PROGRAMACIÓN LÓGICA Y CAMBIO DE TEORÍA *

José Pedro Úbeda Rives
Juan Manuel Lorente Tallada
Enric Casaban Moya
Departamento de Lógica y Filosofía de la Ciencia
Universidad de Valencia

En este trabajo analizamos el modelo *AGM* de cambio de Teoría desde la perspectiva del lenguaje PROLOG y presentamos algunas estrategias para realizar dichos cambios. El trabajo se divide en tres apartados. En el primero ofrecemos una panorámica de los modelos de cambio de teoría. En el segundo establecemos los presupuestos básicos para aplicar dichos modelos sobre bases de conocimientos y en el tercero describimos una estrategia para el análisis de dichos cambios. El instrumento formal sobre el que se materializan estas ideas es el lenguaje PROLOG.

I. Modelos de Cambio de Teoría

El modelo ya clásico de cambio de teoría sobre el que se continúan haciendo modificaciones y generalizaciones es el *AGM* (Alchourrón-Gärdenfors-Makinson [85]). En este modelo, se considera que una teoría, *T,* es un conjunto de enunciados cerrado bajo un operador de consecuencia *Cn* ($T = Cn(T)$). En general, una teoría *T* se define a partir de un conjunto de enunciados *b*, denominado base de *T*, de forma que $T = Cn(b)$. Además, usualmente *Cn* cumple las propiedades de los operadores de consecuencia de Tarski:

$$X \subseteq Cn(X)$$
$$Cn(X) = Cn(Cn(X))$$
$$X \subseteq Y \rightarrow Cn(X) \subseteq Cn(Y)$$

En *AGM* los cambios son representados por operaciones de cambio *mínimo* sobre un conjunto de enunciados. Estas operaciones son la *expansión*, la *contracción* y la *revisión*.

* La investigación presentada en este trabajo se ha realizado en parte con cargo al proyecto de la DGICVT PB96-0764.

a) *Expansión*: se incrementa la teoría T con un enunciado a consistente con T, obteniéndose la teoría T', de modo que

$$T' = Cn\,(T \cup \{a\})$$

En este caso, si el operador de consecuencia Cn es de Tarski y b es una base de T, se cumple

$$T' = Cn\,(b \cup \{a\})$$

Así, es indiferente realizar la expansión sobre la base o sobre la teoría.

b) *Contracción*: un enunciado x perteneciente a la teoría T es rechazado, obteniéndose una teoría que no implica a x. En el modelo *AGM* la forma básica de contracción, denominada contracción de *intersección parcial*, se basa en *funciones de elección* aplicadas sobre subconjuntos máximamente consistentes.

Sea $K \perp p$ el conjunto de subconjuntos máximos de K tales que no impliquen a p. Una *función de elección* γ para K es una función tal que para todo p se tiene:

$$\begin{array}{ll} \varnothing \neq \gamma(K \perp p) \subseteq K \perp p & \text{si } K \perp p \neq \varnothing \\ \gamma(K \perp p) = \{K\} & \text{si } K \perp p = \varnothing \end{array}$$

Entonces, una operación $\div$ es una *contracción de intersección parcial* si existe una función de elección γ para K tal que para todo p se tiene

$$K \div p = \cap\gamma(K \perp p)$$

Dos casos extremos pueden destacarse: las *funciones de elección máxima*, que eligen un único elemento y la *identidad* que eligen todos. En el primer caso se elimina lo menos posible pero se puede demostrar que si K es una teoría y se hace esta revisión, se obtiene una teoría completa, sea K completa o no (en cierto sentido se elimina demasiado poco); en el segundo caso se elimina demasiado. (Véase D. Makinson [85]).

En Alchourrón-Gärdenfors-Makinson [85] se demuestra que toda contracción de intersección parcial cumple los siguientes postulados de Gärdenfors (el operador de consecuencia es el de Tarski, e incluye la implicación tautológica clásica (exigida para 6÷), es compacto (exigida para 4÷) y además satisface la introducción de la disyunción):

(1÷) $K \div p$ es una teoría si K lo es (cierre).
(2÷) $K \div p \subseteq K$ (inclusión).
(3÷) Si $p \notin Cn(K)$, entonces $K \div p = K$ (vacuidad).
(4÷) Si $p \notin Cn(\varnothing)$, entonces $p \notin Cn(K \div p)$ (éxito).
(5÷) Si $Cn(p) = Cn(q)$, entonces $K \div p = K \div q$ (preservación).
(6÷) $K \subseteq Cn((K \div p) \cup \{p\})$, si K es una teoría (recubrimiento).

