots\}$ como una sucesión de puntos y ω como límite de esa sucesión, lo que hacemos es tomar ω como el origen de una nueva sucesión $\{\omega, \omega + 1, \omega + 2, \dots\}$. Esta segunda sucesión tiene como límite $\omega \cdot 2$.



Los siguientes ordinales son $(\omega \cdot 2) + 1, (\omega \cdot 2) + 2, \dots \omega \cdot 3, \omega \cdot 3 + 1, \dots, \omega \cdot 4, \dots \omega \cdot n, \dots \omega^2$. Intuitivamente ω^2 es una cadema formada por ω sucesiones, cada una

de ellas de longitud ω.



7.2 Literales

Antes de empezar nuestro estudio sobre sucesiones infinitas de cláusulas, recordemos el orden de subsunción entre literales.

Definición 7.1 *Sean L_1 y L_2 dos literales. Se dice que L_1 subsume a L_2 y lo denotaremos por $L_1 \leq L_2$ si existe una sustitución θ tal que $L_1\theta = L_2$. Si $L_1 \leq L_2$ y $L_2 \not\leq L_1$ escribiremos $L_1 < L_2$.*

En la literatura existe una gran discrepancia en cuanto a la orientación de los símbolos $\leq$ y $\succeq$ para hacer referencia a la relación de subsunción. Nosotros mantenemos esta orientación por coherencia con los textos básicos en los distintos campos, pero para un mejor seguimiento del texto, siempre escribiremos el objeto más general a la izquierda y el más específico a la derecha, así

$$\begin{aligned} C_1 \succeq C_2 &\Leftrightarrow (\exists\theta)(C_1\theta \subseteq C_2) \\ L_1 \leq L_2 &\Leftrightarrow (\exists\theta)(L_1\theta = L_2) \end{aligned}$$

Diremos que la sucesión de literales $\{L_n\}_{n=0}^{\infty}$ o la sucesión de cláusulas $\{C_n\}_{n=0}^{\infty}$ es una cadena infinita descendente si para todo $n \in \mathbb{N}$ el elemento $(n+1)$–ésimo es estrictamente más general que el elemento n-ésimo de la sucesión, esto es, se tiene que

$$\begin{aligned} \cdots < L_n < \cdots < L_1 < L_0 \\ \cdots \succ C_n \succ \cdots \succ C_1 \succ C_0 \end{aligned}$$

Análogamente, diremos que $\{L_n\}_{n=0}^{\infty}$ o la sucesión de cláusulas $\{C_n\}_{n=0}^{\infty}$ es una cadena infinita ascendente si para todo $n \in \mathbb{N}$ el elemento $(n+1)$–ésimo es estrictamente más específico que el elemento n-ésimo de la sucesión.

Veamos a continuación la función de medida de literales definida por Reynolds [Rey69] en 1970, que denotaremos por *rsize*.

Definición 7.2 *La medida del literal L se define de la siguiente manera:*

$$\begin{aligned} rsize(L) \;=\; & \textit{número de ocurrencias de símbolos en } L \\ & - \textit{número de variables distintas en } L \end{aligned}$$

donde por símbolos en L entendemos el símbolo de predicado, símbolos de función, constantes y variables, pero no el símbolo de negación si lo hubiera.

Esta función *rsize* posee propiedades muy interesantes relacionadas con la subsunción entre literales. Así, si $L_1\theta = L_2$ entonces $rsize(L_1) \leq rsize(L_2)$. De ahí que si L_1 y L_2 son variantes entonces $rsize(L_1) = rsize(L_2)$ y si se verifica $L_1\theta = L_2$ y $rsize(L_1) < rsize(L_2)$ entonces $L_1 < L_2$.

Nótese que el símbolo de negación no juega ningún papel en el tamaño de los literales negativos ni en la relación de subsunción. Podemos pensar que los literales de la forma $\neg p(t_1, \ldots, t_n)$ se reescriben como $not\_p(t_1, \ldots, t_n)$, esto es, integramos la negación dentro del símbolo de predicado. Con esta consideración un literal es sintácticamente similar a un término compuesto y podemos estudiar propiedades sintácticas del conjunto de términos en el conjunto de literales. Una de estas propiedades es la *noetherianidad* [1].

Definición 7.3 *Un orden irreflexivo y transitivo* $\langle X, < \rangle$ *es* noetheriano *si en* X *no existen cadenas infinitas descendentes, esto es, si no podemos encontrar* $\{a_n\}_{n=0}^{\infty} \subseteq X$ *tales que*

$$\cdots < a_n < \cdots < a_2 < a_1 < a_0$$

Las relaciones de subsunción estricta entre literales y cláusulas son dos ejemplos de órdenes irreflexivos y transitivos. Huet demostró en su tesis doctoral [Hue76] que el conjunto de términos de un lenguaje es noetheriano bajo subsunción, por tanto, puesto que los literales son sintácticamente iguales a los términos compuestos y la relación de subsunción entre términos es la misma que entre literales, podemos concluir que si no hay cadenas descendentes infinitas en el conjunto de términos, tampoco las hay en el conjunto de literales. Por tanto el conjunto de literales de un lenguaje es noetheriano bajo subsunción estricta.

La función *rsize* puede darnos una idea de la demostración. Si existiera una cadena infinita descendente de literales

$$\cdots < L_n < \cdots < L_2 < L_1 < L_0$$

[1] En honor de la matemática Emmy Noether.

también existiría una cadena infinita descendente de números naturales

$$\cdots < rsize(L_n) < \cdots < rsize(L_2) < rsize(L_1) < rsize(L_0)$$

pero esto es imposible.

7.3 Cláusulas

Una vez vista la noetherianidad del orden de subsunción en literales, tiene sentido preguntarse qué ocurre con la relación de subsunción en el conjunto de cláusulas, esto es, si existe una sucesión de cláusulas $\{C_n\}_{n\geq 0}$ tal que C_{n+1} sea estrictamente más general que C_n para todo n.

Si nos planteamos el problemas sobre existencia de cadenas *ascendentes*, esto es, sucesiones $\{C_n\}_{n\geq 0}$ tal que C_{n+1} sea estrictamente más específica que C_n para todo n, la respuesta es afirmativa. El siguiente ejemplo está tomado de [MDR94]

Ejemplo 7.4 Consideremos las cláuslas

$$\begin{array}{ll} C_0 \equiv & h(x_1,x_2) \leftarrow \\ C_1 \equiv & h(x_1,x_2) \leftarrow p(x_1,x_2) \\ C_2 \equiv & h(x_1,x_2) \leftarrow p(x_1,x_2), p(x_2,x_3) \\ C_3 \equiv & h(x_1,x_2) \leftarrow p(x_1,x_2), p(x_2,x_3), p(x_3,x_4) \\ \ldots & \quad \ldots \\ C_n \equiv & h(x_1,x_2) \leftarrow p(x_1,x_2), p(x_2,x_3), \ldots p(x_n,x_{n+1}) \end{array}$$

se tiene que para todo n, C_{n+1} es estríctamente más específica que que C_n en el orden de subsunción. Además, la cláusula $D \equiv h(x,x) \leftarrow p(x,x)$ verifica que $C_n \succ D$ para todo $n \geq 0$.

Otro ejemplo lo podemos encontrar en [NCdW97] donde sólo se usa un símbolo de predicado.

Ejemplo 7.5 Si tomamos $C_n = \{p(x_i,x_j) \mid i \neq j,\ 1 \leq i,j \leq n\}$ para $n \geq 2$, esto es

$$\begin{array}{ll} C_2 & = \{p(x_1,x_2), p(x_2,x_1)\} \\ C_3 & = \{p(x_1,x_2), p(x_2,x_1), p(x_1,x_3), p(x_3,x_1), p(x_2,x_3), p(x_3,x_2)\} \\ \ldots & \quad \ldots \end{array}$$

y $C = \{p(x_1, x_1)\}$ se tiene que

- $(\forall n \leq 2)[C_n \succ C_{n+1}]$
- $(\forall n \leq 2)[C_n \succ C]$

Gráficamente $C_2 \succ C_3 \succ C_4 \succ \cdots \succ C_n \succ \ldots C$

Nos planteamos a continuación qué ocurre con las cadenas infinitas descendentes, i.e. ¿podemos generalizar una cláusula indefinidamente?. La respuesta es afirmativa, esto es, el orden de subsunción entre cláusulas no es noetheriano, Nienhuys-cheng y Ronald de Wolf presentan un ejemplo en [NCdW97], que generalizaremos en la sección siguiente.

Dedicaremos esta sección a profundizar en el estudio de cadenas infinitas descendentes y a proporcionar un método nuevo para generar cadenas.

En el siguiente teorema nos planteamos estudiar propiedades generales comunes a *todas* las cadenas infinitas descendentes. Con este resultado nos alejamos de la búsqueda de una cadena concreta y fijamos nuestra atención en cuestiones más generales.

Teorema 7.6 *Sea C una cláusula y sea $\{C_i\}_{i=0}^{\infty}$ una sucesión de cláusulas tales que $C_0 = C$. Si para todo $i \geq 0$ se tiene que $C_{i+1} \succ C_i$, esto es, $\{C_i\}_{i=0}^{\infty}$ es una cadena infinita descendente, entonces existe $n_0 \in \mathbb{N}$ y un literal $L^* \in C_{n_0}$ tal que para todo $n \geq n_0$ existe una variante de L^*, L_n^*, tal que $L_n^* \in C_n$.*

Demostración:
Supongamos que no. Sea $m_0 = \max\{rsize(L) \mid L \in C_0\}$. Sea $k_1 \in \mathbb{N}$ el primer número natural tal que C_{k_1} no contiene ningún literal variante de un literal de C_0. Sea $m_1 = \max\{rsize(L) \mid L \in C_{k_1}\}$. Obviamente $m_1 < m_0$. Este razonamiento se puede repetir y sea k_2 el primer natural mayor que k_1 tal que C_{k_2} no contiene ningún literal variante de un literal de C_{k_1} y sea $m_2 = \max\{rsize(L) \mid L \in C_{k_2}\}$. Tenemos que $m_2 < m_1 < m_0$. Iterando el proceso obtendríamos una sucesión infinita descendente de números naturales $\cdots < m_r < \cdots < m_2 < m_1 < m_0$ pero esto es imposible. ⊣

El siguiente teorema, que llamaremos la regla del cuadrado, es fundamental para comprender cómo podemos producir cadenas infinitas. Antes de pasar a la formulación general, estudiaremos su importancia sobre un ejemplo sencillo. Consi-

deremos la cláusula reducida

$$C = \{p(a), q(a)\}$$

Supongamos que queremos encontrar otra cláusula reducida estrictamente más general que C. Algunas buenas candidatas son

$$C_1 = \{p(x), q(x)\} \qquad C_2 = \{p(x), q(y)\}$$
$$C_3 = \{p(x)\} \qquad C_4 = \{q(y)\} \qquad C_5 = \emptyset$$

En todas estas cláusulas los literales de C han desaparecido y los literales que aparecen son estrictamente más generales que algún literal de C. Si estas fueran las únicas cláusulas más generales que C el orden de subsunción podría ser tratado como un orden de multiconjunto basado en la relación de subsunción entre literales. Pero consideremos la cláusula

$$C^* = \{p(a), p(x), q(x)\}$$

Esta cláusula es reducida, es estrictamente más general que C y en ella aparece el literal $p(a)$, que también aparece en C. La regla del cuadrado estudia estos casos. Una simple inspección nos dice que en C^* puede aparecer $p(a)$ (o $q(a)$), pero no ambos literales simultáneamente, es decir, las cláusulas

$$C_6 = \{p(a), q(a), p(x), q(x)\} \qquad C_7 = \{p(a), q(a), p(x), q(y)\}$$
$$C_8 = \{p(a), q(a), p(x)\} \qquad C_9 = \{p(a), q(a), q(y)\}$$

son equivalentes, no más generales que C. Gráficamente, si $p(a)$ y $q(a)$ son las esquinas superiores del cuadrado, para poder introducir las inferiores $p(x)$ y $q(x)$ una esquina superior debe desaparecer, esto es, no podemos tener el cuadrado completo (Ver Fig. 7.1). Presentamos la regla del cuadrado en su forma más

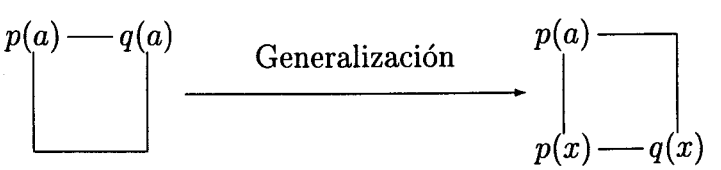


Figura 7.1: Ejemplo de generalización

general.

Teorema 7.7 (Regla del cuadrado) *Sean C_0 y C_1 dos cláusulas reducidas tales que $C_1 \succ C_0$ y sea θ una sustitución tal que $C_1\theta \subseteq C_0$. Supongamos que $\{L_1, \ldots, L_n, L\} \subseteq C_1$ son literales tales que para todo $i \in \{1, \ldots, n\}$ se tiene que $[L_i < L \wedge L_i\theta = L]$. Entonces existe un literal $L^* \in C_1 - \{L_1, \ldots, L_n, L\}$ tal que $L^*\theta \notin C_1$.*

Demostración:
Si para todo $L^* \in C_1 - \{L_1, \ldots, L_n, L\}$ se tuviera que $L^*\theta \in C_1$, se tendría que $C_1\theta \subseteq C_1$, pero como C_1 es reducida, θ debe ser un renombramiento. Pero esto está en contradicción con que $L_i < L$ y $L_i\theta = L$. ⊣

De los teoremas anteriores surge una pregunta. Del teorema 7.6 tenemos que si $\{C_i\}_{i=0}^{\infty}$ es una cadena infinita descendente, entonces existe un literal L que siempre aparece (o un literal variante suyo) a partir de un n_0, pero la regla del cuadrado nos dice que para poder mantener L en una cláusula C_i otro literal de C_i debe desaparecer. ¿Podemos mantener siempre el mismo literal L en un vértice superior del cuadrado y encontrar siempre otro para ocupar el segundo vértice superior? Esto será posible si el literal que se mantiene va generando literales para ocupar el otro vértice. De esta manera encontramos la sucesión del teorema siguiente:

Teorema 7.8 *Sea $C = \{p(a,a), p(a,x_0), q(x_0)\}$. Entonces la sucesión*

- $C_0 = C$
- $C_{2n+1} = [C_{2n} - \{p(a, x_{2n})\}] \cup \{p(x_{2n+1}, a), p(x_{2n+1}, x_{2n})\}$
- $C_{2n+2} = [C_{2n+1} - \{p(x_{2n+1}, a)\}] \cup \{p(a, x_{2n+2}), p(a, x_{2n+1}, x_{2n+2})\}$

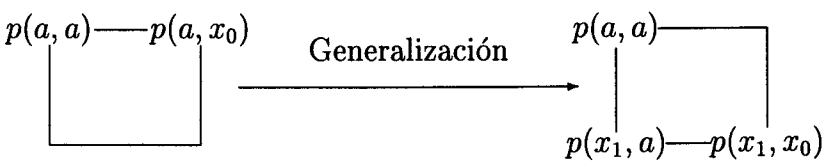
verifica lo siguiente:

1. $Var(C_n) = \{x_i\}_{i=0}^{n}$, *para todo* $n \in \mathbb{N}$
2. $(\forall n \in \mathbb{N})(\forall i < n)[p(x_i, x_{i+1}) \in C_n \vee p(x_{i+1}, x_i) \in C_n]$
3. C_n *es reducida, para todo* $n \in \mathbb{N}$
4. $C_{n+1} \succeq C_n$, *para todo* $n \in \mathbb{N}$
5. $C_n \not\preceq C_{n+1}$, *para todo* $n \in \mathbb{N}$

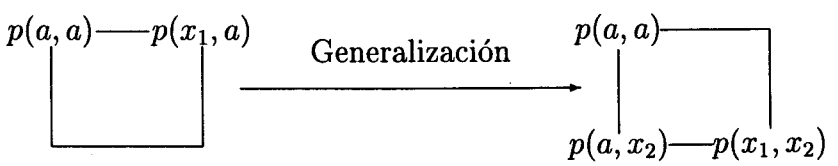
Antes de empezar, veamos cómo son las primeras cláusulas de la sucesión

$$
\begin{aligned}
C_0 &= \{\ q(x_0),\ p(a,a),\ p(a,x_0)\} \\
C_1 &= \{\ q(x_0),\ p(a,a),\ p(x_1,x_0),\ p(x_1,a)\} \\
C_2 &= \{\ q(x_0),\ p(a,a),\ p(x_1,x_0),\ p(x_1,x_2),\ p(a,x_2)\} \\
C_3 &= \{\ q(x_0),\ p(a,a),\ p(x_1,x_0),\ p(x_1,x_2),\ p(x_3,x_2),\ p(x_3,a)\} \\
C_4 &= \{\ q(x_0),\ p(a,a),\ p(x_1,x_0),\ p(x_1,x_2),\ p(x_3,x_2),\ p(x_3,x_4),\ p(a,x_4)\} \\
&\ldots
\end{aligned}
$$

Para pasar de C_0 a C_1 aplicamos la regla del cuadrado de la siguiente manera



Nótese que el literal $p(x_1,a)$ aparece en C_1 mediante una generalización del literal $p(a,a)$, pero que a su vez $p(x_1,a)$ puede tomarse como vértice superior del cuadrado para pasar de C_1 a C_2 de la siguiente manera



Demostración:
Veamos ahora la demostración del teorema

(1) Es trivial comprobar que $Var(C_0) = \{x_0\}$ y $(\forall n)\,[Var(C_{n+1}) = Var(C_n) \cup \{x_{n+1}\}]$.

(2) A partir de la definición se tiene que si un literal de la forma $p(u,v)$ con u y v variables aparece en C_i, entonces aparece en C_j para todo $j \geq i$. Si a esto añadimos que

- $(\forall n)[p(x_{2n+1}, x_{2n}) \in C_{2n+1}]$
- $(\forall n)[p(x_{2n+1}, x_{2n+2}) \in C_{2n+2}]$

el resultado se tiene.

(3) Vamos a demostrar que C_n es reducida viendo que no puede existir una sustitución θ tal que $C_n\theta \subseteq C_n$ y $C_n\theta \neq C_n$. Lo vamos a hacer de la siguiente manera: Supondremos que tal θ existe y demostraremos que para esa sustitución θ se tiene que $x_i\theta = x_i$ para todo $i \in \{1, \ldots, n\}$. Pero por el apartado (1), $Var(C_n) = \{x_0, \ldots, x_n\}$, con lo que tendríamos que $C_n\theta = C_n$ y llegaríamos a contradicción.

