Diferentes formas de hipótesis de mundo cerrado en la programación en lógica rebatible [1]

Alejandro J. García[2] Guillermo R. Simari

Grupo de Investigación en Inteligencia Artificial (GIIA)
Instituto de Ciencias e Ingenieria de la Computación
Departamento de Ciencias de la Computación
Universidad Nacional del Sur
Av.Alem 1253 - (8000) Bahía Blanca, ARGENTINA
FAX: (54) (91) 563401
e-mail: ccgarcia@criba.edu.ar grs@criba.edu.ar

Resumen

Los *programas lógicos rebatibles* (PLR) permiten utilizar dos tipos de negación: la *negación por falla finita* (*not*), y la *negación clásica* ($\neg$). De esta forma, es posible trabajar con información incompleta, y potencialmente inconsistente. El objetivo de este trabajo es mostrar como la expresividad de los PLR, al disponer de dos tipos de negación, permite representar la *hipótesis de mundo cerrado* (closed world assumption o CWA) directamente como cláusulas de programa. La CWA de un predicado particular p puede lograrse incluyendo en el PLR la cláusula "$\neg$p(X) $\leftarrow$ *not* p(X)", esto es, si no puedo probar p, entonces puedo derivar $\neg p$. La posibilidad de incluir la hipótesis de mundo cerrado dentro del mismo lenguaje, permite representar otras formas de CWA como "p(X) $\leftarrow$ *not* $\neg$p(X)", "p(X) $\leftarrow$ *not* p(X)" y "$\neg$p(X) $\leftarrow$ *not* $\neg$p(X)". Por otro lado, la programación en lógica rebatible incorpora las cláusulas rebatibles (CPR), denotadas "$l \prec A$", que deben leerse como: "razones para creer en A son buenas razones para creer en l". Por lo tanto, los PLR permiten representar nuevas formas de CWA mediante la utilización de una cláusula de programa rebatible: "$\neg$p(X) $\prec$ *not* p(X)", es decir, no poder probar "p(X)" es una buena razón (rebatible) para asumir "$\neg$p(X)". Este nuevo tipo de cláusulas de CWA rebatibles soluciona problemas que pueden plantearse al utilizar sólo cláusulas estrictas (no rebatibles).

[1]Financiado parcialmente por la Secretaría de Ciencia y Técnica, Universidad Nacional del Sur.

[2]Becario del Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), República Argentina

1. Introducción

La programación en lógica tradicional ha sido utilizada ampliamente como una herramienta de representación de conocimiento. Lamentablemente presenta limitaciones para manejar ciertas características del razonamiento del sentido común, como utilizar información incompleta, o potencialmente inconsistente. Los trabajos de Gelfond y Lifschitz [10], Inoue [11], Alferes y Pereira [1], Dung [2, 3, 4], y Fillottrani [5], establecen nuevos criterios para extender la programación en lógica introduciendo la negación clásica, y manteniendo la negación por falla.

La *programación en lógica rebatible* [16, 6] fue introducida como una extensión de la programación en lógica, y también permite representar ambos tipos de negación. Pero fundamentalmente permite disponer de un lenguaje de representación que captura aspectos del razonamiento del sentido común que son difíciles de expresar en los trabajos antes mencionados.

La semántica de la negación en la programación en lógica tradicional es la *negación por falla finita*, habitualmente representada por el símbolo *"not"*. La consulta de una meta *"not p"* tiene éxito cuando existe una falla finita de la prueba del predicado p. Este tipo de semántica puede ser útil para algunos predicados en algunos contextos, como en el caso de la cláusula "buscar(X) $\leftarrow$ *not* muerto(X),perdido(X)", (esto es, si 'X' está perdido y no puedo probar que esté muerto, sigo buscándolo). No obstante existen numerosas ocasiones donde la utilización de la negación por falla lleva a resultados no deseables. Por ejemplo si se quiere representar la regla "aterrizar cuando la pista no está ocupada" con la cláusula "aterrizar $\leftarrow$ *not* pista-ocupada", la vida de los pasajeros correrá mucho peligro cuando no se posea información sobre la pista. Por lo tanto, parecería conveniente que la negación por falla sea utilizada sólo en algunos casos y la negación clásica en otros.