También se demuestra que, si K es una teoría, entonces toda operación $\div$ que cumpla los postulados de Gärdenfors es una contracción de intersección parcial sobre K (Teorema de representación).

Además, estudian clases especiales de contracción, al exigir que las funciones de elección tengan ciertas propiedades. Así, se define una función de elección sobre K como *relacional* si y sólo si existe una relación R sobre el conjunto potencia de K tal que para todo $x \notin Cn(\varnothing)$,

$$\gamma(K \perp x) = \{B \in K \perp x : B' \, R \, B \text{ para toda } B' \in K \perp x\}$$

y *transitivamente relacional* si R es transitiva. Para estas clases de contracción se establece además un teorema de representación, mejorado por Rott [93]. Para otros teoremas de representación que no exigen la condición de que K sea una teoría, véase Hansson [93].

Hay otras formas de contracción que se ha demostrado que son contracciones de intersección parcial; se basan en 'ordenaciones' sobre el conjunto de enunciados o entre subconjuntos de enunciados (Alchourron & Makinson [82]), dando lugar a las contracciones *seguras* (*safe*).

c) *Revisión o actualización*: Dada una teoría T y un enunciado x inconsistente con T, se obtiene una teoría $T + x$ que implica x, es consistente y elimina lo "menos" posible de T. Este tipo de cambio puede reducirse a los casos a) y b) anteriores en virtud de la identidad de Levi:

$$T + x = [Cn((T \div \neg x) \cup \{x\}]$$

Las generalizaciones del modelo *AGM* analizan estas operaciones cuando se aplican a conjuntos de enunciados, y no a un único enunciado como lo hace el modelo *AGM*. Además analizan las contracciones y revisiones iterativas. Por último, consideran lo que sucede cuando el operador de consecuencia no cumple las condiciones citadas anteriormente.

II. PROLOG: CONCEPTOS BÁSICOS

El estudio de cambios de teorías escritas en PROLOG nos ha planteado una serie de cuestiones acerca del PROLOG. La primera es definir el operador de consecuencia. Para ello, necesitamos establecer la gramática del PROLOG:

Def. de Literal^{PRO}:
- Una constante es un literal^{PRO}.
- Una constante, seguida de "(", seguido de n ($n \geq 1$) literales^{PRO} o variables separadas por comas, seguido de ")" es un *literal*^{PRO}.

Def. de fórmula:
- Si L es un literal^{PRO}, L y (L) son fórmulas.
- Si A y B son fórmulas $A \, , \, B$ es una fórmula, $A \, ; \, B$ es una fórmula y *not(A)* es una fórmula.

[conjunción, disyunción y negación de fórmulas, son fórmulas]

Def. de fbf^{PRO} (*cláusula*):
- Un literal^{PRO} es fbf^{PRO}.
- Si A es un literal^{PRO} y B es una fórmula, A :- B es una fbf^{PRO}.

[léase "*A si B*", o $B \rightarrow A$]

Def. de Programa-PROLOG:
- Si *A* es una fbf^{PRO}, *A* es un $programa^{PRO}$.
- Si *A* y *B* son $programas^{PRO}$, *A* . *B* es un $programa^{PRO}$.

[La conjunción de programas, es un programa. Puede comprobarse que han sido definidos dos operadores de conjunción: "," conjunta fórmulas mientras que "." conjunta programas. Recalcar también que diferenciamos claramente entre fórmula y fbf^{PRO}.]

El concepto de consecuencia en PROLOG puede definirse de varias formas:

i) Una formula *P* es *consecuencia* de una base o programa, si el programa contesta *yes* ante la pregunta

?P.