Sea θ tal que $C_n\theta \subseteq C_n$ y $C_n \neq C_n$. Entonces $x_0\theta = x_0$, ya que si $x_0\theta = t \neq x_0$ entonces $q(t) \in C_n\theta$ y $q(t) \notin C_n$. Tenemos entonces que $x_0\theta = x_0$, $p(x_1, x_0) \in C_n$ (si $n > 0$) y $C_n\theta \subseteq C_n$. De ahí que $p(x_1, x_0)\theta = p(x_1\theta, x_0) \in C_n$. Luego $p(x_1\theta, x_0) = p(x_1, x_0)$ y por tanto $x_1\theta = x_1$.
Vamos a probar por inducción que para todo $i > 1$ se tiene que $x_i\theta = x_i$. Consideremos $1 < i < n$ y supongamos, por hipótesis de inducción, que $\forall j \in \{1, \ldots, i\}$ se verifica que $x_j\theta = x_j$. Entonces $x_i\theta = x_i$ ya que la otra opción posible es $x_i\theta = x_{i-2}$, pero si esto ocurre entonces se tiene que $\forall k \geq i$ se tiene que $x_k\theta = x_m$ con $m < k$. En particular $x_n\theta \in Var$ y $x_n\theta \neq x_n$. Contradicción.

(4) Basta considerar $\theta_n = \{x_n/a\}$. Veamos que con esas sustituciones $C_{n+1}\theta_{n+1} = C_n$. Por (1) se tiene que $Var(C_n = \{x_0, \ldots, x_n\}$, luego

$$\begin{aligned} C_{2n+1}\theta_{2n+1} &= ([C_{2n} - \{p(a, x_{2n})\}] \cup \{p(x_{2n+1}, a), p(x_{2n+1}, x_{2n})\})\theta_{2n+1} \\ &= [C_{2n} - \{p(a, x_{2n})\}] \cup \{p(a, a), p(a, x_{2n})\} \\ &= C_{2n} \end{aligned}$$

$$\begin{aligned} C_{2n+2}\theta_{2n+2} = {} & ([C_{2n+1} - \{p(x_{2n+1}, a)\}] \cup \\ & \{p(a, x_{2n+2}), p(a, x_{2n+1}, x_{2n+2})\})\theta_{2n+2} = \\ = {} & [C_{2n+1} - \{p(x_{2n+1}, a)\}] \cup \{p(a, a), p(x_{2n+1}, a)\} = \\ = {} & C_{2n+1} \end{aligned}$$

(5) Veamos que no puede existir una sustitución θ tal que $C_n\theta \subseteq C_{n+1}$. Supongamos que tal θ existe y llegaremos a que entonces $x_n\theta = x_n$ y esto es una contradicción con que $C_n\theta \subseteq C_{n+1}$, ya que si $C_n\theta \subseteq C_{n+1}$, necesariamente $x_n\theta = a$.
Razonando de manera análoga al apartado (3) tenemos que $x_0\theta = x_0$, ya que si $x_0\theta = t \neq x_0$ entonces $q(t) \in C_n\theta$ y $q(t) \notin C_{n+1}$. De igual manera, puesto que $x_0\theta = x_0$ se tiene que $p(x_1, x_0)\theta = p(x_1\theta, x_0) \in C_n$. De ahí que

$p(x_1\theta, x_0) = p(x_1, x_0)$ y por tanto $x_1\theta = x_1$.
Vamos a probar ahora por inducción que $\forall i > 1$ también se verifica que $x_i\theta = x_i$. Consideremos $1 < i \leq n$ y supongamos, por hipótesis de inducción, que $\forall j \in \{1, \ldots, i\}$ se verifica que $x_j\theta = x_j$. Entonces para x_i hay dos posibilidades: $x_i\theta = x_i$ o $x_i\theta = x_{i-2}$. Pero si $x_i\theta = x_{i-2}$ entonces $\forall k \geq i$ se tiene que $x_k\theta \in Var$. En particular $x_n\theta \in Var$. Contradicción.
Hemos visto que si $\forall j \in \{1, \ldots, i\}$ se verifica que $x_j\theta = x_j$ entonces $x_i\theta = x_i$, y esto es cierto para $1 < i \leq n$. Por consiguiente $x_n\theta = x_n$ y esto es una contradicción con que $C_n\theta \subseteq C_{n+1}$

⊣

En esta sección hemos estudiado algunas propiedades generales de las cadenas inifinitas descendentes en el orden de subsunción sobre cláusulas. En esas sucesiones $\{C_n\}_{n\geq 0}$, se tiene que $C_{n+1} \succ C_n$, y en cierto sentido, podemos considerar que hemos sumergido el ordinal ω en el orden determinado por la subsunción en el conjunto de cláusulas.

Pero puede ocurrir que exista una cláusula D tal que $D \succ C_n$ para todo $n \geq 0$, por ejemplo, la cláusula $D = \{q(x_0), p(a, a)\}$ es más general que todas las cláusulas de la sucesión del teorema 7.8. Tales cláusulas pueden ser el origen de una nueva sucesión, con lo que tendríamos una segunda copia de ω sumergida en el orden de subsunción, esto es, habríamos sumergido el ordinal $\omega \cdot 2$. ¿Cuántas copias de ω podemos sumergir en este orden de subsunción? Dedicaremos a estudiar esta cuestión en la siguiente sección.

7.4 Diámetro del orden de subsunción

Empezamos nuestro estudio recordando algunas definiciones sobre relaciones entre órdenes.

Definición 7.9 *Sean A y B dos conjuntos y R_A, R_B dos relaciones binarias irreflexivas y transitivas sobre A y B respectivamente. Se dice que la aplicación $\phi : \langle A, R_A\rangle \to \langle B, R_B\rangle$ es un morfismo si $(\forall x, y \in A)\,[xR_Ay \to \phi(x)R_B\phi(y)]$.*

Definición 7.10 *Sea A un conjunto y R_A una relación binaria irreflexiva y transitiva sobre A. Se dice que el ordinal α puede sumergirse en $\langle A, R_A\rangle$ si existe un morfismo*

$$\phi : \langle \alpha, < \rangle \to \langle A, R_A\rangle$$

Ilustramos estas definiciones con el siguiente ejemplo.

Ejemplo 7.11 Sea $X = \{a, b, c\}$ y sea $\mathcal{P}(X)$ es conjunto de las partes de X, esto es $\mathcal{P}(X) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$ con la relación irreflexiva y transitiva *ser subconjunto estricto* $\subset$. Entonces el ordinal $3 = \{0, 1, 2\}$ puede sumergirse en $\langle \mathcal{P}(X), \subset \rangle$ por ejemplo mediante el morfismo

$$\begin{array}{rccl} \phi: & \langle 3, < \rangle & \to & \langle \mathcal{P}(X), \subset \rangle \\ & 0 & \mapsto & \phi(0) = \emptyset \\ & 1 & \mapsto & \phi(1) = \{a\} \\ & 2 & \mapsto & \phi(2) = \{a, b\} \end{array}$$

Damos a continuación la definición de diámetro.

Definición 7.12 *Sea A un conjunto y R_A una relación binaria irreflexiva y transitiva sobre A. Se define el* diámetro *de $\langle A, R_A \rangle$ como el supremo del conjunto de ordinales que puede sumergirse en $\langle A, R_A \rangle$.*

Por ejemplo, si $X = \{a, b, c\}$, entonces el diámetro de $\langle \mathcal{P}(X), \subset \rangle$ es 4.

El diámetro del conjunto de cláusulas junto con el orden de subsunción estricta ($\succ$) dependerá del lenguaje $\mathcal{L}$ con el que expresemos las cláusulas. En esta sección proporcionamos una cota inferior para el diámetro del conjunto de cláusulas con esta relación en el caso en que el lenguaje tenga al menos un símbolo de predicado binario.

El objetivo de esta sección es probar el siguiente resultado

Teorema 7.13 *Sea $\mathcal{L}$ un lenguaje con al menos un símbolo de predicado binario y sea $\langle \mathbb{C}, \succ \rangle$ el conjunto de cláusulas de $\mathcal{L}$ con el orden de subsunción $\succ$. Entonces el diámetro de ese orden está acotado inferiormente por ω^2*

Lo vamos a demostrar dando un morfismo ψ_k para cada $k \in \omega$

$$\psi_k : \langle \omega \cdot k, < \rangle \longrightarrow \langle \mathbb{C}, \succ \rangle$$

esto es, vamos a dar k sucesiones $\{C_n^0\}_{n\in\omega}$, $\{C_n^1\}_{n\in\omega}$,$\ldots$,$\{C_n^{k-1}\}_{n\in\omega}$, tales que

$$\cdots \succ C_{m_i}^i \succ \cdots \succ C_{m_1}^1 \succ \cdots \succ C_1^1 \succ C_0^1 \succ \cdots \succ C_{m_0}^0 \succ \ldots C_1^0 \succ C_0^0$$

Puesto que para cada $k \in \omega$ se tiene que $\omega \cdot k$ puede sumergirse en $\langle \mathbb{C}, \succ \rangle$, el diámetro de $\langle \mathbb{C}, \succ \rangle$ será al menos ω^2.

Para demostrar el resultado vamos a generalizar la construcción de una cadena infinita de [NCdW97]. Empezamos la construcción con la definición de ciclo.

Definición 7.14 (Adaptada de 14.33 en [NCdW97]) *Diremos que una cláusula C es un* ciclo *de longitud n si existe un renombramiento de variables θ de manera que*

$$C\theta = \{p(x_1, x_2), p(x_2, x_3), \ldots, p(x_{n-1}, x_n), p(x_n, x_1)\}$$

donde $x_1, \ldots, x_n$ son variables distintas.

En esta sección consideraremos siempre que los ciclos de distinta longitud tienen las variables separadas. Para simplificar la notación consideraremos siempre que la variable x_i^n es la variable i–ésima del ciclo de longitud n y escribiremos el ciclo de longitud n como

$$C_n = \{p(x_1^n, x_2^n), p(x_2^n, x_3^n), \ldots, p(x_{n-1}^n, x_n^n), p(x_n^n, x_1^n)\}$$

En [NCdW97], Nienhuys–Cheng y de Wolf usaron los ciclos para encontrar una sucesión de cláusulas de longitud ω. En el lenguaje de órdenes esto significa que el diámetro del orden de subsunción es al menos ω. En nuestra construcción generalizamos su idea para obtener como cota ω^2.

El siguiente resultado es el lema 14.36 en [NCdW97].

Lema 7.15 *Si $n = m \cdot k$ para algún $k > 1$, entonces $C_n \succ C_m$.*

En el siguiente resultado probamos que el inverso del lema 7.15 también es cierto. En realidad probamos un resultado un poco más general.

Lema 7.16 *Si n y m son primos entre sí, los ciclos C_n y C_m son cláusulas incomparables bajo subsunción.*

Demostración:
Supongamos $n < m$. Entonces $C_n \not\succeq C_m$ puesto que C_m no contiene ningún ciclo de longitud n. Veamos que $C_m \not\succeq C_n$. Supongamos que existe θ tal que $C_m\theta \subseteq C_n$. Sin pérdida de generalidad podemos considerar que C_n y C_m son las cláusulas

$$C_n = \{p(x_1, x_2), \ldots, p(x_{n-1}, x_n), p(x_n, x_1)\}$$
$$C_m = \{p(y_1, y_2), \ldots, p(y_{n-1}, y_n), p(y_n, y_{n+1}), \ldots, p(y_m, y_1)\}$$

También sin pérdida de generalidad podemos suponer que si $i \in \{1, \ldots, n\}$ se tiene que $y_i\theta = x_i$. Por tanto tenemos que $p(y_n, y_{n+1})\theta = p(x_n, x_1)$ y de ahí que $y_{n+1}\theta = x_1$. En general, para todo $i \in \{1, \ldots, n\}$ y $k \geq 0$ apropiados se tiene que $y_{k \cdot n+i}\theta = x_i$. Pero n y m son primos entre sí, luego deben existir $k \geq 0$ y $i_0 \neq m$ tal que $m = k \cdot n + i_0$ y por tanto $y_m\theta = x_{i_0}$ con $i_0 \neq m$. Pero esto es imposible, ya que entonces

$$p(y_m, y_1)\theta = p(x_{i_0}, x_1) \notin C_n$$

y esto termina la demostración. ⊣

A continuación vamos a hacer unión disjunta de ciclos, para lo que usaremos el operador $\uplus$. Diremos que $C_n \uplus C_m$ es la unión disjunta de los ciclos C_n y C_m si es el conjunto de literales formado por los literales de C_n y C_m.

En la demostración del teorema 7.13, las sucesiones que damos están formadas por la unión disjunta de ciclos. En el siguiente teorema damos una condición suficiente para que la unión disjunta de ciclos sea reducida.

Teorema 7.17 *La unión disjunta de los ciclos $C_{n_1}, C_{n_2}, \ldots, C_{n_k}$, esto es, la cláusula $C_{n_1} \uplus C_{n_2} \uplus \cdots \uplus C_{n_k}$ es reducida si sus longitudes $n_1, n_2, \ldots n_k$ son primos entre sí dos a dos.*

Demostración:
Puesto que estamos haciendo unión de cláusulas reducidas si existiera una sustitución θ tal que

$$(C_{n_1} \uplus C_{n_2} \uplus \cdots \uplus C_{n_k})\theta \subseteq C_{n_1} \uplus C_{n_2} \uplus \cdots \uplus C_{n_k}$$

$$(C_{n_1} \uplus C_{n_2} \uplus \cdots \uplus C_{n_k})\theta \neq C_{n_1} \uplus C_{n_2} \uplus \cdots \uplus C_{n_k}$$

entonces deberían existir $i, j \in \{1, \ldots, k\}$, $i \neq j$ tales que un literal de C_{n_i} se aplicara en un literal de C_{n_j}. Pero debido a la estructura de ciclo de C_{n_i}, si esto ocurriera entonces todos los literales de C_{n_i} se aplicarían en literales de C_{n_j}, esto es $C_{n_i}\theta \subseteq C_{n_j}$. Pero esto es imposible por el lema 7.16. ⊣

Una consecuencia directa del lema 7.15 es que dado $n > 2$ podemos obtener una cadena infinita descendente de ciclos ordenados por subsunción de longitudes $n, n^2, n^3, \ldots$

$$\cdots \succ C_{n^m} \succ \cdots \succ C_{n^3} \succ C_{n^2} \succ C_n$$

Vamos a generalizar ese resultado

Teorema 7.18 *Sean $C_{n_1}, C_{n_2}, \ldots, C_{n_k}$ k ciclos tales que sus longitudes $n_1, n_2, \ldots, n_k$ son primos entre sí dos a dos. Llamaremos D_1 a su unión disjunta*

$$D_1 = C_{n_1} \uplus C_{n_2} \uplus \cdots \uplus C_{n_k}$$

Sean $r > 1$ e $i \in \{1, \ldots, k\}$. Sea $(n_i)^r$ la r–ésima potencia de n_i y consideremos el ciclo $C_{(n_i)^r}$, esto es, el ciclo de longitud $(n_i)^r$. Sea D_2 la cláusula obtenida a partir de D_1 por la sustitución de C_{n_i} por $C_{(n_i)^r}$, esto es,

$$D_2 = C_{n_1} \uplus \cdots \uplus C_{n_{i-1}} \uplus C_{(n_i)^r} \uplus C_{n_{i+1}} \uplus C_{n_k}$$

Entonces $D_2 \succ D_1$.

Demostración:
Veamos primero que $D_2 \succeq D_1$. Para ello basta encontrar una sustitución θ tal que $D_2\theta \subseteq D_1$. Consideremos una susitución θ tal que

- $z\theta = z$ para toda variable z que ocurra en $D_2 - C_{(n_i)^r}$
- Si $C_{n_i} = \{p(x_1, x_2), \ldots, p(x_{n_i}, x_1)\}$ y $C_{(n_i)^r} = \{p(y_1, y_2), \ldots, p(y_{(n_i)^r}, y_1)\}$ entonces para $k \geq 0$ y para tod $j \in \{1, \ldots, n_i\}$ se tiene que $y_{k \cdot n_i + j}\theta = x_i$

Una simple comprobación nos da que con esa θ se tiene que $D_2\theta \subseteq D_1$.

Veamos que $D_1 \not\succeq D_2$. Supongamos ahora que existe una sustitución θ tal que $D_1\theta \subseteq D_2$. Por un razonamiento análogo al del teorema anterior se tiene que ningún literal de C_{n_p} puede aplicarse sobre un literal de C_{n_q} si n_p y n_q son primos entre sí, luego se tendría

- $C_{n_j}\theta \subseteq C_{n_j}$ $\forall j \in \{1, \ldots, i-1, i+1, \ldots, k\}$
- $C_{n_i}\theta \subseteq C_{(n_i)^r}$

Pero esto último es imposible ya que $C_{(n_i)^r}$ tendría que contener un ciclo de longitud menor o igual que n_i. ⊣

Demostremos por último el teorema 7.13.
Demostración:
Se define para cada $k \in \omega$ el morfismo

$$\psi_k : \langle \omega \cdot k, < \rangle \longrightarrow \langle \mathbb{C}, \succ \rangle$$

de la siguiente manera: Sean $n_0 = 2, n_1 = 3, n_2 = 5, \ldots, n_{k-1}$ los k primeros números primos. Entonces, para todo $j \in \{0, 1, \ldots, k-1\}$ y todo $p \in \omega$ se define

$$\psi_k(\omega j + p) = C_{(n_j)^{p+1}} \uplus C_{n_{j+1}} \uplus C_{n_{j+2}} \uplus \cdots \uplus C_{n_k}$$

De esta manera, la primera sucesión de cláusulas, correspondiente a los valores $\{0, 1, 2, \ldots\}$ son

$$\begin{array}{lcllllll}
\psi_k(0) & = & C_{n_0} & \uplus & C_{n_1} & \uplus & \ldots\ldots & \uplus\ C_{n_{k-1}} \\
\psi_k(1) & = & C_{(n_0)^2} & \uplus & C_{n_1} & \uplus & \ldots\ldots & \uplus\ C_{n_{k-1}} \\
\ldots & & \ldots & & \ldots & & & \\
\psi_k(p) & = & C_{(n_0)^{p+1}} & \uplus & C_{n_1} & \uplus & \ldots\ldots & \uplus\ C_{n_{k-1}} \\
\ldots & & \ldots & & \ldots & & &
\end{array}$$

esto es, para pasar de $\psi_k(i)$ a la siguiente cláusula $\psi_k(i+1)$ sólo sustituimos el ciclo de longitud $(n_0)^{i+1}$, esto es, $C_{(n_0)^{i+1}}$, por el ciclo de longitud $(n_0)^{i+2}$, esto es, $C_{(n_0)^{i+2}}$. Con lo que tenemos que $\psi_k(i+1)$ es más general que $\psi_k(i)$. Para el primer ordinal límite ω tenemos la cláusula

$$\psi_k(\omega) = C_{n_1} \uplus \cdots \uplus C_{n_{k-1}}$$

que está contenida en todas las anteriores, y por tanto es más general que ellas. A continuación, podemos realizar el mismo proceso a partir de $\psi_k(\omega)$ y tener

$$\begin{array}{lcllllll}
\psi_k(\omega) & = & C_{n_1} & \uplus & C_{n_2} & \uplus & \ldots\ldots & \uplus\ C_{n_{k-1}} \\
\psi_k(\omega+1) & = & C_{(n_1)^2} & \uplus & C_{n_2} & \uplus & \ldots\ldots & \uplus\ C_{n_{k-1}} \\
\ldots & & \ldots & & \ldots & & & \\
\psi_k(\omega+p) & = & C_{(n_1)^{p+1}} & \uplus & C_{n_2} & \uplus & \ldots\ldots & \uplus\ C_{n_{k-1}} \\
\ldots & & \ldots & & \ldots & & &
\end{array}$$

que correspondería a la segunda sucesión de longitud ω y llegaríamos al segundo ordinal límite

$$\psi_k(\omega 2) = C_{n_2} \uplus \cdots \uplus C_{n_{k-1}}$$

que está contenida en todas las anteriores y por tanto las subsume a todas. Por último, en el caso general se tiene que

$$\psi_k(\omega j + p) = C_{(n_j)^{p+1}} \uplus C_{n_{j+1}} \uplus C_{n_{j+2}} \uplus \cdots \uplus C_{n_k}$$

Capítulo 8

Conclusiones y trabajo futuro

8.1 Resumen de resultados

En esta memoria hemos estudiado los procesos de generalización, el paso de lo particular a lo general, cuando la información está expresada en lenguaje clausal. Para ello hemos definido unos operadores adaptados a los distintos órdenes de generalidad. Ha sido necesario compaginar adecuadamente los distintos niveles en los que se produce la generalización: términos, literales, cláusulas y programas. La solución propuesta se apoya en la utilización de conjuntos de posiciones, conjuntos de literales y conjuntos de cláusulas, es decir, los operadores actúan sin necesidad de considerar órdenes sobre los literales de una cláusula o entre las cláusulas de un programa.