Los *programas lógicos rebatibles* (PLR) permiten utilizar los dos tipos de negación: la *negación por falla finita* (*not*), presente en la mayoría de los lenguajes de programación en lógica, y que permite trabajar con información incompleta, y la *negación clásica* ($\neg$) que resulta imprescindible para representar conocimiento y trabajar con información potencialmente inconsistente. Utilizando la negación clásica ahora es posible escribir la regla anterior como "aterrizar $\leftarrow$ $\neg$pista-ocupada", que representa el significado deseado, esto es, "aterrizar si puedo probar que la pista no está ocupada".

En un programa lógico tradicional, la respuesta a una consulta es '*si*' o '*no*', ya que automáticamente se aplica la *hipótesis de mundo cerrado (closed world assumption o CWA)*, asumiendo como falso a todo átomo instanciado que no pueda derivarse del programa. Esto resulta demasiado restrictivo y como en el caso de la negación por falla, puede llevar a situaciones no deseables. Como se verá a continuación, en la programacion en lógica rebatible hay cuatro respuestas posibles para una consulta: *si, no, indeciso* y *desconocido.*

La respuesta a un átomo que no puede derivarse del programa será '*desconocido*', en lugar de '*no*', y por lo tanto no se aplicará CWA a ningún predicado.

El objetivo de este trabajo es mostrar como la expresividad de los PLR, al disponer de dos tipos de negación, permite representar la *hipótesis de mundo cerrado* directamente como cláusulas de programa, y además crear nuevas formas de CWA. La CWA *clásica* de un predicado particular p se logra incluyendo en el PLR la cláusula "$\neg$p(X) $\leftarrow$ *not* p(X)", esto es, si no puedo probar p, entonces puedo derivar $\neg p$. Al poder definir la hipótesis de mundo cerrado dentro del mismo lenguaje, entonces es posible representar otras formas de CWA como "p(X) $\leftarrow$ *not* $\neg$p(X)", "p(X) $\leftarrow$ *not* p(X)" y "$\neg$p(X) $\leftarrow$ *not* $\neg$p(X)". Por ejemplo puede escribirse la regla de CWA "inocente(X) $\leftarrow$ *not* inocente(X)".

Pero la característica más importante que incorpora la programación en lógica rebatible, son las *cláusulas rebatibles* (CPR) denotadas "$l \prec A$" (ver definición 2.2), las cuales deben leerse como: "razones para creer en el antecedente A son buenas razones para creer en el consecuente l". Por lo tanto, los PLR permiten representar un nuevo tipo de cláusulas de CWA mediante la utilización de una CPR: "$\neg$p(X) $\prec$ *not* p(X)", es decir, no poder probar "p(X)" es una buena razón (rebatible) para asumir "$\neg$p(X)". Como se verá en la sección 4, este nuevo tipo de cláusulas de CWA rebatibles soluciona problemas que pueden plantearse al utilizar sólo cláusulas estrictas (no rebatibles).

2. La programación en lógica rebatible

En esta sección se describirá brevemente la sintaxis y semántica de los PLR y cómo se realiza una inferencia dentro de la programación en lógica rebatible. Al final de este trabajo se presenta un apéndice donde se amplían estos conceptos (más detalles pueden encontrarse en [16] y [6]).

En el lenguaje de la programación en lógica rebatible, un *literal* "l" es un átomo "a" o un átomo negado "$\neg a$", siguiendo la definición de Lloyd [12]. El símbolo "$\neg$" representa la negación clásica, y el símbolo "*not*" la negación por falla (el significado y uso de estos dos símbolos está explicado en detalle en la sección 3). El símbolo "$\sim$" en cambio, se utiliza para indicar el complemento de un literal con respecto a la negación clásica, *i.e.*, $\sim l = \neg l$, y $\sim\neg l = l$. Dos literales son complementarios, si uno es el complemento del otro, y la complementariedad está definida siempre sobre "$\neg$".