Esta caracterización del concepto de *consecuencia* no es muy satisfactoria, ya que se nos presenta el resultado como surgiendo de una *caja negra* y, además, elimina la posibilidad de que una cláusula sea consecuencia de una base, ya que una cláusula no puede estar en el lugar de *P*. Sin embargo, parece evidente que la cláusula

a:-c

es una consecuencia de la base:

a:-b.
b:-c. (β)

ii) Una definición *semántica* de consecuencia tiene que tener en cuenta la forma de las fórmulas y fbf^{PRO}. Uno de los problemas con este tratamiento se plantea en la definición de consecuencia para el caso de una cláusula. No podemos considerar como ocurre en las definiciones semánticas usuales en lógica clásica que '*a:-b*' sea equivalente a '*not(b);a*', ya que en ese caso debería considerarse que la cláusula

e:-f

es una consecuencia de la base (β), lo que no parece correcto. Esto puede solucionarse estableciendo que la cláusula *P:-Q* es consecuencia de una base, si *añadiendo Q* a la base, entonces *P* es una consecuencia de la nueva base. (Su caracterización en PROLOG exigiría usar el predicado *assert* aunque sea de forma indirecta).

iii) Para una definición que supere los problemas señalados establecemos tres conceptos de consecuencia estrechamente relacionados: *consecuencia* de una base, *consecuencia* de una base y unas premisas y *consecuencia simultánea* de una base y un conjunto de conjuntos de premisas.

```
/* CONSECUENCIA EN PROLOG */

/* CONSECUENCIA de una base */
consecuencia(X) :- cn(X,[]).

/* CONSECUENCIA de un conjunto de premisas L */
cn(true,L). /* true es consecuencia de cualquier cosa */
```

```
cn(P,L) :- pert(P,L).
cn(P,L) :- P=..[‘,’,P1,P2], cn(P1,L), cn(P2,L).
cn(P,L) :- P=..[‘;’,P1,P2], (cn(P1,L); cn(P2,L)).
cn(P,L) :- P=..[not,P1], not(cn(P1,L)).
cn(P,L) :- clause(P,Q), cn(Q,L).
cn(P,L) :- P1=..[(‘:-’),P,Q], pert(P1,L), cn(Q,L).
cn(P,L) :- P=..[(‘:-’),P1,P2], fnd_dcha(P2,A), incrementa(L,A,L1),cnmul(P1,L1).

/* CONSECUENCIA simultánea de una lista de conjuntos de premisas*/
cnmul(P,[L|[]]) :- cn(P,L).
cnmul(P,[L|L1]) :- cn(P,L), cnmul(P,L1).

/* pertenencia a una lista */
pert(X,[X|L]).
pert(X,[L|L1]) :- pert(X,L1).

/* halla la forma normal disyuntiva asociada por la derecha */
fnd_dcha(A,B) :- fnd(A,C), enlista(C,L), deslista(L,B).

/* halla la forma normal disyuntiva de una expresión con not, y, o*/
fnd(A,B) :- int_not(A,C), disyuntiva(C,B).

/* transforma una expresión en otra con el not afectando a literales */
int_not(A,B) :- A=..[‘,’,A1,A2], int_not(A1,B1), int_not(A2,B2),B =.. [‘,’,B1,B2].
int_not(A,B) :- A=..[‘;’,A1,A2], int_not(A1,B1), int_not(A2,B2),B =.. [‘;’,B1,B2].
int_not(A,B) :- A=..[not,A1], A1 =.. [not,A2], int_not(A2,B).
int_not(A,B) :- A=..[not,D], D=..[‘,’,A1,A2], B1 =..[not,A1],B2=..[not,A2],
      C=..[‘;’,B1,B2], int_not(C,B).
int_not(A,B) :- A=..[not,D], D=..[‘;’,A1,A2], B1 =..[not,A1], B2=..[not,A2],
      C=..[‘,’,B1,B2], int_not(C,B).
int_not(A,A).

/* si A solo tiene la negación con literales, la pone en forma disyuntiva */
disyuntiva(A,B) :- A=..[‘;’,A1,A2], disyuntiva(A1,B1), disyuntiva(A2,B2),
      B=..[‘;’,B1,B2].
disyuntiva(A,B) :- A=..[‘,’,A1,A2], A1=..[‘;’,B1,B2], C1=..[‘,’,B1,A2],
      C2=..[‘,’,B2,A2], disyuntiva(C1,D1), disyuntiva(C2,D2), B=..[‘;’,D1,D2].
disyuntiva(A,B) :- A=..[‘,’,A1,A2], A2=..[‘;’,B1,B2], C1=..[‘,’,A1,B1],
      C2=..[‘,’,A1,B2], disyuntiva(C1,D1), disyuntiva(C2,D2), B=..[‘;’,D1,D2].
disyuntiva(A,B) :- A =..[‘,’,A1,A2], disyuntiva(A1,A3), disyuntiva(A2,A4),
      C=..[‘,’,A3,A4], not(A=C), disyuntiva(C,B).
disyuntiva(A,A).

/* transforma en lista los elementos de una disyunción */
enlista(A,L) :- A=..[‘;’,A1,A2], enlista(A1,L1), enlista(A2,L2), union(L1,L2,L).
enlista(A,[A]).

/* union de dos listas */
union([],L,L).
union(L,[],L).
union([L|L1],L2,L3) :- pert(L,L2), union(L1,L2,L3).
union([L|L1],L2,[L|L3]) :- union(L1,L2,L3).

/* transforma una lista en una disyunción asociada por la derecha */
deslista([L|[]],L).
deslista([L|L1],A) :- deslista(L1,A1), A =..[‘;’,L,A1].
```

```
/* si A es disyuncion crea tantos conjuntos de premisas como miembros en A */
incrementa(L,A,[L1|L2]) :- A=..[';',A1,A2], aumenta(L,A1,L1),incrementa(L,A2,L2).
 incrementa(L,A,[L1]) :- aumenta(L,A,L1).

/*aumenta una lista de premisas con los elementos de una conjunción A */
 aumenta(L,A,L1) :- conlista(A,L2), union(L,L2,L1).

/*transforma en lista los elementos de una conjunción*/
 conlista(A,L) :- A=..[',',A1,A2], conlista(A1,L1), conlista(A2,L2), union(L1,L2,L).
 conlista(A,[A]).
```