Las principales aportaciones realizadas en esta memoria han sido:

- La definición de una nueva familia de operadores, los *Operadores Clausales*, con la propiedad de ser operadores universales, esto es, que dadas dos cláusulas cualesquiera C_1 y C_2 podemos alcanzar C_2 desde C_1 mediante la sucesiva aplicación de estos operadores.

- La definición de los *Operadores de Aprendizaje para la Subsunción (OAS).* Estos operadores son un subconjunto del conjunto de *Operadores Clausales* y su principal propiedad es que representan una *caracterización* mediante operadores de la relación de subsunción entre cláusulas, esto es, dadas dos cláusulas cualesquiera C_1 y C_2, se verifica que C_1 subsume a C_2 si y sólo si podemos obtener C_1 a partir de C_2 mediante la aplicación de una cadena de OAS.

- La definición de una quasi-métrica sobre el conjunto de cláusulas que permite *cuantificar* la proximidad entre cláusulas basada en la relación de subsunción.
- Un algoritmo para calcular dicha quasi-métrica
- Una fórmula para una rápida estimación de esta quasi-métrica que permite reducir costes computacionales.
- La definición de operadores de generalización para el orden de derivación por resolución: Los *Operadores de Inversión Sesgados (OIS)*. Una apropiada combinación de estos operadores junto con los Operadores de Aprendizaje para la Subsunción nos permiten generalizar una cláusula D para obtener la cláusula C cuando $C \models D$.
- La definición de los *Operadores de Generalización Minimales (OGM)*. Estos representan las unidades mínimas de generalización clausal de manera que si C y D son cláusulas y están relacionadas por alguna de las tres relaciones estudiadas: subsunción, derivación o consecuencia, entonces podemos obtener C a partir de D mediante una combinación apropiada de OGM.
- La definición de operadores de generalización adaptados a la subsunción entre programas: los *OAS compuestos*. Estos operadores son extensiones a programas de los Operadores de Aprendizaje para la Subsunción (OAS) y de manera análoga a como ocurría con los OAS, también representan una *caracterización* de la relación de subsunción entre programas.
- La definición de una *métrica débil (una pseudo-quasi-distancia)* para cuantificar la proximidad entre programas basada en estos operadores.
- Un método de cálculo de esta distancia débil.
- La definición de un orden de aprendizaje entre programas definidos mediante sus menores modelos de Herbrand.
- La relación entre el operador de consecuencia de Kowalski, el aprendizaje clausal, la relación de subsunción y los operadores que hemos definido para programas (OAS compuestos).
- Diversos resultados relacionados con cadenas infinitas de cláusulas y programas.

- Una cota para el diámetro del conjunto de cláusulas por la relación de subsunción.

8.2 Discusión y perspectivas

La correspondencia estudiada entre órdenes de generalización y operadores clausales nos lleva a plantearnos otras cuestiones interesantes para estudiar en trabajos futuros. Los órdenes de generalidad estudiados –subsunción, derivación y consecuencia para las cláusulas y subsunción para programas– son relaciones binarias para las cuales hemos definido distintas familias de operadores adaptadas a estas relaciones. La pregunta que surge de manera natural es si podemos caracterizar otras relaciones decidibles (o semidecidibles) sobre cláusulas o programas mediante un subconjunto apropiado de operadores clausales y si podemos usar las distancias débiles definidas para encontrar *mínimos* en los procesos de transformación de cláusulas o programas.

Como estudio complementario al trabajo realizado cabe plantearse en primer lugar la relación *dual* a la generalización: la *especialización* en aprendizaje clausal. Cuando una teoría es demasiado general, esto es, cuando la hipótesis actual cubre instancias negativas, el concepto debe ser especializado. Una línea para continuar el trabajo realizado es el estudio de órdenes de especialización entre cláusulas y programas mediante operadores adecuados.

Pero la metodología presentada puede abarcar un abanico más amplio de relaciones e ir más allá del ámbito del aprendizaje. Cuando representamos conocimiento mediante cláusulas y programas lógicos, la necesidad de actualizar o simplemente transformar programas para adaptarlos a situaciones nuevas nos lleva a la búsqueda de operadores clausales adaptados a esas transformaciones. El trabajo realizado en la definición de estos operadores[1] puede servir de soporte para la definición de otros operadores clausales vinculados a nuevas relaciones entre programas.

Otra línea para continuar con la investigación es el estudio de otras propiedades relacionadas con los operadores ya definidos. Es posible definir cuándo un operador es más general que otro y estudiar tanto relaciones cualitativas del conjunto de operadores como cuantitativas relacionadas con la *proximidad* entre

[1]Una versión previa de los operadores de esta memoria fue presentada en CL-2000 [GNAJBD00].

operadores.

Otro camino a seguir es la profundización en el estudio de los procesos infinitos. La existencia de cadenas infinitas muestra la complejidad intrínseca del aprendizaje y la idea de límite nos lleva a plantearnos cuestiones topológicas sobre estos conjuntos. ¿Podemos estudiar el aprendizaje clausal en términos topológicos? En su artículo *Language Identification in the Limit* [Gol67], E.M. Gold presenta un modelo de aprendizaje, ya clásico, en el que al aumentar el número de observaciones la sucesión de hipótesis obtenida *converge* al *concepto objetivo.* La relación entre el paradigma de Gold y las cadenas infinitas de programas donde cada programa se obtiene del anterior mediante operadores clausales surge de manera natural. Y el marco en el que modelamos el paradigma de aprendizaje en el límite de Gold debe ser topológico. Sin duda esta modelización es una de las líneas más prometedoras para la continuación del trabajo realizado (una primera aproximación la tenemos en [GNAJBD00]).

Otra vía de investigación consiste en la adaptación de las relaciones de proximidad estudiadas a sistemas reales. Sistemas de Programación Lógica Inductiva como Progol [Mug95] o Aleph [Sri01] están en continua expansión y pueden integrar nuestras métricas débiles como una opción para el estudio de proximidad.

Apéndice A

Breve historia de los operadores de refinamiento

El estudio de operadores de transformación para cláusulas ha estado ligado a la historia de la PLI desde el principio y, en sentido amplio, podemos pensar que todos los algoritmos usados en PLI no son más que distintas adaptaciones de esta idea de transformación.

No obstante, en la literatura, no hay muchos autores que se hayan dedicado a estudiar de manera sistemática estos operadores, llamados *de refinamiento.* En esta sección repasamos brevemente la evolución de la definición de estos *operadores de refinamiento*[1].

En 1981, Shapiro, [Sha81] introdujo la noción de inferencia de modelos. Dada una sucesión de ejemplos positivos y negativos de un concepto desconocido, su sistema MIS *(Model Inference System)* buscaba una teoría que pudiera inferir todos los ejemplos positivos y ninguno de los negativos. El sistema usa esencialmente dos técnicas: Si podemos inferir de la teoría un ejemplo negativo, entonces usamos el retroceso *(backtracing)* y buscamos una hipótesis que lo refute. Si encontramos un ejemplo positivo que no pueda ser inferido por la teoría, entonces aplicamos unos *operadores de refinamiento.* Sus operadores están definidos sobre *cláusulas reducidas* en un lenguaje de primer orden donde las cláusulas y los operadores de refinamiento están limitados por una medida de complejidad.

Vemos a continuación su definición de operador, pero antes necesitamos algunas definiciones previas:

[1]La principal referencia para esta sección ha sido [PvdL92].

Definición A.1 *[Sha81] Diremos que la sustitución θ* no hace decrecer *la cláusula C si $|C\theta| = |C|$.*

Definición A.2 *[Sha81] Un literal L es más general que M con respecto a la cláusula C si existe una sustitución θ tal que $L\theta = M$ y $C\theta = C$.*

A continuación presentamos la definición de operador de Shapiro.

Definición A.3 *[Sha81] Sea C una cláusula reducida de $\mathcal{L}$. Entonces $D \in \rho_0(C)$ cuando se da exactamente una de las siguientes condiciones:*

(ρ_0^1) *$D = C\theta$ donde $\theta = \{x/y\}$ no hace decrecer C y ambas variables ocurren en C.*

(ρ_0^2) *$D = C\theta$ donde $\theta = \{x/f(y_1, \ldots, y_n)\}$ no hace decrecer C, f es un símbolo de función de aridad n, x ocurre en C y las y_i son variables distintas que no ocurren en C.*

(ρ_0^3) *$D = C \cup \{L\}$ donde L es un literal de máxima generalidad con respecto a C para el cual $C \cup \{L\}$ es reducida.*

En [vdLNC98] Patrick R. J. van der Laag y Shan-Hwei Nienhuys–Cheng estudian estos operadores y detectan una imprecisión en el trabajo de Shapiro en el estudio de las propiedades teóricas de su operador. La intención de Shapiro era probar que para todo par de cláusulas reducidas C y D tales que $C \subseteq D$ ó $C\theta = D$ donde θ es no decreciente, existe una cadena de cláusulas reducidas entre C y D de manera que cada una de ellas se obtiene de la anterior aplicando el operador de refinamiento. Nienhuys y van der Laag presentaron el siguiente ejemplo en [vdL92] y probaron que en ese caso la afirmación de Shapiro no es cierta.

Ejemplo A.4 Consideremos las siguientes cláusulas:

$$C = \{p(X_1, X_2), p(X_2, X_3), p(X_3, X_1)\}$$
$$D = \{p(X_1, X_2), p(X_2, X_3), p(X_3, X_1), p(Z_1, Z_2), p(Z_2, Z_1)\}$$

C y D son cláusulas reducidas, $C \subseteq D$, luego C subsume a D y no existe una cadena de cláusulas reducidas de C a D en la cual cada cláusula se obtenga de la anterior por aplicación del operador. No obstante, si consideramos la cláusula C_1, equivalente a C por subsunción

$$C_1 = \{p(X_1, X_2), p(X_2, X_3), p(X_3, X_1), p(Z_1, Z_2), p(Z_2, Z_3)\}$$

podemos obtener D aplicando a C_1 la sustitución $\theta = \{Z_3/Z_1\}$

La clave del problema estriba en que es necesario considerar cláusulas que no sean reducidas dentro de la cadena de cláusulas que nos lleva de C a D. Para subsanar esta limitación de los operadores definidos por Shapiro, Nienhuys–Cheng y van der Laag ofrecen una nueva definición de operador

Definición A.5 ([PvdL92]) *Sea C una cláusula reducida y sea $[C]$ el conjunto de cláusulas equivalentes a C bajo subsunción. Entonces $D \in \rho_r(C)$ si D es reducida y se verifica una de las siguientes condiciones:*

- ρ_r^1: *$C \succ D$ y existen dos cláusulas $C' \in [C]$ y $D' \in [D]$ tales que $C'\theta = D'$, donde θ es la sustitución $\theta = \{x/f(y_1, \ldots, y_n)\}$, f es un símbolo de función*[2], *x ocurre en C', y las variables $y_1, \ldots, y_n$ son variables distintas que no ocurren en C'.*
- ρ_r^2: *$C \succ D$ y existen dos cláusulas $C' \in [C]$ y $D' \in [D]$ tales que $C'\theta = D'$, donde $\theta = \{x/y\}$ y además las variables x e y ocurren en C'.*
- ρ_r^3: *$D = C \cup \{L\}$ donde L sólo tiene variables distintas que no ocurren en C y para todo literal $M \in C$, L difiere de M en el símbolo de predicado o en el signo.*

Ilustramos la definición anterior con el siguiente ejemplo:

Ejemplo A.6 Consideremos la cláusula

$$encendida(x) \leftarrow bombilla(x), cable(x, y), corriente(y)$$

que expresada como conjunto de literales es

$$C = \{encendida(x), \neg bombilla(x), \neg cable(x, y), \neg corriente(y)\}$$

Si esta cláusula fuera demasiado general podemos hacerla más específica aplicando alguno de los operadores de refinamiento de [PvdL92]. Por el operador ρ_r^1, si aplicamos la sustitución $\theta_1 = \{y/bateria\}$, donde *bateria* es un símbolo de función de aridad cero, tendríamos la cláusula

$$C\theta_1 = \{encendida(x), \neg bombilla(x), \neg cable(x, bateria), \neg corriente(bateria)\}$$

[2] *Se consideran las constantes como símbolos de función de aridad cero.*

Por el operador ρ_r^2, si aplicamos la sustitución $\theta_2 = \{y/x\}$, tendríamos la cláusula

$$C\theta_2 = \{encendida(x), \neg bombilla(x), \neg cable(x,x), \neg corriente(x)\}$$

El operador ρ_r^3 es más técnico. Nos permite añadir literales nuevos como

$$no\_fundida(z)$$

y conseguir cláusulas como

$$C_1 = \{encendida(x), \neg bombilla(x), \neg cable(x,y), \neg corriente(y), no\_fundida(z)\}$$

para después aplicar otros operadores, por ejemplo ρ_r^3 mediante la sustitución $\theta_3 = \{z/x\}$ y obtener

$$C_1\theta_3 = \left\{ \begin{array}{lll} encendida(x), & \neg bombilla(x), & \neg cable(x,y), \\ \neg corriente(y), & no\_fundida(x) & \end{array} \right\}$$

Van der Laag y Nienhuys–Cheng probaron [PvdL92] que si C y D son cláusulas reducidas y $C \succ D$ entonces existía una ρ_r-cadena de C a D. De este modo salvaban la imprecisión encontrada en el trabajo de Shapiro.

Los operadores presentados hasta ahora son operadores *descendentes* para el orden de subsunción, puesto que si $D \in \rho(C)$, entonces $C \succeq D$. En [NCdW97] encontramos el dual de estos operadores: Los operadores ascendentes.

En su definición Nienhuys-Cheng y de Wolf hacen uso de una estructura asociada a una cláusula C y a un término t, llamada $dup(C,t)$, en la que aparecen literales repetidos y la noción de orden. Más adelante veremos que en nuestra definición de operador no es necesaria una estructura similar.

Definición A.7 *[NCdW97] Sea $C = \{L_1, \ldots, L_n\}$ una cláusula y t un término que ocurre en C. Supongamos que t ocurre k_1 veces en L_1, k_2 veces en L_2, etc ... Entonces $dup(C,t) = \vec{C}$ es una cláusula ordenada consistente en 2^{k_1} copias de L_1, 2^{k_2} copias de L_2, ..., 2^{k_n} copias de L_n*

Definición A.8 *[NCdW97] Un literal $p(x_1, \ldots, x_n)$ o $\neg p(x_1, \ldots, x_n)$ es de máxima generalidad respecto a una cláusula C si $x_1, \ldots, x_n$ son variables distintas que no ocurren en C.*

La definición de operador ascendente también es por casos:

Definición A.9 *[NCdW97] Sea $\mathbb{C}$ el conjunto de cláusulas del lenguaje. El operador de refinamiento ascendente δ_u para $\langle \mathbb{C}, \succeq \rangle$ se define como sigue:*

1. *Para todo $t = f(x_1, \ldots, x_n)$ de C para el cual todas las x_i son variables distintas y cada ocurrencia de x_i en C es dentro de t, $\delta_u(C)$ contiene la cláusula obtenida reemplazando todas las ocurrencias de t en C por alguna nueva variable z que no estuviera previamente en C.*

2. *Para toda constante a de C y cada subconjunto no vacío del conjunto de ocurrencias de a en $\vec{C} = dup(C, a)$, si $\vec{D}$ es la cláusula ordenada obtenida reemplazando estas ocurrencias de a en C por la nueva variable z, entonces $\delta_u(C)$ contiene a D (D es el conjunto de literales de la cláusula ordenada $\vec{D}$).*

3. *Para toda variable x de C y todo subconjunto propio no vacío del conjunto de ocurrencias de a en $\vec{C} = dup(C, x)$, si $\vec{D}$ es la cláusula ordenada obtenida reemplazando estas ocurrencias de x en C por la nueva variable z, entonces $\delta_u(C)$ contiene a D.*

4. *Si $C = D \cup \{L\}$ y L es un literal de máxima generalidad respecto a D, entonces $\delta_u(C)$ contiene a D.*

Apéndice B

Programas

En este apéndice presentamos el código para SWI-Prolog de las definiciones, algritmos y relaciones más significativas presentadas en la memoria.