Definición 2.1 *Cláusula de programa extendido* (CPE) [10]: Es una cláusula de programa de la forma "$l \leftarrow p_1, \ldots, p_n$", $(n \geq 0)$ donde l es un literal, y cada p_i es un literal o un literal precedido por el símbolo *not* de la negación por falla. Si n=0, entonces se denotará "$l \leftarrow$ TRUE", y se dirá que l es un *hecho* (donde TRUE tiene la interpretación usual). □

Obsérvese que de acuerdo a la sintaxis definida "¬p ← *not* ¬q" es una CPE válida mientras que "*not* p ← q" , "p ← ¬¬p", "p ← *not not* p", y "p ← ¬*not* q" no lo son.

Las *cláusulas de programa extendido* fueron introducidas por Gelfond y Lifchitz en [10], donde además se define una semántica para los *programas extendidos*, basada en conjuntos de respuestas. En su propuesta, un programa extendido se dice *contradictorio* cuando el conjunto de respuestas tiene un par de literales complementarios, como es el caso del programa { p ← TRUE ; ¬p ← TRUE }. La definición de conjunto de respuestas establece que "si contiene un par de literales complementarios, entonces el conjuto de respuestas es *Lit*, el conjunto de todos los literales instanciados del lenguaje del programa". Por lo tanto (ver [10] proposición 1), "todo programa contradictorio tiene exactamente un conjunto de respuestas: el conjunto *Lit*." Al representar conocimiento, es muy común que se tengan reglas que permitan derivar literales complementarios, y esto lleva directamente a programas contradictorios. Inoue en [11] intenta solucionar esta limitación, pero no define un criterio de preferencia para decidir en el caso que se deriven dos literales complementarios.

La programación en lógica rebatible soluciona ambos problemas: es posible representar información potencialmente inconsistente sin que se derive todo el lenguaje, y existe un criterio para decidir entre conclusiones complementarias. Para esto se introduce un nuevo tipo de cláusulas de programa:

Definición 2.2 *Cláusula de programa rebatible* (CPR) [16]: Es una cláusula de la forma "$l \prec p_1, \ldots, p_n$" , ($n \geq 0$) donde l es un literal, y cada p_i es un literal o un literal precedido por el simbolo "*not*" de la negación por falla. Si n=0 se denotará "$l \prec$ TRUE", y se dira que l es una *presuposición*. El símbolo "$\prec$ " se utiliza para distinguir una CPR de una CPE, porque una CPR se utilizará para representar conocimiento rebatible, *i.e.*, información tentativa que puede ser usada en la medida que no sea contradecida. Una cláusula "$l \prec A$" debe leerse como: "razones para creer en el antecedente A son buenas razones para creer en el consecuente l". □

Un *programa lógico rebatible* (PLR), es un conjunto finito de CPE y CPR (ver ejemplo 2.1). Una *meta definida* es simplemente un literal m. Una *prueba rebatible* para una meta definida m (ver apéndice definición 6.1) es el conjunto de cláusulas de programa instanciadas (CPE y CPR) que permiten derivar m. Un conjunto de cláusulas es *consistente* (resp. *inconsistente*) si no es posible (resp. es posible) probar rebatiblemente un par de literales complementarios. Dado un PLR, el conjunto de CPE debe ser consistente, mientras que el conjunto de CPR y el propio PLR pueden ser *inconsistentes*. De esta forma, el lenguaje permite trabajar con información incompleta y potencialmente inconsistente. Como la noción de prueba rebatible no prohibe que puedan derivarse rebatiblemente dos literales complementarios, resulta necesario definir un criterio de inferencia, a fin de que sólo una de las metas complementarias sea aceptada como una nueva creencia.

Es en este punto donde el lenguaje utiliza los conceptos de la *argumentación rebatible* [8, 14, 15, 16]. En primer lugar se define la noción de *argumento* para una meta m como el subconjunto de CPR instanciadas utilizadas en la generación de una prueba rebatible para m (ver apéndice definición 6.2). La consulta de una meta m tendrá éxito, si m pertenece al *conjunto de respuestas positivas* del lenguaje, esto es, existe un argumento que es una justificación de m. El proceso de obtención de una justificación para m, involucra la construcción de un *argumento aceptable* para m. Para decidir sobre la aceptabilidad de un argumento, se construyen a partir del PLR, *contraargumentos* que son posibles *derrotadores* (ver apéndice definiciones 6.4 y 6.5). De igual forma, se verifica la aceptabilidad de los posibles derrotadores. Aquellos derrotadores aceptables son comparados con el argumento original usando un *criterio de preferencia.* Un argumento es una justificación, si es mejor que todos sus derrotadores aceptados.