Otra cuestión que se plantea es la definición de *consistencia*. En las bases de conocimientos en PROLOG no puede aparecer la contradicción y así todas las bases son consistentes. Las cláusulas en PROLOG son del tipo Horn (tienen, a lo sumo, un literal afirmado). Para introducir hechos "negativos" que permitan el tratamiento de la "contradicción" debe recurrirse a "estratagemas", tal como el uso del operador *neg* de la máquina inferencial *CNV* (Covington, Nute & Vellino [88] o Sacks [90]), la que se trata en Lorente & Úbeda [93] o el uso del operador *no* (Lorente & Úbeda [97]). Así, en el primer caso, *neg p(X,Y)* corresponde a la negación de *p(X,Y)*; en el segundo, *p(si,X,Y)* tiene como negación la fórmula *p(no,X,Y)* y en el tercero *no(p(X,Y))* es la negación de *p(X,Y)*. Por todo ello establecemos un predicado *contradictorio* en PROLOG, a través de las siguientes reglas:

```
contradictorio(P,Q) :- Q =..[neg,P].
contradictorio(P,Q) :- P =..[neg,Q].
contradictorio(P,Q) :- P =..[P1|[si|L]], Q =.. [P1|[no|L]].
contradictorio(P,Q) :- P =..[P1|[no|L]], Q =.. [P1|[si|L]].
contradictorio(P,Q) :- Q =..[no,P].
contradictorio(P,Q) :- P =..[no,Q].
```

que permite usar todas las estratagemas mencionadas y, entonces es posible postular que una base es consistente si no existe ningún literal *P* tal que *P* y su contradictorio sean consecuencias de la base.

En Lorente & Úbeda [93] se presenta un algoritmo que detecta si una base de conocimientos es contradictoria e indica los predicados contradictorios, así como los argumentos para los que se da la contradicción. Además se desarrolla un algoritmo para que, cuando *p* sea contradictorio respecto a los argumentos $a_1,...,a_n$, produzca como output dos listas: una que indica a partir de qué reglas y hechos y con qué unificaciones se deduce $p(si, a_1, ...,a_n)$ y otra que establece lo mismo para $p(no, a_1, ..., a_n)$. Estos algoritmos son fácilmente trasladables al caso de las otras estratagemas.

En el modelo de cambio de teoría es esencial encontrar subconjuntos máximos que no impliquen un enunciado. Es decir, calcular

$$K \perp p$$

siendo *K* un conjunto de enunciados y *p* un enunciado.