En todos los programas de esta memoria distinguiremos entre las variables del lenguje y del metalenguaje. En este caso en metalenguaje es Prolog y expresaremos sus variables de la manera habitual. Las variables del lenguaje objeto las representaremos con la notación `xN` donde `N` es un número natural. Además, acordaremos que las únicas constantes Prolog cuyo primer símbolo sea `x` será una variable del lenguaje objeto.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Directivas

:- op(600,xfx,-->).
:- dynamic delta/3.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% inserta_pos(L1,Pos,Term,L2)
% toma como entrada un literal L1, una lista de números Pos
% que representa la posición donde vamos a insertar el término
% Term y devuelve el literal resultante L2. Ejemplo:
%
%       ?- inserta_pos(-p(f(x1),a,x1),[1,1],x2,T).
%               T = -p(f(x2), a, x1) ;
%           No
%
%       ?- inserta_pos(x1,[],x2,T).
%               T = x2 ;
```

```
%           No
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

inserta_pos(_,[],Term,Term):-!.
inserta_pos(Lit_in,Pos,Term,Lit_out):-
     atomo(Lit_in,Atomo_in,Signo),
     inserta_pos_aux(Atomo_in,Pos,Term,Atomo_out),
     atomo(Lit_out,Atomo_out,Signo).

inserta_pos_aux(A1,[],_,A1).
inserta_pos_aux(L_in,[N],Term,L_out):-
     inserta_1(L_in,N,Term,L_out).
inserta_pos_aux(L_in,[N1,N2|Resto],Term,L_out):-
     L_in =.. [F|Args],
     N0 is N1 - 1,
     length(L,N0),
     append(L,[Arg_N1|Resto_args],Args),
     inserta_pos_aux(Arg_N1,[N2|Resto],Term,Arg_out),
     append(L,[Arg_out|Resto_args],Args_out),
     L_out =.. [F|Args_out].

% inserta_1(L_in,N,Term,L_out)
% Toma como entrada un término Term_in, un número N y un
% término Term_insertado y devuelve el átomo L_out resultante
% de sustituir el N-ésimo argumento de L_in por Term.

inserta_1(Term_in,N,Term_insertado,Term_out):-
     N1 is N-1,
     length(L1,N1),
     Term_in =.. [F|Args],
     append(L1,[_|L2],Args),
     append(L1,[Term_insertado|L2],Nuevo_args),
     Term_out =.. [F|Nuevo_args].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% inserta_en_posiciones(+L_in,+Conj,+Term,?L_out)
% Dado un literal L_in y un conjunto de posiciones
% independiente Conj formado por posiciones de L_in y un
% término Term, el programa devuelve el literal L_out, formado
% a partir de L_in insertando el término Term en cada una de
% las posiciones determinadas por el conjunto Conj
%
```

```
%
%  ?- inserta_en_posiciones(-p(f(x1,x2),x1,x1,x3),
%                           [[1,1],[3]],x2,T).
%          T = -p(f(x2, x2), x1, x2, x3) ;
%     No
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

inserta_en_posiciones(L_in,Conj,Term,L_out):-
     atomo(L_in,Atomo_in,Signo),
     inserta_en_posiciones_aux(Atomo_in,
                               Conj,Term,Atomo_out),
     atomo(L_out,Atomo_out,Signo).


inserta_en_posiciones_aux(Atomo,[],_,Atomo).
inserta_en_posiciones_aux(Atomo,[P|Resto],Term,Atomo_1):-
     inserta_pos(Atomo,P,Term,A_aux),
     inserta_en_posiciones_aux(A_aux,Resto,Term,Atomo_1).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% compatible_1(+Posiciones,+Literal,+Termino) tiene
% éxito si el conjunto de Posiciones es compatible con el par
% <Literal, Termino>. Ejemplos:
%
%   ?- compatible_1([[1]],p(f(x1),a,f(x1)),f(x1)).
%      Yes
%
%   ?- compatible_1([[1],[4,3]],p(f(x1),a,f(x1)),f(x1)).
%      No
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

compatible_1(Posiciones,Literal,Termino):-
     atomo(Literal,Atomo,_),
     compatible_1_aux(Posiciones,Atomo,Termino).

compatible_1_aux([],_,_).
compatible_1_aux([P1|Resto_posiciones],Atomo,Termino):-
     posicion(P1,Atomo,Termino),
     compatible_1_aux(Resto_posiciones,Atomo,Termino).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% compatible_2(+Conjunto,+Literal,+Termino) tiene
```

```
% éxito si el ''Conjunto'' de conjuntos de posiciones es
% compatible con el par <Literal, Termino> Por ejemplo,
% consideremos
%
%      Literal  = p(f(_X),a,f(_X))
%      Termino  = f(_X)
%      Conjunto = {P1,P2,P3}
%
% donde P1, P2 y P3 son los conjuntos de posiciones:
%
%     P1 = [[1],[3]]      (P1 = {<1>,<3>})
%     P2 = [[1]]          (P2 = {<1>})
%     P3 = []             (P3 = {})
%
%   ?- compatible_2([[[1],[3]],[[1]],[]],
%                    p(f(x1),a,f(x1)),f(x1)).
%      Yes
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

compatible_2([],_,_).
compatible_2([C1|Resto_conjuntos],Literal,Termino):-
     compatible_1(C1,Literal,Termino),
     compatible_2(Resto_conjuntos,Literal,Termino).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% comprueba_delta(Identificador,Termino)
% Comprueba si una aplicación Delta: Lit --> PPN^* es una
% asignación de posciones o no. En caso afirmativo, Termino
% es el termino que permite esta compatibilidad en caso de
% que sea único, si no es único entonces Termino = ***
%
%  ?- comprueba_delta(1,Termino).
%     Termino = a ;
%     No
%
%  ?- comprueba_delta(2,Termino).
%     Termino = *** ;
%     No
%
%  ?- comprueba_delta(3,Termino).
%     No
%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

comprueba_delta(Id,Termino):-
     recoge(Id,Lista_pares),
     comprueba_delta_aux(Lista_pares,L_terms),
     selecciona(L_terms,Termino).

comprueba_delta_aux([],[]).
comprueba_delta_aux([par(Lit,Conj)|Resto_objs],[T|Resto_t]):-
     comprueba_delta_literal(Conj,Lit,T),
     comprueba_delta_aux(Resto_objs,Resto_t).

comprueba_delta_literal(Conj,Lit,Termino):-
     comprueba_delta_literal_aux(Conj,Lit,Terms),
     selecciona(Terms,Termino).

selecciona(Terms,Termino):-
     list_to_set(Terms,Terms_0),
     elimina_*(Terms_0,Terms_1),
     depura_terms(Terms_1,Termino).

comprueba_delta_literal_aux([],_,[]).
comprueba_delta_literal_aux([P|Ps],Lit,[T|Ts]):-
     mismo_termino(P,Lit,T),
     comprueba_delta_literal_aux(Ps,Lit,Ts).

% Definimos a continuación recoge/2 y mismo_termino_3

% recoge(+Identificador, -Lista_de pares).
% Recibe como dato de entrada el identificador de la función
% de asignación y devuelve la lista de pares <L, Delta(L)>
% Ejemplo:
%
%   ?- recoge(1,L).
%      L = [par(p(f(a), b, a), [[], [[3]], [[1, 1]]]),
%           par(q(a),          [[]]),
%           par(_G393,         [])] ;
%      No
%

recoge(Id,L):-
     recoge_aux(Id,[],L1),
     reverse(L1,L),!.
```

```
recoge_aux(Id,Acum,L):-
     Obj =.. [delta,Id,Lit,Conj],
     retract((Obj:- !)),
     recoge_aux(Id,[par(Lit,Conj)|Acum],L),
     asserta((Obj:-!)).

recoge_aux(Id,Acum,[par(Lit,Conj)|Acum]):-
     Obj =.. [delta,Id,Lit,Conj],
     retract(Obj),
     asserta(Obj).

% mismo_termino(+Posiciones,+Atomo,?Term)
% Toma como entrada un conjunto de posiciones y un átomo y
% tiene éxito si todas las posiciones (distintas de la vacía)
% están ocupadas por el mismo término. En ese caso devuelte
% el término Term o *** en caso de que el término no esté
% determinado. Ejemplos:
%
%    ?- mismo_termino([[1,1],[],[3]],p(f(a),b,a),Term).
%           Term = a ;
%       No
%
%    ?- mismo_termino([[1,1],[],[3]],p(f(a),b,c),Term).
%       No
%
%    ?- mismo_termino([[]],p(f(a),b),Term).
%           Term = *** ;
%       No
%
%    ?- mismo_termino([],p(f(a),b),Term).
%           Term = *** ;
%       No
%

mismo_termino(Posiciones,Atomo,Termino):-
        setof_0(Pos,(member(Pos,Posiciones),\+ Pos=[]),Lista),
     busca_termino(Atomo,Lista,Termino).

busca_termino(_,[],***).
busca_termino(Atomo,[P1|Posiciones],Term):-
     compatible_1([P1|Posiciones],Atomo,Term).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% operador_literal(+Id,+Lit,+Term,?Clausula)
% Toma como dato de entrada el idendificador de una función
% de asignación (Id), un literal y un término y devuelve el
% conjunto de literales (Clausula) resultante de hacer las
% inserciones del término Term en las posiciones determinadas
% por la función de asignación. Ejemplos:
%
%
% Ejemplo 1:
%
%   ?- operador_literal(1,p(f(a),b,a),h(b),Clausula).
%
%  Clausula = [p(f(a),b,a), p(f(a),b,h(b)), p(f(h(b)),b,a)] ;
%    No
%
%   Este resultado se obtiene porque en nuestro ejemplo, la
%   imagen del literal
%                        L=p(f(a),b,a)
% por la aplicación delta_1 es el conjunto [[],[[3]],[[1,1]]] y
% * Si insertamos "h(b)" en "p(f(a),b,a)" en las posiciones
%   determinadas por "[]" obtenemos "p(f(a),b,a)" (el literal
%   no se modifica)
% * Si insertamos "h(b)" en "p(f(a),b,a)" en las posiciones
%   determinadas por "[[3]]" obtenemos "p(f(a),b,h(b))"
%   (modificamos la posición <3>)
% * Si insertamos "h(b)" en "p(f(a),b,a)" en las posiciones
%   determinadas por "[[1,1]]" obtenemos "p(f(h(b)),b,a)"
%   (modificamos la posición <1,1>)
%
%
% Ejemplo 2:
%
%     ?- operador_literal(1,q(a),h(b),Clausula).
%        Clausula = [q(a)] ;
%        No
%
%   En este caso la imagen de "q(a)" por la función delta_1
%   es "[[]]", esto es, la cláusula resultante tiene un único
%   literal, que es el propio "q(a)"
%
%
% Ejemplo 3:
```

```
%
%     ?- operador_literal(1,r(a,b),h(b),Clausula).
%        Clausula = [] ;
%        No
%
%   La imagen de "r(a,b)" por la aplicación delta_1 es "[]"
%   por lo cual la imagen obtenida es la cláusula vacía
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

operador_literal(Id,Lit,Term,Clausula):-
     Obj =.. [delta,Id,Lit,Pos],
     Obj,
     operador_literal_aux(Pos,Lit,Term,[],Claus),
     reverse(Claus,Clausula).

operador_literal_aux([],_,_,Acum,Acum).
operador_literal_aux([P|Resto],Lit,Term,Acum,Clausula):-
     inserta_en_posiciones(Lit,P,Term,L1),
     operador_literal_aux(Resto,Lit,Term,
                              [L1|Acum],Clausula).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% operador_clausal(op(Id,Term),Clausula_in,Clausula_out)
% Toma un operador determinado por el identificador Id de la
% asignación y el término Term que vamos a insertar y la
% cláusula de origen y devuelve el resultado de aplicar el
% operador a la cláusula. Ejemplo:
%
%  ?- operador_clausal(op(1,h(c)),
%                      [p(f(a),b,a),q(a),r(a,b)],Clausula).
%
%     Clausula = [p(f(a),b,a),
%                 p(f(a),b,h(c)),p(f(h(c)),b,a),q(a)] ;
%
%     No
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

operador_clausal(op(Id,Term),Clausula_in,Clausula_out):-
     operador_aux(Id,Term,Clausula_in,[],Clausula_out_1),
     list_to_set(Clausula_out_1,Clausula_out).
```

```
operador_aux(_,_,[],Acum,Acum).
operador_aux(Id,Term,[Lit|Resto],Acum,Sal):-
     operador_literal(Id,Lit,Term,Op_lit),
     append(Acum,Op_lit,Nuevo_acum),
     operador_aux(Id,Term,Resto,Nuevo_acum,Sal).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% operador_clausal_extendido(Par,op(Id,Term),Clausula_in,
% Clausula_out)
% Al igual que en la definición de operador clausal, el
% programa toma como dato de entrada la signación de
% posiciones (Delta) el término que vamos a insertar (Term) y
% la cláusula de entrada (Clausula_in). En este caso también
% toma como dato de entrada el elemento par, que puede ser el
% átomo @ o bien un par <Simbolo_de_predicado/Aridad, Signo>.
% Ejemplos:
%
% ?- operador_clausal_extendido(
%             @,                                % Marca
%             op(1,                             % Identificación
%                h(c)),                         % Término
%             [p(f(a),b,a),q(a),r(a,b)],        % C. de entrada
%             Clausula).                        % C. de salida
%
%    Clausula = [p(f(a), b, a), p(f(a), b, h(c)),
%                p(f(h(c)), b, a), q(a)] ;
%    No
%
%
% ?- operador_clausal_extendido(
%             [s/3,-],                          % <S/A,Signo>
%             op(1,                             % Asignación
%                h(c),                          % Término
%             [p(f(a),b,a),q(a),r(a,b)],        % C. de entrada
%             Clausula).                        % C. de salida
%
%     Clausula = [-s(h(c), h(c), h(c)),         % Literal nuevo
%                   p(f(a), b, a), p(f(a), b, h(c)),
%                   p(f(h(c)), b, a),
%                   q(a)] ;
%
% Se obtiene la misma cláusual que en el ejemplo anterior,
% aumentada con el literal "-s(h(c), h(c), h(c))".
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

operador_clausal_extendido(@,op(Id,Term),
                             Clausula_in,Clausula_out):-
     !,
     operador_clausal(op(Id,Term),Clausula_in,
                         Clausula_out).

operador_clausal_extendido([P/A,S],op(Id,Term),
                       Clausula_in,[Literal|Clausula_out]):-
     crea_literal(P/A,S,Term,Literal),
     operador_clausal(op(Id,Term),Clausula_in,
                         Clausula_out).

crea_literal(P/A,Signo,Term,Literal):-
     crea_argumentos(A,Term,Argumentos),
     Atomo =.. [P|Argumentos],
     atomo(Literal,Atomo,Signo).

crea_argumentos(0,_,[]).
crea_argumentos(N,T,[T|Resto]):-
     N > 0,
     N1 is N-1,
     crea_argumentos(N1,T,Resto).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejemplo 3.23:
% Consideremos las cláusulas
%
%       C = [q(x1,x2), -r(a),p(x1,b)]
%       D = [s(a,f(b)), r(x3)]
%
% Veamos que podemos pasar de C a D mediante operadores
% clausales extendidos. Seguimos los pasos del teorema
%
% Paso 1: Consideremos la asignación siguiente vacía
% (definida en asignaciones.pl con índice 0) y veamos
% que el operador asociado a esa asignación y una variable
% cualquiera, por ejemplo x1, nos lleva C en la cláusula
% vacía
%
%  ?- operador_clausal(op(0,x1),[q(x1,x2), -r(a),p(x1,b)],C1).
```

UNIVERSIDAD DE SEVILLA
FACULTAD DE MATEMÁTICAS


```
%           C1 = [] ;
%     No
%
%
% Paso 2a: Puesto que la cláusula D tiene dos literales,
% tomamos dos variables que no ocurran en D, por ejemplo x1 y
% x2 y consideramos dos asignaciones de variables a partir de
% las cuales vamos a generar nuestros operadores clausales
% extendidos. En este caso corresponden con los índices 7 y 8
% en el fichero asignaciones. (La asignación 7 corresponde,
% según el teorema, a la variable x1 y al literal s(a,f(b))
% y la asignación 8 a la variable x2 y al literal r(x3).
% Si aplicamos a C1 = [] el operador extendido asociado al
% par correspondiente al literal s(a,f(b)), esto es al par
% [s/2,+], a la asignación de índice 7 y a la variable x1
% tendríamos una nueva cláusula C2
%
%    ?- operador_clausal_extendido([s/2,+],op(7,x1),[],C2).
%           C2 = [s(x1, x1)] ;
%       No
%
% Y si ahora a C2 le aplicamos el operador extendido
% asociado al par correspondiente al literal r(x3), esto es
% al par [r/1,+], a la asignación de índice 8 y a la variable
% x2 tendríamos una nueva cláusula C3
%
%   ?- operador_clausal_extendido(
%                 [r/1,+],op(8,x2),[s(x1,x1)],C3).
%           C3 = [r(x2), s(x1, x1)] ;
%      No
%
% Paso 2b: En este paso tenemos que encontrar operadores que
% nos lleven la cláusula C3 = [r(x2), s(x1, x1)] en la
% cláusula D = [s(a,f(b)), r(x3)] Siguiendo el teorema,
% asociado a s(x1,x1) hay dos operadores clausales, uno
% insertará "a" en la posición 1 y el segundo insertará f(b)
% en la posición 2. corresponden con las asignaciones 9 y 10
% del fichero de asignaciones
%
%   ?- operador_clausal(op(9,a),[r(x2), s(x1, x1)],C4).
%         C4 = [r(x2), s(a, x1)] ;
%      No
%
```

```
%   ?- operador_clausal(op(10,f(b)),[r(x2), s(a, x1)],C5).
%         C5 = [r(x2), s(a, f(b))] ;
%
% De manera análoga, asociado a r(x2) tomamos la
% asignación 11
%
%   ?- operador_clausal(op(11,x3),[r(x2), s(a, f(b))],C6).
%         C6 = [r(x3), s(a, f(b))] ;
%
% Con lo que obtenemos C6, que es la cláusula D a la que
% queríamos llegar.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% operador_asociado_sustitucion(C1,[X --> T],C2,op(Id,Term))
% Dada la cláusula C1 y la sustitución elemental [X -- >T], el
% predicado devuelve la cláusula C2 resultante de aplicar la
% sustitución a C1, inserta en la base de conocimiento la
% asignación correspondiente y devuelve un operador clausal,
% caracterizado por el identificador de la asignación y el
% término que debe ser insertado. Ejemplo:
%
%    ?- operador_asociado_sustitucion(
%             [p(x1,a),p(a,x1),q(x1)],      % C1
%             [x1 --> a],                   % Sust
%             C2,
%             op(Id, Term)).
%
%                    C2 = [p(a, a), q(a)]
%                    Id = 5
%                    Term = x1
%        Yes
%
%  Podemos ver ahora que al aplicarle el operador a C2
%  recuperamos C1
%
%          ?- operador_clausal(5,x1,[p(a,a), q(a)],C1).
%
%                C1 = [p(x1, a), p(a, x1), q(x1)] ;
%
%             No
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
operador_asociado_sustitucion(C1,[X --> T],C2,op(Id,X)):-
     crea_delta_sustitucion(C1,[X --> T],C2,Pares),
     inserta_delta(Pares,Id).

% Definimos a continuación crea_delta_sustitucion/4 e
% inserta_delta/2

% crea_delta_sustitucion(C1,[X --> T],C2,Pares)
% Siguiendo el teorema, a partir de C1 y la sustitución
% elemental [X --> T] devolvemos C2, el conjunto de pares de
% la función de asignación correspondiente. Ejemplo:
%
%  ?- crea_delta_sustitucion(
%           [p(x1,a),p(a,x1),q(x1)],  % C1
%           [x1 --> a],               % Sust. elemental
%           C2,
%           Pares).
%
%               C2 = [p(a, a), q(a)]
%               Pares = [par(p(a, a), [[[1]], [[2]]]),
%                        par(q(a), [[[1]]]),
%                        par(_G1196, [[]])] ;
%
%       No
%

crea_delta_sustitucion(C1,[X --> T],C2,Pares):-
     aplica_a_clausula(C1,[X --> T],C2),
     !,
     aux_crea_delta_sustitucion(C1,[X --> T],C2,Pares).

aux_crea_delta_sustitucion(_,_,[],[par(_,[[]])]).
aux_crea_delta_sustitucion(C1,
                           [X --> T],
                           [L|Resto],
                           [par(L,Lista_pos)|Pares]):-
     literales_inversa(L,C1,[X --> T],Literales),
     lista_de_posiciones(Literales,X,Lista_pos),
     aux_crea_delta_sustitucion(C1,[X --> T],Resto,Pares).

% inserta_delta(+Pares,-Id)
% Recibe una asignación de posiciones en forma de lista de
% pares y las inserta en la base de conocimiento con el
```

```
% formato adecuado. Además devuelve el identificador de esta
% nueva asignación. Ejemplo:
%
%  ?- inserta_delta([par(p(a, a), [[[1]], [[2]]]),
%                    par(q(a), [[[1]]]),
%                    par(_, [[]])],
%                    Id).
%          Id = 4;
%          No
%
%  ?- listing(delta).
%            ...
%        delta(4, p(a, a), [[[1]], [[2]]]) :- !.
%        delta(4, q(a), [[[1]]]) :- !.
%        delta(4, A, [[]]).
%
%         Yes.
%

inserta_delta(Pares,N):-
     nuevo_identificador(N),
     !,
     aux_inserta_delta(N,Pares).

aux_inserta_delta(N,[par(L,S)]):-
     Obj =.. [delta,N,L,S],
     assertz(Obj).
aux_inserta_delta(N,[par(L,S),par(L1,S1)|Resto]):-
     Obj =.. [delta,N,L,S],
     assertz((Obj :- !)),
     aux_inserta_delta(N,[par(L1,S1)|Resto]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% operador_asociado_subconjunto(+C2,?C1,op(?Id,?Term))
% Recibe como entrada una cláusula C2 y opcionalmente una
% cláusula C1 tal que C1 esté contenida en C2 (si no se
% proporciona genera las posibles) y devuelve un operador
% clausal que aplicado a C2 nos da C1, caracterizado por el
% identificador de su asignación Id y el término insertado Term
% (Nota: En este caso el término insertado no juega ningún
% papel y lo fijamos a x1)
%
%    ?- operador_asociado_subconjunto(
```

```
%                          [p(a,a), q(b), -r(a,x1)],        % C2
%                          [q(b)],                          % C1
%                          op(Id,
%                             Term)).
%
%           Id = 16
%           Term = x1 ;
%
%            No
%
%    ?- listing(delta).
%              ...
%
%         delta(16, p(a, a), []) :- !.
%         delta(16, -r(a, x1), []) :- !.
%         delta(16, A, [[]]).
%
%  Si ahora aplicamos el operador a C2 obtenemos C1
%
%    ?- operador_clausal(16,x1,[p(a,a), q(b), -r(a,x1)],C1).
%                  C1 = [q(b)] ;
%             No
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

operador_asociado_subconjunto(C2,C1,op(Id,x1)):-
     subconjunto(C2,C1),
     diferencia(C2,C1,Diferencia),
     crea_delta_subconjunto(Diferencia,Pares),
     inserta_delta(Pares,Id).

% Definimos crea_delta_subconjunto/2

% crea_delta_subconjunto(+C,?Pares)
% Toma como dato de entrada un conjunto de literales y
% devuelve la asignación que asocia a cada literal del
% conjunto el vacío, y al resto de los literales el
% conjunto [[]],
%
%     ?- crea_delta_subconjunto([p(a,x1), -q(b)],Pares).
%
%        Pares = [par(p(a, x1), []),
%                 par(-q(b), []),
```

```
%                 par(_G517, [[]])] ;
%       No
%

crea_delta_subconjunto([],[par(_,[[]])]).
crea_delta_subconjunto([L|Resto_1],[par(L,[])|Resto_2]):-
     crea_delta_subconjunto(Resto_1,Resto_2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% descompone_sustitucion(+Sust,-Lista_elementales)
% Toma como entrada una sustitución y devuelve una lista de
% sustituciones elementales
%
%  ?- descompone_sustitucion(
%      [x1 --> f(x1,x3), x2 --> x4, x6 --> a],   % Sustitución
%                Lista).
%
%                Lista = [[x1-->f(x7, x9)],        % Sigma 1
%                         [x2-->x4],               % Sigma 2
%                         [x6-->a],                % Sigma 3
%                         [x7-->x1],               % Sigma 4
%                         [x8-->x2],               % Sigma 5
%                         [x9-->x3],               % Sigma 6
%                         [x10-->x4],              % Sigma 7
%                         [x11-->x6]] ;            % Sigma 8
%
%          No
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

descompone_sustitucion(Sust,Lista_elementales):-
     variables_nuevas(Sust,[],Var,Var_nuevas),
      enlaza_variables(Var,Var_nuevas,Delta,Parte_2),
     descompone_sustitucion_aux(Sust,Delta,Parte_1),
     append(Parte_1,Parte_2,Lista_elementales).

% Crea el renombramiento de la proposición y las
% sustituciones elementales que lo invierten

enlaza_variables([],[],[],[]).
enlaza_variables([Y|Ys],[Z|Zs],
                 [Y --> Z |Resto_delta],
                 [[Z --> Y]|Resto_sust]):-
```

```
     enlaza_variables(Ys,Zs,Resto_delta,Resto_sust).

% Crea las primeras sustituciones elementales según la
% proposición

descompone_sustitucion_aux([],_,[]).
descompone_sustitucion_aux([X --> Y |Ss],Delta,[[X --> T]|Pp]):-
     aplica_a_literal(Y,Delta,T),
     descompone_sustitucion_aux(Ss,Delta,Pp).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% genera_operadores_clausales(C1,C2,Lista_operadores)
% Toma como dato de entrada dos cláusulas C1 y C2. Si C1
% subsume a C2, devuelve una lista de operadores
% [op(Id_1,T_1),...,op(Id_k,T_k)] determinados por el
% identificador de la asignación de literales y el
% término insertado tales que aplicados a C2 obtenemos C1.
% Por ejemplo:
%
%  ?- genera_operadores_clausales(
%              [p(x1,x3),q(x3)],        % C1
%              [p(x2,x3),q(x3),r(x1)],  % C2
%              Lista).
%
%   Lista = [op(4, x1), op(7, x5), op(6, x4), op(5, x1)] ;
%
%     No
%
% Comprobamos que por aplicación de estos operadores
% obtenemos C2 a partir de C1
%
%  ?- operador_clausal(op(4,x1),
%                      [p(x2,x3),q(x3),r(x1)],Caux_1).
%
%          Caux_1 = [p(x2, x3), q(x3)]
%
%          con        delta(4, r(x1), []) :- !.
%                     delta(4, A, [[]]).
%
%
%  ?- operador_clausal(op(7,x5),[p(x2,x3),q(x3)],Caux_2).
%
%         Caux_2 = [p(x5, x3), q(x3)]
```

```
%
%         con          delta(7, p(x2, x3), [[[1]]]) :- !.
%                      delta(7, q(x3), [[]]) :- !.
%                      delta(7, A, [[]]).
%
%  ?- operador_clausal(op(6,x4),[p(x5,x3),q(x3)],Caux_3).
%
%         Caux_3 = [p(x5, x3), q(x3)]
%
%         con          delta(6, p(x5, x3), [[]]) :- !.
%                      delta(6, q(x3), [[]]) :- !.
%                      delta(6, A, [[]]).
%
%  ?- operador_clausal(op(5,x1),[p(x5,x3),q(x3)],C1).
%
%         C1 = [p(x1, x3), q(x3)]
%
%         con          delta(5, p(x5, x3), [[[1]]]) :- !.
%                      delta(5, q(x3), [[]]) :- !.
%                      delta(5, A, [[]]).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

genera_operadores_clausales(C1,C2,[Op_sub|Lista_op_sust]):-
     genera_operadores_clausales_aux(
             C1,C2,_,C1_sust,Lista_elementales),
     operador_asociado_subconjunto(C2,C1_sust,Op_sub),
     crea_lista_op_sust(
             C1,Lista_elementales,[],Lista_op_sust).


% Definimos crea_lista_op_sust/4 y
% genera_operadores_clausales_aux/5

crea_lista_op_sust(_,[],Acum,Acum).
crea_lista_op_sust(C1,[Sust|Resto_sust],Acum,Lista_op_sust):-
     operador_asociado_sustitucion(C1,Sust,C2,OP),
     crea_lista_op_sust(
             C2,Resto_sust,[OP|Acum],Lista_op_sust).

% genera_operadores_clausales_aux(C1,C2,Sust,
%                                 C1_sust,Lista_elementales)
%
```

```
% Toma como dato de entrada cos cláusulas. Si C1 subsume a
% C2 devuelve una sustitución Sust que permite la relación de
% subsuncion y la cláusula C1_sust resultante de aplicar Sust
% a C1. Además devuelve la descomposición de la sustitución
% en sustituciones elementales en las que las variables
% nuevas que aparecen no ocurren en la sustitución ni en C1
% Ejemplo:
%
%      ?- genera_operadores_clausales_aux(
%                      [p(x1,x2),q(x2)],          % C1
%                      [p(x2,x1),q(x1),r(x1)],    % C2
%                       Sust,
%                       C1_sust,
%                       Lista_elementales).
%
%                  Sust              = [x1-->x2, x2-->x1]
%                  C1_sust           = [p(x2, x1), q(x1)]
%                  Lista_elementales = [[x1-->x4],
%                                       [x2-->x3],
%                                       [x3-->x1],
%                                       [x4-->x2]] ;
%
%       No
%

genera_operadores_clausales_aux(
                    C1,C2,Sust,C1_sust,Lista_elementales):-
     subsuncion_clausulas(C1,C2,Sust),
     aplica_a_clausula(C1,Sust,C1_sust),
     variables_en_termino(C1,Vars_1),
     variables_nuevas(Sust,Vars_1,Var,Var_nuevas),
     enlaza_variables(Var,Var_nuevas,Delta,Parte_2),
     descompone_sustitucion_aux(Sust,Delta,Parte_1),
     append(Parte_1,Parte_2,Lista_elementales).

% Hemos usado los auxiliares de descompone

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejemplos 4.12
%
% Ejemplo 1: Consideremos la asignación 4 del fichero
% asignaciones.pl (Al final del apéndice).
%
```

```
% (1) Es asignación de posiciones
%
%      ?- comprueba_delta(4,K).
%              K = f(x1,x3) ;
%         No
%
% (2) Consideremos el operador "op(4,x1)". Si insertamos una
%     constante "a" en los literales tales que su imagen por
%     la asignación no es vacía, vemos que x1 no ocurre en
%     ningún literal, esto es, op(4,x1) verifica la tesis
%     del lema.
%
%     ?- operador_clausal(op(4,a),
%            [p(f(x1,x3),x2),q(h(f(x1,x3))),-r(x2)],C).
%                   C = [p(a, x2), q(h(a)), -r(x2)] ;
%        No
%
%
% (3) Siguiendo el lema, la imagen de cualquier cláusula C
%     por el operador es más general que C. Por ejemplo
%
%         ?- _C1 = [p(f(x1,x3),x2),
%                   q(h(f(x1,x3))),-r(x2),p(x1,x2),q(x3)],
%            operador_clausal(op(4,x1),_C1,C2),
%            subsuncion_clausulas(C2,_C1,Sust).
%
%            C2 = [p(x1, x2), q(h(x1)), -r(x2)]
%            Sust = [x1-->f(x1, x3)]
%
%
%  Ejemplo 2: Consideremos ahora la asignación 5 del
%             fichero asignaciones.pl
%
% (1) Es asignación de posiciones
%
%       ?- comprueba_delta(5,K).
%               K = c ;
%          No
%
% (2) Consideremos el operador "op(5,x1)". En este caso no se
%     verifica la hipótesis del lema. Por ejemplo, si tomamos
%     el literal "p(f(c),x1,c)" y el conjunto de posiciones
%     [[1,1]] que pertenece a la imagen del literal por la
```

```
%     asignación 5, es trivial comprobar que x1 ocurre en el
%     literal resultante de insertar la constante a en la
%     posicion [1,1]
%
%        ?- inserta_en_posiciones(p(f(c),x1,c),[[1,1]],a,L).
%                 L = p(f(a), x1, c) ;
%           No
%
% (3) Ilustramos con un ejemplo que en este caso podemos
%     encontrar una cláusula C tal que la imagen de C por el
%     operador op(5,x1) no es más general bajo subsunción que C.
%     Para construir tal cláusula usamos el literal
%     "p(f(c),x1,c)".
%
%      ?- operador_clausal(
%           op(5,x1),
%           [p(f(c),x1,c),q(h(c,x2)),-r(x2,x1)],       % C1
%           C2).
%
%                  C2 = [p(f(c), x1, c),
%                        p(f(c), x1, x1),
%                        p(f(x1), x1, c),
%                        q(h(x1, x2))] ;
%
%       Y la imagen obtenida por el operador no subsume la
%       cláusula original
%
%     ?- subsuncion_clausulas(
%         [p(f(c), x1, c), p(f(c), x1, x1),
%                         p(f(x1), x1, c),q(h(x1,x2))],
%         [p(f(c),x1,c),q(h(c,x2)),-r(x2,x1)],
%         Sust).
%
%        No
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% operador_incidente(+Op)
% Se verifica si OP es un operador incidente
% Por ejemplo:
%
%        ?- operador_incidente(op(1,x2)).
```

```
%            No
%
%   op(1,x2) no es incidente porque el único término que
%   permite la compatibilidad es "a" y x2 no ocurre en "a".
%
%
%        ?- operador_incidente(op(2,x2)).
%            No
%  op(2,x2) no es incidente porque la compatibilidad no
%  viene determinada por un único término
%
%       ?- operador_incidente(op(6,x2)).
%           Yes
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


operador_incidente(op(Id,Var)):-
     comprueba_delta(Id,Term),
     \+ Term = ***,
     ocurre(Var,Term).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% no_incidente(op(+Id,-Soporte))
% Lo definimos siguiendo la equivalencia del lemas 4.14,
% apdo 3. Si la imagen de todo literal es [] o [[]] entonces
% Term es *** y ocurre(V,***) falla. Si no es *** entonces
% Termino es el único término que permite la compatibilidad,
% en cuyo caso debe verificarse que V no ocurra en Term
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

no_incidente(op(Id,V)):-
     comprueba_delta(Id,Term),
     \+ ocurre(V,Term).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% oas_soporte_finito(+op(Id,Term))
% Se verifica si el operador es un oas con soporte finito
% Nótese que los operadores clausales generados según los
% lemas 4.18 y 4.19 son de soporte finito
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oas_soporte_finito(op(Id,Var)):-
```

```
     no_incidente(op(Id,Var)),
     soporte(Id,Soporte),
     operador_clausal(op(Id,a),Soporte,Soporte_insertado),
     \+ ocurre(Var,Soporte_insertado).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% oas_asociado_subconjunto(+C,,op(?Id,?Term))
% Recibe como entrada una cláusula C devuelve un oas de
% soporte finito según el lema 4.18. Si tenemos una cláusula
% D tal que C esté contenido en D, al aplicar el operador a
% D nos devuelve C.
%
%    ?- oas_asociado_subconjunto([-q(a),p(x1,a),r(b,x2)],OAS).
%            OAS = op(7, x3) ;
%       No
%
%    ?- listing(delta).
%            ...
%          delta(7, -q(a), [[]]) :- !.
%          delta(7, p(x1, a), [[]]) :- !.
%          delta(7, r(b, x2), [[]]) :- !.
%          delta(7, A, []).
%
%    ?- oas_soporte_finito(op(7,x3)).
%         Yes
%
%    ?- operador_clausal(
%                  op(7,x3),
%                  [-q(a),p(x1,a),r(b,x2),p(a,b),r(x3,x2)],
%                   C).
%
%           C = [-q(a), p(x1, a), r(b, x2)] ;
%
%       No
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oas_asociado_subconjunto(C,op(Id,Var_nueva)):-
     variables_en_termino(C,Lista_variables),
     nuevo_indice(Lista_variables,Indice),
     crea_variable(Indice,Var_nueva),
     crea_oas_subconjunto(C,Pares),
     inserta_delta(Pares,Id).
```

```
% Definimos al auxiliar crea_oas_subconjunto/2

% crea_oas_subconjunto(+C,?Pares)
% Toma como dato de entrada un conjunto de literales y
% devuelve la asignación que asocia a cada literal el
% conjunto [[]], y al resto de los literales el conjunto
% vacío []
%
%   ?- crea_oas_subconjunto([-q(a),p(x1,a),r(b,x2)],Pares).
%               Pares = [par(-q(a), [[]]),
%                        par(p(x1, a), [[]]),
%                        par(r(b, x2), [[]]),
%                        par(_G520, [])] ;
%         No
%
%

crea_oas_subconjunto([],[par(_,[])]).
crea_oas_subconjunto([L|Resto_1],[par(L,[[]])|Resto_2]):-
     crea_oas_subconjunto(Resto_1,Resto_2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% oas_asociado_sustitucion(+C,+[X --> T],?D,?op(Id,Term))
% Dada la cláusula C y la sustitución elemental [X -- >T],
% el predicado devuelve la cláusula D resultante de aplicar
% la sustitución a C, inserta en la base de conocimiento la
% asignación correspondiente y devuelve el oas asociado,
% caracterizado por el identificador de la
% asignación y el término que debe ser insertado.
% Ejemplo:
%
%        ?- oas_asociado_sustitucion(
%                   [p(x1,x2,f(x1)),-q(x1)],          % C
%                   [x1 --> f(x2)],                   % Sust
%                   D,
%                   OP).
%
%
%            D = [p(f(x2), x2, f(f(x2))), -q(f(x2))]
%            OP = op(9, x1) ;
%          No
%
```

```
%       ?- listing(delta)
%            ...
%       delta(9, p(f(x2), x2, f(f(x2))), [[[1], [3, 1]]]) :- !.
%       delta(9, -q(f(x2)), [[[1]]]) :- !.
%       delta(9, A, [[]]).
%
%
%       ?- oas_soporte_finito(op(9,x1)).
%          Yes
%
%       ?- operador_clausal(
%           op(9,x1),                                  % Operador
%           [p(f(x2), x2, f(f(x2))), -q(f(x2))],       % D
%           C)
%
%               C = [p(x1, x2, f(x1)), -q(x1)] ;
%
%           No
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oas_asociado_sustitucion(C,[X --> T],D,op(Id,X)):-
     crea_delta_sustitucion(C,[X --> T],D,Pares),
     inserta_delta(Pares,Id).