Desde el punto de vista del PROLOG, *K* es un conjunto de cláusulas y *p* es una fórmula o una cláusula y existen varias formas de especificar *K*:

a) $K = Cn(B)$, donde *B* es una base. En ese caso *K* es infinito y es imposible describir en general y efectivamente los subconjuntos máximos de *K* tales que *p* no sea consecuencia de ellos.

b) *K* es una base. Comprobar que cada uno de los subconjuntos de *K* no implican a *p* es una tarea costosa (exponencial en el cardinal de *K*). Un procedimiento más efectivo consiste en realizar todas las derivaciones de *p* a partir de *K* (en cada derivación se usa un conjunto de cláusulas de *K*). Si se elimina una sola cláusula de cada derivación, se obtienen subconjuntos máximos de *K* que no implican a *p*. Para realizar esto se define el predicado *camino*, que permite saber a partir de qué cláusulas se obtiene *p*:

```
camino(P,[[P]]) :- clause(P, true).
camino(P,[[]]) :- P=..[not,Q], not(Q).
camino(P,[[L|L1]]) :- P=..[',',Q,R], camino(Q,L), camino(R,L1).
camino(P,[[P|Q]|L1]) :- clause(P,Q), camino(Q,L1).
```

En caso de que *p* sea una cláusula, habrá que modificar el predicado utilizando el concepto de consecuencia definido anteriormente.

c) *K* se obtiene a partir de una base *B* añadiendo a *B* sus consecuencias, bajo las siguientes condiciones: que éstas se formen con las mismas constantes (predicados y hechos) que aparecen en *B*, y que no sean cláusulas tautológicas tales como '*p:-p*' y tampoco ninguna de sus *subcláusulas* (*P* es una subcláusula de *Q* si ambas tienen la misma cabeza y si el conjunto de antecedentes de *P* es un subconjunto de los antecedentes de *Q*). (Véase Büning [87] y Schätz [88]).

III. Estrategias para la revisión de bases contradictorias

Realizar una expansión en una base de conocimientos de PROLOG es muy sencillo: se inserta el enunciado en la base utilizando el predicado *insert*. La contracción y revisión, por otra parte, exigen calcular $K \perp p$, lo que hemos visto que es computacionalmente costoso. Por ello los cambios de teoría que realizamos en PROLOG se basan en 'ordenaciones' de cláusulas (dichas ordenaciones pueden basarse en su grado de certeza, de fiabilidad, de aceptabilidad, etc.), que permiten definir funciones de elección y, de este modo, también contracciones. El procedimiento consiste en elegir los subconjuntos máximos a los que se halla eliminado las cláusulas minimales relativas a dicha ordenación.

De este modo, si lo que interesa es saber si un enunciado pertenece a la contracción o revisión de una teoría, sólo se precisa ver si es deducible de la base después de eliminar las cláusulas minimales.

En el caso de bases de conocimientos contradictorias, que es el más interesante y con mayor repercusión práctica, hemos desarrollado la siguiente estrategia: establecer unas "meta-reglas" que modifican automáticamente dicha base.

El funcionamiento o estructura de las "meta-reglas" es el siguiente:

i) Se observa si existen contradicciones en la base (para lo que se usa el algoritmo establecido en Lorente & Úbeda [93]).

ii) Si no hay contradicciones, la meta-regla no hace nada más. En caso contrario, establece para cada par de enunciados contradictorios las cláusulas que permiten su derivación (se utiliza el predicado *camino*).

iii) De acuerdo con la 'ordenación' de las cláusulas, se seleccionan las cláusulas minimales de entre las relacionadas en el punto ii).

iv) Una vez detectadas las cláusulas minimales, se modifica automáticamente la base. Estas modificaciones son de tres tipos y es el usuario el que las elige:

1) Cambiar el operador *:-* del PROLOG en "ciertas" cláusulas por el operador *:=* de Sacks (que corresponde a reglas que pueden tener excepciones). En este caso, las meta-reglas indican cuáles son dichas cláusulas (las minimales en la 'ordenación' de cláusulas).

2) Elegir uno de los términos contradictorios como deducible. Dicha elección se realiza después de haber establecido las derivaciones de cada uno de ellos y haber observado cual es la derivación en la que se usa la cláusula o las cláusulas minimales.