% Definimos el auxiliar crea_oas_sustitucion/4

% crea_oas_sustitucion(+C,+[X --> T],?D,?Pares)
% Siguiendo el lema, a partir de C y la sustitución elemental
% [X --> T] devolvemos D, el conjunto de pares de la función
% de asignación correspondiente. Ejemplo:
%
%        ?- crea_oas_sustitucion(
%                  [p(x1,x2,f(x1)),-q(x1)],          % C
%                  [x1 --> f(x2)],                   % Sust
%                  D,
%                  Pares).
%
%     D = [p(f(x2), x2, f(f(x2))), -q(f(x2))]
%     Pares = [par(p(f(x2), x2, f(f(x2))), [[[1], [3, 1]]]),
%              par(-q(f(x2)), [[[1]]]),
%              par(_G1150, [])] ;
%
```

```
%          No
%

crea_oas_sustitucion(C,[X --> T],D,Pares):-
     aplica_a_clausula(C,[X --> T],D),
     !,
     aux_crea_oas_sustitucion(C,[X --> T],D,Pares).

aux_crea_oas_sustitucion(_,_,[],[par(_,[])]).
aux_crea_oas_sustitucion(
        C,[X --> T],[L|Resto],[par(L,Lista_pos)|Pares]):-
     literales_inversa(L,C,[X --> T],Literales),
     lista_de_posiciones(Literales,X,Lista_pos),
     aux_crea_oas_sustitucion(C,[X --> T],Resto,Pares).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% genera_oas(C1,C2,Lista_operadores)
% Toma como dato de entrada dos cláusulas C1 y C2. Si C1
% subsume a C2, devuelve una lista de oas
% [op(Id_1,T_1),...,op(Id_k,T_k)] determinados por el
% identificador de la asignación de literales y el término
% insertado tales que aplicados a C2 obtenemos C1.
% Por ejemplo:
%
%  ?- genera_oas([p(x1,f(x2)),-q(x2)],                  % C
%                [p(x2,f(h(x1))),-q(h(x1)),r(x1,x2)],   % D
%                Lista_op).
%
%                   Lista_op = [op(10, x3),
%                               op(14, x4),
%                               op(13, x3),
%                               op(12, x2),
%                               op(11, x1)] ;
%
%               No
%
%          ?- oas_soporte_finito(op(10,x3)).
%             Yes
%
%          ?- oas_soporte_finito(op(14,x4)).
%             Yes
%
%          ?- oas_soporte_finito(op(13,x3)).
```

```
%              Yes
%
%           ?- oas_soporte_finito(op(12,x2)).
%              Yes
%
%           ?- oas_soporte_finito(op(11,x1)).
%              Yes
%
%
%          ?- aplica_lista_operadores(
%              [p(x2,f(h(x1))),-q(h(x1)),r(x1,x2)],      % D
%              [op(10, x3),
%               op(14, x4),
%               op(13, x3),                        % Operadores
%               op(12, x2),
%               op(11, x1)],
%               C).
%
%              C = [p(x1, f(x2)), -q(x2)]
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

genera_oas(C1,C2,[Op_sub|Lista_op_sust]):-
     aux_genera_operadores_clausales(
            C1,C2,_,C1_sust,Lista_elementales),
     oas_asociado_subconjunto(C1_sust,Op_sub),
     crea_lista_oas_sust(
            C1,Lista_elementales,[],Lista_op_sust).

crea_lista_oas_sust(_,[],Acum,Acum).
crea_lista_oas_sust(C1,[Sust|Resto_sust],Acum,Lista_op_sust):-
     oas_asociado_sustitucion(C1,Sust,C2,OP),
     crea_lista_oas_sust(
           C2,Resto_sust,[OP|Acum],Lista_op_sust).

% Definimos a continuación varios auxiliares relacionados
% con la quasi-distancia de subsunción

% desc(+Sustitución,+Variables,-Long,-Desc)
% Se verifica si Long es la longitud de la Sustitución y Desc
% es una descomposición de las sustituciones elementales en el
% que las variables nuevas, (si las hubiera), no ocurren en la
```

```
% lista Variables
%
%  ?- setof(_L-_Desc,
%           desc([x1 --> f(x1,h(x2)),
%                 x2 --> h(x3), x3 --> h(h(f(x1,x3)))],
%                [x1,x2,x3,x4],
%                _L,
%                _Desc),
%           [Peso-Minimal|_L]),
%           length([Peso-Minimal|_L],N).
%
%       Peso = 5
%       Minimal = [[x1-->x5], [x3-->x6], [x6-->h(h(f(x1, x3)))],
%                  [x2-->h(x3)], [x5-->f(x1, h(x2))]]
%       N = 88
%
%       Yes

desc(Sust,Variables,Long,Desc):-
     crea_ternas(Sust,Ternas),
     dominio_y_ran(Ternas,Dom,Ran),
     append(Dom,Ran,Dom_y_Ran),
     append(Dom_y_Ran,Variables,Var_aux),
     list_to_set(Var_aux,Var),
     !,
     asserta((prohibidas(Ternas,V):-!,V=Var)),
     desc_aux_1(Ternas,Var,0,Long,[],Desc).


desc_aux_1([],_,Long,Long,Desc,Desc).

desc_aux_1(Ternas,Var,Acum_long,Long,A2,Desc):-
     dominio_y_ran(Ternas,_,Ran),
     select(terna(X,T,_),Ternas,Resto_ternas),
     \+ member(X,Ran),
     nueva_sustitucion_1(Resto_ternas,T,X,Nuevas_ternas),
     Nuevo_acum_long is Acum_long + 1,
     desc_aux_1(Nuevas_ternas,Var,
               Nuevo_acum_long,Long,[[X --> T]|A2],Desc).

desc_aux_1(Ternas,Var,Acum_long,Long,A2,Desc):-
     dominio_y_ran(Ternas,Dom,Ran),
     subset(Dom,Ran),
```

```
    lista_de_subterminos(Ternas,Subterminos),
    member(Sub_T,Subterminos),
      \+ member(Sub_T,Var),
    encuentra_variable_nueva(Sub_T,Var,Nueva),
      nueva_sustitucion_1(Ternas,Sub_T,Nueva,Nuevas_ternas),
    \+ Ternas = Nuevas_ternas,
      Nuevo_acum_long is Acum_long + 1,
      desc_aux_1(Nuevas_ternas,[Nueva|Var],
         Nuevo_acum_long,Long,[[Nueva --> Sub_T]|A2],Desc).

% peso(+Sustitución,+Variables,-Peso)
% Calcula el peso de una variable y las variables nuevas para
% la descomposición las toma, si las necesita,fuera de Variables.
% Además devuelve una descomposición mínimal.
%
%    ?- peso([x1 --> f(g(x2)), x2 --> g(x3), x3 --> f(g(x2),x1)],
%            [x1,x2,x3],
%            Peso,
%            Desc).
%
%      Peso = 4
%      Desc = [[x1-->f(x4)], [x3-->f(x4, x1)],
%              [x2-->g(x3)], [x4-->g(x2)]]
%

peso(Sustitucion,Variables,Peso,Desc):-
   setof(L-D,desc(Sustitucion,Variables,L,D),[Peso-Desc|_]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% d_subsuncion(C,D,Distancia)
% Dadas las cláusulas C y D, devuelve su quasi-distancia de
% subsunción
%
%   ?- d_subsuncion([p(x1,x2),-q(f(h(a,x2)))],
%                   [-q(f(h(a,h(a,x2)))),
%                     p(f(f(x1)),h(a,x2)),p(b,h(a,x2))],
%                     Distancia).
%
%        Distancia = 3
%           (con Desc =  [[x2-->x3],
%                         [x3-->h(a, x2)], [x1-->b]])
%
%     ?- d_subsuncion([p(x1,x2),-q(f(h(a,x2)))],
```

```
%                    [-q(f(h(a,h(a,x2)))),p(f(f(x1)),
%                     h(a,x2)),p(b,h(a,b))],
%                Distancia).
%
%       Distancia = 4
%          (con Desc  = [[x1-->x4], [x4-->f(f(x1))],
%                        [x2-->x3], [x3-->h(a, x2)]])
%
%
%    ?- d_subsuncion([p(x1,x2),-q(f(h(a,x2)))],
%                    [r(a,f(x1)),p(x1,x2),-q(f(h(a,x2)))],
%                     Distancia).
%
%       Distancia = 1
%         (Es subconjunto poero no igual)
%
%
%    ?- d_subsuncion([p(x1,x2),-q(f(h(a,x2)))],
%                    [-q(f(h(a,x2))),p(x1,x2)],Distancia).
%
%       Distancia = 0
%         (Igualdad entre cláusulas)
%
%    ?- d_subsuncion([p(x1,a),-q(f(h(a,x2)))],
%                    [-q(f(h(a,x2))),p(x1,x2),-r(h(a,x2))],
%                    Distancia).
%
%       Distancia = infinito ;
%         (No se verifica la subsunción).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

d_subsuncion(C,D,Distancia):-
     subset(C,D),
     !,
     (subset(D,C) -> Distancia = 0 ; Distancia = 1).
d_subsuncion(C,D,Distancia):-
     setof(Peso,peso_candidato(C,D,Peso),[Menor|_]),
     !,
     Distancia=Menor.
d_subsuncion(_,_,infinito).

peso_candidato(C,D,Peso):-
```

```
    subsuncion_clausulas(C,D,Sust),
    variables_en_clausula(C,Var_C),
    peso(Sust,Var_C,Peso,_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% pq_dist(P1,P2,Pqd)
% Se verifica si la pseudo-quasi-distancia de P1 a P2 es PQD
% Ejemplo de uso:
%
%  prueba_pq(C0,C1,C2,C3,D_01,D_02,D_03,D_10,D_20,D_30,
%                 Dist_12,Dist_21):-
%         C0  = [p(x10), -q(x10,x11), -r(x11)],
%         C1 = [p(c), - q(c,x2), -r(x2)],
%         C2 = [p(x1), - q(x1,x2), -r(x2), -q(x3,x2)],
%         C3 = [p(x1), - q(x1,x2), -r(x2), -s(x3,x2)],
%        P1 = [C0],
%        P2 = [C1,C2,C3],
%        d_subsuncion(C0,C1,D_01),
%        d_subsuncion(C0,C2,D_02),
%        d_subsuncion(C0,C3,D_03),
%        d_subsuncion(C1,C0,D_10),
%        d_subsuncion(C2,C0,D_20),
%        d_subsuncion(C3,C0,D_30),
%         pq_dist(P1,P2,Dist_12),
%         pq_dist(P2,P1,Dist_21).
%
%
%    ?- prueba_pq(C0,C1,C2,C3,D_01,D_02,D_03,D_10,D_20,D_30,
%                   Dist_12,Dist_21).
%
%          C0 = [p(x10), -q(x10, x11), -r(x11)]
%          C1 = [p(c), -q(c, x2), -r(x2)]
%          C2 = [p(x1), -q(x1, x2), -r(x2), -q(x3, x2)]
%          C3 = [p(x1), -q(x1, x2), -r(x2), -s(x3, x2)]
%          D_01 = 2
%          D_02 = 2
%          D_03 = 2
%          D_10 = infinito
%          D_20 = 3
%          D_30 = infinito
%          Dist_12 = 2
%          Dist_21 = 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
pq_dist(P1,P2,Pqd):-
     setof(Min,es_minimo(P1,P2,Min),L_min_D),
     max_list_inf(L_min_D,Pqd).

es_minimo(P1,P2,Min):-
     member(D,P2),
     calcula_minimo(P1,D,Min).

calcula_minimo(P1,D,Min):-
   lista_de_distancias(P1,D,Lista_qd),
     min_list_inf(Lista_qd,Min).

lista_de_distancias([],_,[]).
lista_de_distancias([C|R1],D,[Dist_CD|R2]):-
        d_subsuncion(C,D,Dist_CD),
     lista_de_distancias(R1,D,R2).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicados auxiliares:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

atomo(-A,A,-):-!.
atomo(A,A,+).

setof_0(X,G,L):-setof(X,G,L),!.
setof_0(_,_,[]).

posicion([],Atomo,Term):-
        Atomo=Term.
posicion([N],Atomo,Term):-
     n_arg(N,Atomo,Term).
posicion([M1,M2|Pos],Atomo,Term):-
     n_arg(M1,Atomo,Sub),
     posicion([M2|Pos],Sub,Term).

elimina_*(L1,L2):-
     select(***,L1,L2),!.
elimina_*(L1,L1).

depura_terms([],***).
depura_terms([T],T).
```

```
es_variable(V):-
     atom(V),
     name(V,[120|_]).

% subconjunto(Conj, Subconjunto)
% El predefinido "subset" sólo comprueba, no genera.

subconjunto([],[]).
subconjunto([X|L1],[X|L2]):-
     subconjunto(L1,L2).
subconjunto([_|L1],L2):-
     subconjunto(L1,L2).

partes(Conjunto,Partes):-
     setof(Sub,subconjunto(Conjunto,Sub),Partes).

% diferencia(C1,C2,Dif)   (Dif es la cláusula formada por los
% literales de C1 que no están en C2)

diferencia([],_,[]).
diferencia([X|L1],C2,L2):-
     memberchk(X,C2),
     !,
     diferencia(L1,C2,L2).
diferencia([X|L1],C2,[X|L2]):-
     diferencia(L1,C2,L2).

max_list(L,N):-
     aux_max_list(L,0,N).

aux_max_list([],N,N).
aux_max_list([X|L],A,N):-
     X >= A,
     !,
     aux_max_list(L,X,N).
aux_max_list([_|L],A,N):-
     aux_max_list(L,A,N).

ocurre(V,T):-
     posicion(_,T,V).

subtermino_en_literal(Lit,Subtermino):-
```

```
      atomo(Lit,Atomo,_),
      Atomo =.. [_|Argumentos],
      member(Arg,Argumentos),
      ocurre(Subtermino,Arg).

subtermino_en_clausula(Clausula,Subtermino):-
      member(Lit,Clausula),
      subtermino_en_literal(Lit,Subtermino).

lista_de_posiciones([],_,[]).
lista_de_posiciones([L|Resto_1],X,[Pos|Resto_2]):-
      posiciones(L,X,Pos),
      lista_de_posiciones(Resto_1,X,Resto_2).

indice(Var,N):-
      name(Var,[120|Resto]),
      name(N,Resto).

crea_variable(Indice,Variable):-
      name(Indice,Indice_ascii),
      name(Variable,[120|Indice_ascii]).

nuevo_indice([],1).
nuevo_indice([X|L],N):-
      sort([X|L],Ordenadas),
      last(Var,Ordenadas),
      indice(Var,N1),
      N is N1 + 1.

variables_en_clausula(Clausula,Vars):-
      variables_en_termino(Clausula,Vars).

% Máximo de una lista no vacía (con infinito),

max_list_inf([X|L],Max):-
      max_list_inf_aux(L,X,Max).

max_list_inf_aux(_,infinito,infinito):- !.
max_list_inf_aux([],X,X).
max_list_inf_aux([Y|L],A,Max):-
      max_inf(Y,A,Z),
      max_list_inf_aux(L,Z,Max).
```

```
max_inf(infinito,_,infinito):- !.
max_inf(_,infinito,infinito):- !.
max_inf(X,Y,Z):- Z is max(X,Y).

% Mínimo de una lista no vacía (con infinito),

min_list_inf([X|L],Min):-
     min_list_inf_aux(L,X,Min).

min_list_inf_aux(_,0,0):-!.
min_list_inf_aux([],X,X).
min_list_inf_aux([Y|L],A,Min):-
     min_inf(Y,A,Z),
     min_list_inf_aux(L,Z,Min).

min_inf(infinito,X,X):- !.
min_inf(X,infinito,X):- !.
min_inf(X,Y,Z):- Z is min(X,Y).

% n_arg(?N,+Term,?Arg)
% Evita los errores producidos por arg/3.
%
%  ?- arg(N,a,Arg).
%     No
%
%  ?- arg(N,0,Arg).
%     ERROR: arg/3: Type error: 'compound' expected, found '0'
%

n_arg(N,Term,Arg):-
     Term=..[_|Args],
     nth1(N,Args,Arg).


%  ?- n_arg(N,a,Arg).
%     No
%
%  ?- n_arg(N,0,Arg).
%     No

% posiciones(Literal,Termino,Posiciones)
% Dado un Literal y un Término devuelve la lista de todas las
% posiciones del Literal ocupadas por el Término.
```

```
% Consideramos que cualquier término ocupa la posición [] en
% sí mismo. Ejemplo de uso:
%
%    ?- posiciones(p(x1,x2,h(x1,x1)),x1,L).
%       L = [[1], [3, 1], [3, 2]] ;
%       No
%
%    ?- posiciones(x1,x1,L).
%       L = [[]] ;
%       No
%
%    ?- posiciones(p(x1,x2,h(x1,x1)),a,L).
%       L = [] ;
%       No
%

posiciones(Literal,Termino,Posiciones):-
     atomo(Literal,Atomo,_),
     findall(Pos,posicion(Pos,Atomo,Termino),Posiciones).

% aplica_a_literal(L_in,Sust,L_out)
% Toma como entrada un literal L_in y una sustitución Sust
% representada como una lista [x1 --> t1, ... , xn --> tn] y
% devuelve el resultado L_out de aplicar a L_in la
% sustitución. Ejemplo de uso:
%
%    ?- aplica_a_literal(-p(x1,x2,x3),
%                        [x1 --> f(x1), x2 --> x1, x3 --> a],L).
%            L = -p(f(x1), x1, a) ;
%       No
%
%    ?- aplica_a_literal(x1,[x1 --> x2],T).
%            T = x1 ;
%       No
%

aplica_a_literal(L_in,Sust,L_out):-
     extrae_posiciones(L_in,Sust,Pares),
     aux_aplica(L_in,Pares,L_out).

extrae_posiciones(L,Sust,Pares):-
     findall(par(Pos,Term),
             (member(X --> Term, Sust), posiciones(L,X,Pos)),
```

```
          Pares).

aux_aplica(L_acum,[],L_acum).
aux_aplica(L_acum,[par(Pos,Term)|Resto],L_out):-
     inserta_en_posiciones(L_acum,Pos,Term,Nuevo_acum),
     aux_aplica(Nuevo_acum,Resto,L_out).

% aplica_a_clausula(C1,Sust,C2)
% Toma como entrada una cláusula y una sustitución Sust
% representada como una lista [X_1 --> T_1, ... , X_n --> T_n]
% y devuelve la cláusula C2 resultado de aplicar a C1 la
% sustitución. Ejemplo de uso:
%
%  ?- aplica_a_clausula(
%                [p(x1,f(x1)),p(x2,f(x1)),q(x3,x4)],    % C1
%                [x2 --> x1, x3 --> x4, x4 --> a],      % Sust
%                C2).
%            C2 = [p(x1, f(x1)), q(x4, a)]
%        Yes
%

aplica_a_clausula(C1,Sust,C2):-
     aplica_a_clausula_aux(C1,Sust,C2_aux),
     list_to_set(C2_aux,C2).

aplica_a_clausula_aux([],_,[]).
aplica_a_clausula_aux([L1|Resto_1],Sust,[L2|Resto_2]):-
     aplica_a_literal(L1,Sust,L2),
     aplica_a_clausula_aux(Resto_1,Sust,Resto_2).

% literales_inversa(L,C1,Theta,Literales)
% Siguiendo el lema, tomamos como dato de entrada un literal
% L, una cláusula C1 y una sustitución Theta. Literales es la
% lista de literales L1 de C1 tales que al aplicarles Theta
% obtenemos L1. Ejemplo de uso
%
%   ?- literales_inversa(
%         p(a,x1),                                % Lit
%         [p(a,x1),p(x3,x1),p(a,x2),p(a,x4)],    % C1
%         [x3 --> a,x2 --> x1],                  % Sust
%         Literales).
%
%        Literales = [p(a, x1), p(x3, x1), p(a, x2)] ;
```

```
%     No
%

literales_inversa(L,C1,Theta,Literales):-
     findall(L1,(
                member(L1,C1),aplica_a_literal(L1,Theta,L)),
                Literales).

% nuevo_identificador(-N).
% Cuando vamos a crear una nueva asignación necesitamos un
% identificador que no se haya usado.

nuevo_identificador(N):-
     aux_nuevo_identificador(1,N).

aux_nuevo_identificador(M,N):-
     delta(M,_,_),
     !,
     M1 is M+1,
     aux_nuevo_identificador(M1,N).
aux_nuevo_identificador(N,N).

% variables_en_termino(Term,Lista_de_variables)
% Dado un término devuelve la lista de variables que ocurren
% en el término
%
%    ?- variables_en_termino(p(x2,a,f(x1,b)), Lista).
%              Lista = [x1, x2] ;
%       No
%

variables_en_termino(Term,Lista_de_variables):-
     setof_0(Var,
                (ocurre(Var,Term),es_variable(Var)),
                 Lista_de_variables).

% variables_en_sustitucion(Sust,Lista_de_variables,Max)
% Toma una sustitución y devuelve la lista de todas las
% variables que ocurren en la sustitución
%
%  ?- variables_en_sustitucion(
%        [x1 --> f(x1,x3), x2 --> x4, x6 --> a],  % Sust
%        Lista).
```

```
%     Lista = [x1, x2, x3, x4, x6] ;
%
%         No

variables_en_sustitucion(Sust,Lista_de_variables):-
     variables_en_termino(Sust,Lista_de_variables).

% variables_nuevas(+Sust,+Otras,?Var,-Var_nuevas)
% Toma como dato de entrada una sustitución y una lista de
% variables y devuelve la lista de variables que ocurren en
% la sustitución (Var) y una lista Var_nuevas de variables
% que no ocurren en la susitución ni en la lista Otras.
% Genera tantas variables nuevas tantas como aparezcan en la
% lista Var.  Ejemplo:
%
%    ?- variables_nuevas([x1 --> f(x2), x2 --> x3],
%                        [x3,x4],
%                        Var,
%                        Var_nuevas).
%
%          Var = [x1, x2, x3]
%          Var_nuevas = [x5, x6, x7] ;
%       No
%

variables_nuevas(Sust,Otras,Var,Var_nuevas):-
     variables_en_termino(Sust,Var),
     append(Var,Otras,Todas),
     nuevo_indice(Todas,Indice),
     crea_variables_nuevas(Var,Indice,Var_nuevas).

crea_variables_nuevas([],_,[]).
crea_variables_nuevas([_|Vs],Indice,[Nueva|Ns]):-
     crea_variable(Indice,Nueva),
     Nuevo_indice is Indice + 1,
     crea_variables_nuevas(Vs,Nuevo_indice,Ns).

% aplicacion_lista_de sustituciones(C1,Lista,C2)
% Toma como dato de entrada una cláusula y una lista de
% sustituciones y devuelve la cláusula obtenida por la
% aplicación sucesiva de las sustituciones que aparecen en la
% lista. Ejemplo:
%
```

```
%  ?- aplicacion_lista_de_sustituciones(
%       [p(x1,x2),q(x1,x3)],                      % Cláusula
%       [[x1 --> x2],[x2 --> f(x1), x3 --> a]],   % Lista sust.
%       C).
%
%       C = [p(f(x1), f(x1)), q(f(x1), a)] ;
%
%           No
%

aplicacion_lista_de_sustituciones(C,[],C).
aplicacion_lista_de_sustituciones(C1,[S|Lista],C2):-
       aplica_a_clausula(C1,S,C_aux),
       aplicacion_lista_de_sustituciones(C_aux,Lista,C2).

% Ejemplo .-
% Veamos con un ejemplo que la imagen de las variables del
% dominio de una sustitución son las mismas una vez
% descompuesta la sustitución en sustituciones elementales
%
%  ?- descompone_sustitucion(
%       [x1 --> f(x1,x3), x2 --> x4, x3 --> a],  % Sust
%       _Lista),                                 % Lista elem.
%       aplicacion_lista_de_sustituciones(
%       [p(x1,x2,x3)],                           % C1
%       _Lista,                                  % Lista elem.
%       [p(f(x1,x3),x4,a)]).                     % C2
%
%          Yes
%

% Subsunción

% subsuncion_literales(L1,L2,Sust)
% Recibe como entrada los literales L1 y L2 y devuelve, si
% existe, una sustitución que aplicada a L1 nos da L2.
% Ejemplo:
%
%    ?- subsuncion_literales(
%                      -q(x1,x2,f(x1)),
%                      -q(a,x1,f(a)),
%                       Sust).
%            Sust = [x1-->a, x2-->x1] ;
```

```
%
%       No
%

subsuncion_literales(L1,L2,Sust):-
     subsuncion_literales_aux(L1,L2,[],Sust).

subsuncion_literales_aux(L1,L2,Acum,Sust):-
     L1 = L2  -> Sust = Acum
   ;
  (es_variable(L1) -> extiende(L1 --> L2,Acum,Sust)
   ;
     L1 =.. [F,A1|Arg_1],
     L2 =.. [F,A2|Arg_2],
     sustitucion_lista_literales(
             [A1|Arg_1],[A2|Arg_2],Acum,Sust)).

sustitucion_lista_literales([],[],Acum,Acum).
sustitucion_lista_literales([T1|Arg_1],
                            [T2|Arg_2],
                            Acum,
                            Sust):-
     subsuncion_literales(T1,T2,Sust_1),
     extiende_lista(Sust_1,Acum,Acum_aux),
     sustitucion_lista_literales(Arg_1,Arg_2,
                              Acum_aux,Sust).

extiende(X --> T1,Acum,Nuevo_acum):-
     member(X --> T2, Acum) ->
        (T1 = T2, Nuevo_acum = Acum)
   ;
     Nuevo_acum = [X --> T1|Acum].

extiende_lista([],Acum,Acum).
extiende_lista([Enlace|Resto],Acum,Nueva_sust):-
     extiende(Enlace,Acum,Acum_1),
     extiende_lista(Resto,Acum_1,Nueva_sust).


% Una lista de enlaces es función si no encontramos
% dos enlaces con el mismo origen y distinta imagen

funcion([]).
```

```
funcion([X --> Y |Resto]):-
     member(X --> T, Resto),
     !,
     T == Y,
     funcion(Resto).
funcion([_|Resto]):-
     funcion(Resto).

% subsuncion_clausulas(C1,C2,Sust)
% Recibe como entrada las cláusulas C1 y C2 y devuelve, si
% existe, una sustitución que apilicada a C1 nos devuelva un
% subconjunto de C2. Ejemplo:
%
%      ?- subsuncion_clausulas([p(x1)],[p(a),p(b)],Sust).
%
%             Sust = [x1-->a] ;
%
%             Sust = [x1-->b] ;
%         No
%
%      ?- subsuncion_clausulas(
%                [p(x1),q(f(x1),x3),r(x2)],        % C1
%                [p(a),p(b),q(f(a),x4),r(x2)],     % C2
%                Sust).
%
%             Sust = [x3-->x4, x1-->a] ;
%         No
%

subsuncion_clausulas(C1,C2,Sust):-
     subsuncion_clausulas_aux(C1,C2,[],Sust),
     aplica_a_clausula(C1,Sust,C1_sust),
     subset(C1_sust,C2).

subsuncion_clausulas_aux([],_,Sust,Sust).
subsuncion_clausulas_aux([L1|Resto_1],C2,Acum,Sust):-
     member(L2,C2),
     subsuncion_literales(L1,L2,Sust_lit),
     extiende_lista(Sust_lit,Acum,Nuevo_acum),
     subsuncion_clausulas_aux(Resto_1,C2,Nuevo_acum,Sust).

% aplica_lista_operadores(C2,Lista_de_operadores,C1)
% Aplicamos la Lista_de_operadores a C2 para obtener C1.
```

```
% Por ejemplo,
%
%      ?- genera_operadores_clausales(
%                          [p(x1,x3),q(x3)],          % C1
%                          [p(x2,x3),q(x3),r(x1)],    % C2
%                          Lista),
%         aplica_lista_operadores(
%                          [p(x2,x3),q(x3),r(x1)],    % C2
%                          Lista,
%                          C1).
%
%    Lista = [op(8, x1), op(11, x5), op(10, x4), op(9, x1)]
%    C1    = [p(x1, x3), q(x3)] ;

aplica_lista_operadores(C1,Lista_operadores,C2):-
     aux_aplica_lista_operadores(C1,Lista_operadores,C2).


aux_aplica_lista_operadores(C,[],C).
aux_aplica_lista_operadores(C1,[OP|Resto],C2):-
     operador_clausal(OP,C1,C_aux),
     aux_aplica_lista_operadores(C_aux,Resto,C2).

% soporte(+Id,-Soporte)
% Devuelve los literales tales que la asignación delta les
% asigna un conjunto distinto de vacío
%
%       ?- soporte(1,L).
%              L = [p(f(a), b, a), q(a)] ;
%          No
%
%       ?- soporte(2,L).
%              L = [-_G263]
%          Yes
%

soporte(Id,Soporte):-
     recoge(Id,Pares),
     aux_soporte(Pares,Soporte).

aux_soporte([],[]).
aux_soporte([par(_,[])|Resto],Soporte):-
     !,
```

```
     aux_soporte(Resto,Soporte).
aux_soporte([par(X,_)|Resto],[X|Soporte]):-
     aux_soporte(Resto,Soporte).

% Auxiliar.
% Además de la distancia obtenemos una sustitución
% minimal. Sólo para el caso en que se verifique
% sustitución sin contención

d_subsuncion_2(C,D,Dist,Minimal):-
     setof(Peso-Desc,peso_candidato_2(C,D,Peso,Desc),
                     [Dist-Minimal|_]).

peso_candidato_2(C,D,Peso,Desc):-
     subsuncion_clausulas(C,D,Sust),
     variables_en_clausula(C,Var_C),
     peso(Sust,Var_C,Peso,Desc).

encuentra_variable_nueva(_,Var,Nueva):-
       nuevo_indice(Var,Indice),
       crea_variable(Indice,Nueva),
     !.

lista_de_subterminos(Ternas,Subterminos):-
     setof_0(S,ocurre_en_alguna_imagen(Ternas,S),Subterminos).

ocurre_en_alguna_imagen(Ternas,S):-
     member(terna(_,T,_),Ternas),
     ocurre(S,T).

nueva_sustitucion_1([],_,_,[]).
nueva_sustitucion_1([terna(Y,T1,_)|Resto_1],Sub_T,
                  Var,[terna(Y,T2,Var_T2)|Resto_2]):-
     posiciones(T1,Sub_T,Pos),
     subconjunto(Pos,Subconjunto),
     inserta_en_posiciones(T1,Subconjunto,Var,T2),
     variables_en_termino(T2,Var_T2),
     nueva_sustitucion_1(Resto_1,Sub_T,Var,Resto_2).

dominio_y_ran(Ternas,Dom,Ran):-
     findall(X,member(terna(X,_,_),Ternas),Dom),
     findall(Z,(member(terna(_,_,Var_T),Ternas),
                member(Z,Var_T)),Ran_1),
```

```
     sort(Ran_1,Ran).

crea_ternas([],[]).
crea_ternas([X --> T |RS],[terna(X,T,Lista)|RV]):-
     variables_en_termino(T,Lista),
     crea_ternas(RS,RV).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejemplos de asignaciones (Fichero asignaciones.pl)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- dynamic delta/3.

% Asignación vacía

delta(0,_,[]).

% Ejemplo 1: Es asignación de variables y el término que
% permite la compatibilidad es Termino = a.

delta(1,p(f(a),b,a),[[],[[3]],[[1,1]]]):-!.
delta(1,q(a),[[]]):-!.
delta(1,_,[]).

% Ejemplo 2: Es asignación de variables, pero la
% compatibilidad no viene determinada por un único término

delta(2,-_,[[]]):-!.
delta(2,_,[]).

% Ejemplo 3: No es asignación de variables

delta(3,p(f(a),b,a),[[],[[3]],[[1,1]]]):-!.
delta(3,q(b),[[1]]):-!.
delta(3,_,[]).

% Otros ejemplos

delta(4,p(f(x1,x3),x2),[[[1]]]):- !.
delta(4,q(h(f(x1,x3))),[[[1,1]]]):- !.
delta(4,-r(x2),[[]]):- !.
delta(4,_,[]).
```

```
delta(5,p(f(c),x1,c),[[],[[3]],[[1,1]]]):-!.
delta(5,q(h(c,x2)),[[[1,1]]]):- !.
delta(5,q(c),[[]]):-!.
delta(5,_,[]).

delta(6,p(f(f(x2)),x1,f(x2)),[[],[[3]],[[1,1]]]):-!.
delta(6,q(h(f(x2),x2)),[[[1,1]]]):- !.
delta(6,q(f(x2)),[[]]):-!.
delta(6,_,[]).

delta(7,_,[]).

delta(8,s(x1,x1),[[]]):-!.
delta(8,_,[]).

delta(9,s(x1,x1),[[[1]]]):-!.
delta(9,_,[[]]).

delta(10,s(a,x1),[[[2]]]):-!.
delta(10,_,[[]]).