3) Cambiar la base de datos *eliminando* cláusulas, primitivas o derivadas, de forma que en la nueva base no pueda deducirse una contradicción (se eliminan las cláusulas minimales que se usan en la derivación de una contradicción).

Las 'ordenaciones' que se usan en las meta-reglas y, que pueden ser elegidas por el usuario son:

1) La relación de *dependencia*: a partir del retículo de dependencia de los predicados de una base (se usa un algoritmo que obtiene dicho retículo y que se describe en Úbeda & Lorente [93]) se realiza una ordenación de cláusulas, de modo que una cláusula es menor que otra si el predicado de la cabeza de la primera está en un nivel menor del retículo que el predicado de la segunda. Intuitivamente, una cláusula es menor en esta ordenación si es menos 'general'; se da preferencia a las leyes sobre los hechos.

2) La relación de *dominación*. Se obtiene de forma análoga a la de dependencia, pero invirtiéndose el orden. Intuitivamente, una regla es menor en esta ordenación si es más 'general'; se da preferencia a los hechos sobre las leyes.

3) Relación arbitraria. En este caso, el usuario introduce una relación entre las cláusulas por medio de una lista de pares.

Conclusiones

En línea con otros autores, por ejemplo Nute [1988], usamos el lenguaje PROLOG para analizar conceptos y teorías. El análisis de conceptos se realiza estableciendo los predicados que los definen, de tal modo, que éstos permiten tanto la representación como la experimentación y dando además lugar con ello a la posibilidad de estudio de la complejidad conceptual. Esto ha ocurrido, en nuestro caso, con el concepto de consecuencia.

El análisis de teorías, en este caso el de cambio de teoría, se realiza al modelizarla sobre las bases de conocimientos tal como se representan en PROLOG (i.e., PROLOG se utiliza como un laboratorio en el que se experimentan las teorías). Dicho análisis puede permitir conocer tanto las dificultades de aplicación de una teoría como las posibles deficiencias de las representaciones usuales en PROLOG, dando lugar a nuevas pistas formales para la representación del conocimiento.

Finalmente, se ha presentado una forma de utilización de bases contradictorias, ya que este fenómeno ocurre muy frecuentemente al actualizar bases de conocimientos.

Bibliografía

Alchourrón, C. E. & Makinson, D., 'Hierarchies Of Regulations And Their Logic', en R. Hilpenin (ed): *New Studies in Deontic Logic*, Reidel, 1982, pp. 125-148.

Alchourrón, C. E., Gärdenfors, P. & Makinson, D., 'On the logic of theory change: Partial meet functions for contraction and revision', *Journal of Symbolic Logic 50* (1985), pp. 510-530.

Covington, M., Nute, D. & Vellino, A., *Prolog Programming Techniques*, Scott-Foresman, Glenview, Illinois, 1988.

Hansson, S. O., 'Reversing the Levi Identity', *Journal of Philosophical Logic 22* (1993), pp. 637-669.

Lorente, J. M. & Úbeda, J. P., 'Detector of negation as failure in PROLOG', 3er Congreso Internacional de Ciencia Cognitiva, San Sebastian, 1993.

———, 'Tipos de negación en PROLOG', *Actas del II Congreso de la Sociedad de Lógica, Metodología y Filosofía de la Ciencia en España,* Barcelona, 1997, pp. 444-447.

Makinson, D., 'How to give it up: A survey of some formal aspects of the Logic of Theoy Change', *Synthese 62* (1985), pp. 347-363. 'Errata', *ibidem 63* (1986), pp. 185-186.

Nute, D., 'Defeasible reasoning: a philosophical analysis in PROLOG', en *Aspects of Artificial Intelligence,* ed. by J. H. Fetzer, Kluwe, 1988, pp. 251-288.

Rott, H., 'Belief contraction in the context of the general theory of rational choice', *Journal of Symbolic Logic 58* (1993), pp. 1426-1450.

Sacks, G. E., 'Prolog Programming', en Homer, Nerode, Platek, Sack & Scedrov (eds): *Logic and Computer Science*, Springer, 1990, pp. 90-110.

Úbeda, J. P. & Lorente, J. M., 'Circularidad en PROLOG', Primer Congreso de Lógica, Filosofía y Metodología de la Ciencia, Madrid, 1993.