delta(11,r(x2),[[[1]]]):-!.
delta(11,_,[[]]).
```

Bibliografía

[AL97] A. Aliseda-LLera. *Seeking Explanations: Abduction in Logic, Philosophy of Science and Artificial Intelligence.* Tesis Doctoral, Institute for Logic, Language and Computation. Universiteit van Amsterdam, 1997.

[Apt97] K. R. Apt. *From logic programming to Prolog.* Prentice-Hall, Inc., 1997.

[Ban87] R. B. Banerji. Learning in the limit in a growing language. En John McDermott, editor, *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, páginas 280–282. Morgan Kaufmann, 1987.

[Bax77] L.D Baxter. The NP-completeness of subsumption. No publicado, 1977.

[BG95] F. Bergadano y D. Gunetti. *Inductive Logic Programming: From Machine Learning to Software Engineering.* The MIT Press, 1995.

[BGA56] J.S. Bruner, J.J Goodnow, y G.A. Austin. *A study of thinking.* Willey, 1956.

[Car50] R. Carnap. *Logical Foundations of Probability.* Routledge & Kegan Paul, London, 1950.

[Car52] R. Carnap. *The Continuum of Inductive Methods.* Chicago University, 1952.

[CM85] E. Charniak y D. McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley Publishing Co., 1985.

[Coh93] W.W. Cohen. PAC-learning a restricted class of recursive logic programs. En S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, páginas 73–86. J. Stefan Institute, 1993.

[CP86] P. T. Cox y T. Pietrzykowski. Causes for events: Their computation and applications. En J. Siekmann, editor, *Proceedings, Eighth International Conference on Automated Deduction (CADE-8)*, páginas 608–621. Springer-Verlag, 1986.

[DB92] L. De Raedt y M. Bruynooghe. An overview of the interactive concept-learner and theory revisor CLINT. En S. H. Muggleton, editor, *Inductive Logic Programming*, páginas 163–191. Academic Press, 1992.

[DIJ94] B. Dolšak, Bratko I., y A. Jezernik. Finite element mesh design: An engineering domain for ILP application. En S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volumen 237 de *GMD-Studien*, páginas 305–320. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[DLCD82] T. G. Dietterich, R. L. London, K. Clarkson, y G. Dromey. Learning and inductive inference. En *The Handbook of Artificial Intellegence*, volumen 3, páginas 323–512. William Kaufman, Inc., 1982.

[DR91] L. De Raedt. *Interactive Concept-Learning.* Tesis Doctoral, Department of Computer Science, Katholieke Universiteit Leuven, 1991.

[DRB89] L. De Raedt y M. Bruynooghe. Towards friendly concept-learners. En *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, páginas 849–856. Morgan Kaufmann, 1989.

[DRB92] L. De Raedt y M. Bruynooghe. A unifying framework for concept-learning algorithms. *The Knowledge Engineering Review*, 7(3):251–269, 1992.

[Dž95] S. Džeroski. *Numerical constraints and learnability in inductive logic programming.* Tesis Doctoral, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia, 1995.

[EK92] C. Evans y A. C. Kakas. Hypothetico-deductive reasoning. En Institute for New Generation Computer Technology (ICOT), editor, *Proceedings of the International Conference on Fifth Generation Computer Systems.*, volumen 2, páginas 546–554. IOS Press, 1992.

[EM97] T. Eiter y H. Mannila. Distance measures for point sets and their computation. *Acta Informatica*, 34(2):109–133, 1997.

[Fen92] C. Feng. Inducing temporal fault diagnostic rules from a qualitative model. En S. Muggleton, editor, *Inductive Logic Programming*, páginas 473–494. Academic Press, 1992.

[FK98] P.A. Flach y A.C. Kakas. Abduction and induction in AI: Report of the IJCAI'97 workshop. *Logic Journal of the Interest Group on Pure and Applied Logic*, 6(4):651–656, 1998.

[FK00a] P. Flach y A. Kakas. The cycle of abductive and inductive knowledge development. En K. Furukawa, S. Muggleton, D. Michie, y L. De Raedt, editors, *Proceedings of the Machine Intelligence 17 workshop*, páginas 17–24. Keio University, 2000.

[FK00b] P. A. Flach y A. C. Kakas, editors. *Abduction and Induction: Essays on their relation and integration.* Kluwer Academic Publishers, 2000.

[FK00c] P. A. Flach y A. C. Kakas. On the relation between abduction and inductive learning. En Dov M. Gabbay y Rudolf Kruse, editors, *Handbook of defeasible reasoning and uncertainty management systems, Vol. 4: Abductive reasoning and learning*, páginas 1–33. Kluwer Academic Publishers, 2000.

[Fla92a] P. A. Flach. A framework for inductive logic programming. En S. H. Muggleton, editor, *Inductive Logic Programming*, páginas 193–211. Academic Press, 1992.

[Fla92b] P.A. Flach. Logical approaches to machine learning - An overview. *THINK*, 1(2):25–36, 1992.

[Fla94] P.A. Flach. Inductive logic programming and philosophy of science. En S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volumen 237 de *GMD-Studien*, páginas 71–84. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[Fla95] P. Flach. *Conjectures - an inquiry concerning the logic of induction.* Institute for Language Technology and Artificial Intelligence, Tilburg, the Netherlands, 1995.

[Fré06] M. Fréchet. *Sur quelques points du calcul fonctionnel*, volumen 22. Reudicont del Circulo Matematico di Palermo, 1906.

[Für93] J. Fürnkranz. Avoiding noise fitting in a FOIL-like learning algorithm. En F. Bergadano, L. De Raedt, S. Matwin, y S. Muggleton, editors, *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, páginas 14–23. Morgan Kaufmann, 1993.

[Für94] J. Fürnkranz. A comparison of pruning methods for relational concept learning. En *Proceedings of the AAAI-94 Workshop on Knowledge Discovery in Databases*, 1994.

[Gal86] J. H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*, volumen 5 de *Computer Science and Technology Series.* Harper & Row, 1986.

[GJ79] M. R. Garey y D. S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[GL01] D. Gamberger y N. Lavrac. Filtering noisy instances and outliers. En H. Liu y H. Motoda, editors, *Instance Selection and Construction for Data Mining*, páginas 375–394. Kluwer Academic Publishers, 2001.

[GNAJBD00] M. A. Gutiérrez-Naranjo, J. A. Alonso-Jiménez, y J. Borrego-Díaz. A topological study of the upward refinement operators in ILP.

En J. Cussens y A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, páginas 120–137, 2000.

[Gol67] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[Got87] G. Gottlob. Subsumption and implication. *Information Processing Letters*, 24(2):109–111, 1987.

[Gou50] T. A. Goudge. *The Thought of C. S. Peirce.* Dover Publications Inc., New York, 1950.

[Hau14] F. Hausdorff. *Grundzüge der Mengenlehre.* Leipzig, 1914.

[Hel89] N. Helft. Induction as nonmonotonic inference. En *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, páginas 149–156. Morgan Kaufmann, 1989.

[Hem43] C.G. Hempel. A purely syntactical definition of confirmation. *J. Symbolic Logic*, 6(4):122–143, 1943.

[Hem45] C.G. Hempel. Studies in the logic of confirmation. *Mind*, 54(213 (Part I) – 214 (Part II)):1–26 (Part I) 97–121 (Part II), 1945.

[HJ84] K. Hrbacek y T. Jech. *Introduction to Set Theory. Second edition.* Marcel Dekker, Inc, 1984.

[Hue76] G. Huet. *Resolution d'Equations dans les langages d'ordre 1, 2, ..., ω.* Tesis Doctoral, Université de Paris VII, 1976.

[Hue80] G. Huet. Confluent reductions : Abstract properties and applications to term rewriting systems. *Journal of the Association for Computing Machinery*, 27(4):797–821, 1980.

[Hul20] C.L. Hull. Quantitative aspects of the evolution of concepts: An experimental study. *Psycological Monographs*, 28, 1920.

[Hut97] A. Hutchinson. Metrics on terms and clauses. En Maarten van Someren y Gerhard Widmer, editors, *Proceedings of the 9th European Conference on Machine Learning*, volumen 1224 de *Lecture Notes in Artificial Intelligence*, páginas 138–145. Springer, 1997.

[Kaz99] D. Kazakov. *Natural Language Processing Applications of Machine Learning.* Tesis Doctoral, Department of Cybernetics, Czech Technical University, Prague, 1999.

[Kie93] J-U. Kietz. Some lower bounds for the computational complexity of inductive logic programming. En P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volumen 667 de *Lecture Notes in Artificial Intelligence*, páginas 115–123. Springer-Verlag, 1993.

[KK01] A. Karwath y R. D. King. An automated ilp server in the field of bioinformatics. En Céline Rouveirol y Michèle Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volumen 2157 de *Lecture Notes in Artificial Intelligence*, páginas 91–103. Springer-Verlag, 2001.

[KKCD00] R. D. King, A. Karwath, A. J. Clare, y L. Dehaspe. Genome scale prediction of protein functional class from sequence using data mining. En R. Ramakrishnan, S. Stolfo, R. Bayardo, y I. Parsa, editors, *The Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, páginas 384–389. The Association for Computing Machinery, 2000.

[KKT93] A. C. Kakas, R. A. Kowalski, y F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.

[KL94] J.-U. Kietz y M. Lübbe. An efficient subsumption algorithm for inductive logic programming. En W. Cohen y H. Hirsh, editors, *Proc. Eleventh International Conference on Machine Learning (ML-94)*, páginas 130–138, 1994.

[Kle97] S. B. Klein. *Aprendizaje. Principios y aplicaciones (Segunda edición).* McGraw Hill, 1997.

[KM90] A. C. Kakas y P. Mancarella. Database updates through abduction. En Dennis McLeod, Ron Sacks-Davis, y H.-J Schek, editors, *Very large data bases: 16th International Conference on Very Large Data Bases, Brisbane, Australia*, páginas 650–661. Morgan Kaufmann Publishers, 1990.

[KME99] D. Kazakov, S. Manandhar, y T. Erjavec. Learning word segmentation rules for tag prediction. En S. Džeroski y P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volumen 1634 de *Lecture Notes in Artificial Intelligence*, páginas 152–161. Springer-Verlag, 1999.

[KS95] R.D. King y A. Srinivasan. Relating chemical activity to structure: An examination of ILP successes. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):411–434, 1995.

[KW92] J-U. Kietz y S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. En S. Muggleton, editor, *Inductive Logic Programming*, páginas 335–359. Academic Press, 1992.

[Lan96] P. Langley. *Elements of Machine Learning.* Morgan Kaufmann, 1996.

[Law91] J.D. Lawson. Order and strongly sober compactifications. En A.W. Roscoe G.M. Reed y R.F. Wachter, editors, *Topology and Category Theory in Computer Science*, páginas 179–205. Oxford University Press, 1991.

[LD94] N. Lavrač y S. Džeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1994.

[LDG91] N. Lavrač, S. Džeroski, y M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. En Y. Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volumen 482 de *Lecture Notes in Artificial Intelligence*, páginas 265–281. Springer-Verlag, 1991.

[Lin91] C.X. Ling. Inventing necessary theoretical terms. Informe técnico Nr. 302, Dept. of Computer Science, University of Western Ontario, 1991.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer, 2 edición, 1987.

[LM92] S. Lapointe y S. Matwin. Sub-unification: A tool for efficient induction of recursive programs. En D. Sleeman y P. Edwards, editors, *Proceedings of the 9th International Workshop on Machine Learning*, páginas 273–281. Morgan Kaufmann, 1992.

[LN92] C.X. Ling y M.A. Narayan. A critical comparison of various methods based on inverse resolution. En S. Muggleton, editor, *Inductive Logic Programming*, páginas 131–144. Academic Press, 1992.

[LR95] N. Lavrač y L. De Raedt. Inductive logic programming: A survey of European research. *AI Communications*, 8(1):3–19, 1995.

[Mah86] M. J. Maher. Equivalences of logic programs. En *Proceedings of Third International Conference on Logic Programming.* Springer, 1986.

[Mah88] M. J. Maher. Equivalence of logic programs. En J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, páginas 627–658. Morgan Kaufmann Pub., 1988.

[MB88] S. H. Muggleton y W. Buntine. Machine invention of first-order predicates by inverting resolution. En *Proc. Fifth International Conference on Machine Learning*, páginas 339–352. Morgan Kaufmann, 1988.

[MB92] S. Muggleton y W. Buntine. Machine invention of first-order predicates by inverting resolution. En S. Muggleton, editor, *Inductive Logic Programming*, páginas 261–280. Academic Press, 1992.

[MDR94] S. Muggleton y L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[MF90] S. Muggleton y C. Feng. Efficient induction of logic programs. En *Proceedings of the 1st Conference on Algorithmic Learning Theory*, páginas 368–381. Ohmsma, Tokyo, Japan, 1990.

[MF92] S. Muggleton y C. Feng. Efficient induction of logic programs. En S. Muggleton, editor, *Inductive Logic Programming*, páginas 281–298. Academic Press, 1992.

[Mit82] T. M. Mitchell. Generalisation as search. *Artificial Intelligence*, 18:203–226, 1982.

[Mit97] T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[MP92] J. Marcinkowski y L. Pacholski. Undecidability of the Horn-clause implication problem. En IEEE, editor, *33rd Annual Symposium on Foundations of Computer Science. Pittsburgh, Pennsylvania: proceedings [papers]*, páginas 354–362. IEEE Computer Society Press, 1992.

[MP94] S. Muggleton y C.D. Page. Self-saturation of definite clauses. En S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volumen 237 de *GMD-Studien*, páginas 161–174. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[Mug91] S. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8(4):295–318, 1991.

[Mug92] S. Muggleton. Inductive logic programming. En S. Muggleton, editor, *Inductive Logic Programming*, páginas 3–27. Academic Press, 1992.

[Mug94] S. Muggleton. Predicate invention and utilization. *Journal of Experimental and Theoretical Artificial Intelligence*, 6(1):121–130, 1994.

[Mug95] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[MWKE93] K. Morik, S. Wrobel, J-U. Kietz, y W. Emde. *Knowledge Acquisition and Machine Learning: Theory, Methods and Applications.* Academic Press, 1993.

[NC97] S-H. Nienhuys-Cheng:. Distance between herbrand interpretations: a measure for approximations to a target concept. Informe técnico EUR–FEW–CS–97–05, Department of Computer Science, Erasmus University, 1997.

[NCdW96] S-H. Nienhuys-Cheng y R. de Wolf. The subsumption theorem in inductive logic programming: Facts and fallacies. En L. De Raedt, editor, *Advances in Inductive Logic Programming*, páginas 265–276. IOS Press, 1996.

[NCdW97] S.-H. Nienhuys-Cheng y R. de Wolf. *Foundations of inductive logic programming*, volumen 1228 de *Lecture Notes in Artificial Intelligence.* Springer-Verlag Inc., 1997.

[Pau93] G. Paul. Approaches to abductive reasoning: an overview. *Artificial Intelligence Review*, 7:109–152, 1993.

[Pea87] J. Pearl. Embracing causality in formal reasoning. En Kenneth Forbus y Howard Shrobe, editors, *Proceedings of the Seventh National Conference on Artificial Intelligence*, páginas 369–373. American Association for Artificial Intelligence, AAAI Press, 1987.

[Pei32] C.S. Peirce. Elements of logic. En C. Hartshorne y P. Weiss, editor, *Collected Papers of Charles Sanders Peirce*, volumen 2. Harvard University Press, 1932.

[Pei33] C. S. Peirce. *C. S. Peirce Collected Papers*, volumen 1-6. C. Hartshorne and P. Weiss, Harvard Univ. Press, Cambridge, 1933.

[Plo70] G. D. Plotkin. A note on inductive generalisation. En B. Meltzer y D. Michie, editors, *Machine Intelligence 5*, páginas 153–163. Elsevier North Holland, 1970.

[Plo71] G. D. Plotkin. *Automatic Methods of Inductive Inference.* Tesis Doctoral, Edinburgh University, 1971.

[Pol45] G. Polya. *How to Solve It: A New Aspect of Mathematical Method.* Princeton University Press, 1945.

[Poo88] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27–47, 1988.

[Pop58] K. Popper. *Logic of Scientific Discovery.* Hutchinson, London, 1958.

[Pop63] K. Popper. *Conjectures and Refutations. The Growth of Scientific Knowledge.* Routledge, London and New York, Quinta edición, 1963.

[Pop73] H. E. Pople, Jr. On the mechanization of abductive logic. En Nils J. Nilsson, editor, *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, páginas 147–152. William Kaufmann, 1973.

[Pop98] L. Popelinsky. Knowledge discovery in spatial data by means of ilp. En J. M. Zytkow y M. Quafafou, editors, *Principles of Data Mining and Knowledge Discovery. PKDD'98 Nantes France.*, volumen 1510 de *Lecture Notes in Computer Science*, páginas 271–279. Springer Verlag, 1998.

[PvdL92] S.-H. Nienhuys-Cheng P.R.J. van der Laag. Subsumption and refinement in model inference. Informe técnico EUR–FEW–CS–92–07, Department of Computer Science, Erasmus University, 1992.

[Qui90] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[RB92] L. De Raedt y M. Bruynooghe. Belief updating from integrity constraints and queries. *Artificial Intelligence*, 53(2-3):291–307, 1992.

[RB98] J. Ramon y M. Bruynooghe. A framework for defining distances between first–order logic–objects. Informe técnico CW 263, Department of Computer Science, Katholieke Universiteit Leuven, 1998.

[RD94] L. De Raedt y S. Džeroski. First-order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70(1–2):375–392, 1994.

[Rey69] J. C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. En Bernard Meltzer y Donald Michie, editors, *Machine Intelligence 5*, páginas 135–151. Edinburgh University Press, 1969.

[Rob65] J. Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[SB86] C. Sammut y R.B Banerji. Learning concepts by asking questions. En R. Michalski, J. Carbonnel, y T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach. Vol. 2*, páginas 167–192. Kaufmann, 1986.

[SG89] A. Sattar y R. Goebel. Using crucial literals to select better theories. Informe técnico, Dept. of Computer Science, University of Alberta, Canada, 1989.

[Sha81] E. Y. Shapiro. Inductive inference of theories from facts. Informe técnico 192, Yale University Department of Computer Science, 1981.

[Sha83] E.Y. Shapiro. *Algorithmic program debugging.* MIT Press, 1983.

[Smy91] M.B. Smyth. Totally bounded spaces and compact ordered spaces as domains of computation. En A.W. Roscoe G.M. Reed y R.F. Wachter, editors, *Topology and Category Theory in Computer Science*, páginas 207–229. Oxford University Press, 1991.

[Sri01] A. Srinivasan. Aleph. A learning engine for proposing hypotheses. En `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html` , 2001.

[SS88] M. Schmidt-Schauss. Implication of clauses is undecidable. *Theoretical Computer Science*, 59(3):287–296, 1988.

[Sta93] I. Stahl. Predicate invention in ILP - An overview. En P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine*

Learning, volumen 667 de *Lecture Notes in Artificial Intelligence*, páginas 313–322. Springer-Verlag, 1993.

[Sta94] I. Stahl. On the utility of predicate invention in inductive logic programming. En F. Bergadano y L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning*, volumen 784 de *Lecture Notes in Artificial Intelligence*, páginas 272–286. Springer-Verlag, 1994.

[Sta95] I. Stahl. The appropriateness of predicate invention as bias shift operation in ILP. *Machine Learning*, 20(1/2):95–118, 1995.

[Sti90] M. E. Stickel. Rationale and methods for abductive reasoning in natural language interpretation. En Rudi Studer, editor, *Natural Language and Logic*, Lecture Notes in Artificial Intelligence, páginas 233–252. Springer-Verlag, 1990.

[SW94] I. Stahl y I. Weber. The arguments of newly invented predicates in ILP. En S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volumen 237 de *GMD-Studien*, páginas 233–246. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[TMS01] M. Turcotte, S. H. Muggleton, y M. J. E. Sternberg. The effect of relational background knowledge on learning of protein three-dimensional fold signatures. *Machine Learning*, 43(1/2):81–95, 2001.

[UM82] P. E. Utgoff y T. M. Mitchell. Acquisition of appropriate bias for concept learning. En *Proceedings of the 2nd National Conference On Artificial Intelligence*, páginas 414–418. Morgan Kaufmann, 1982.

[Val84] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[vdL92] P.R.J. van der Laag. A most general refinement operator for reduced sentences. Informe técnico EUR–FEW–CS–92–03, Department of Computer Science, Erasmus University, 1992.

[vdLNC98] P. R. J. van der Laag y S.-H. Nienhuys-Cheng. Completeness and properness of refinement operators in inductive logic programming. *Journal of Logic Programming*, 34(3):201–225, 1998.

[VK76] M. H. Van Emden y R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[Wil31] W.A. Wilson. On quasi–metric spaces. *Amer. J. Math*, 53:675–684, 1931.

[Win00] B. Windels. The scott approach structure: an extension of the scott topology for quantitative domain theory. *Acta Math. Hungar.*, 88(1-2):35–44, 2000.

UNIVERSIDAD DE SEVILLA
600103202

UNIVERSIDAD DE SEVILLA

Reunido el Tribunal [illegible] los abajo firmantes en el día de la fecha, para juzgar la Tesis Doctoral de D. MIGUEL ÁNGEL GUTIÉRREZ NARANDO titulada OPERADORES DE GENERALIZACIÓN PARA EL APRENDIZAJE CLAUSAL acordó otorgarle la calificación de SOBRESALIENTE CUM LAUDE POR UNANIMIDAD

Sevilla, 16 de SEPTIEMBRE 2002

El Vocal, El Vocal, El Vocal,

Fdo.: Luis de Ledesma Otamendi — Fdo.: Delia Balbontín Noval — Fdo.: Mario de J. Pérez Jiménez

El Presidente — El Secretario. — El Doctorado.

Fdo.: José Muñoz Pérez — Fdo.: Eugenio Roanes Lozano — Fdo.: Miguel A. Gutiérrez Naranjo