La semántica de un PLR está caracterizada por cuatro conjuntos de repuestas (ver apéndice definición 6.9): (1) el *conjunto de respuestas positivas*, que representa las conclusiones positivas que pueden inferirse del programa, y está formado por los literales que tienen un argumento que es una justificación; (2) el *conjunto de respuestas negativas*, que representa la información negativa que puede derivar el programa, y está formado por los literales cuyos argumentos fueron derrotados; (3) el *conjunto de respuestas indecisas*, formado por literales que si bién tienen una prueba rebatible, no resulta posible decidir entre sus argumentos a favor y en contra (se trata de información que no es ni positiva, ni negativa); y por último (4) el *conjunto de respuestas desconocidas*, un conjunto posiblemente infinito de todos los literales para los cuales no existe una prueba rebatible. De lo anterior puede verse claramente que ante una consulta q ya no habrá sólo dos respuestas posibles como en la programación en lógica tradicional. Ahora las respuestas serán cuatro: *si*, *no*, *indeciso* y *desconocido* (ver definición 6.10).

A continuación se presenta un ejemplo de PLR, con los conjuntos de respuestas correspondientes. En las cláusulas de programa se utilizará la convención tipográfica habitual de PROLOG.

Ejemplo 2.1

ñandú(charo) $\leftarrow$ TRUE.
ave(X) $\leftarrow$ ñandú(X).
$\neg$vuela(X) $\prec$ ñandu(X).
vuela(X) $\prec$ ave(X).
anida_árbol(X) $\prec$ vuela(X).
anida_suelo(X) $\prec$ *not* anida_árbol(X).

El PLR anterior tiene los siguientes conjuntos de respuestas: positivas = { ñandú(charo), ave(charo), $\neg$vuela(charo), anida_suelo(charo) }, negativas = { vuela(charo), anida_árbol(charo) }, e indecisas = { }. □

3. Semántica de la negación en los PLR

Como se indicó anteriormente, en los PLR es posible utilizar dos tipos de negación: la clásica denotada con el símbolo "¬", y la negación por falla finita denotada con el símbolo "*not*". El objetivo de incluir la negación clásica es poder representar información potencialmente inconsistente, ya que tanto una meta de la forma "¬a", como una de la forma "a" pueden ser derivadas rebatiblemente de un PLR. Esto permite escribir cláusulas como "¬vuela(X) $-\prec$ pingüino(X)" y "vuela(X) $-\prec$ ave(X)".

La negación por falla, en cambio, se incluye para poder trabajar con información incompleta. Por ejemplo al escribir la cláusula "inocente(X) $-\prec$ *not* culpable(X)", se está indicando que "no poder probar la culpabilidad de X, es una buena razón para asumir que X es inocente". Es por ello que la utilización de la negación por falla se restringe sólo al cuerpo de una cláusula. La diferencia fundamental radica en que el significado buscado para "¬a" es "*existe una justificación para* ¬a", mientras que el de "*not* a" es "*no existe una justificación para* a". Es importante destacar que aunque las metas precedidas por "*not*" puedan probarse rebatiblemente, estas nunca forman parte de los conjuntos de respuestas, (como "*not* anida_árbol(charo)" en el ejemplo 2.1).

La consulta de una meta "¬a" tendrá éxito cuando "¬a" figure en el conjunto de respuestas positivas del PLR (*i.e.*, tenga una justificación), como ocurre con "¬vuela(charo)" en el ejemplo 2.1. Pero sin embargo, una submeta "*not* l" tendrá éxito, cuando "l" no figure en el conjunto de respuestas positivas, *i.e.*, cuando "l" no tenga éxito. En el ejemplo 2.1 "anida_árbol(charo)" no figura en el conjunto de respuestas positivas del PLR, y por lo tanto puede probarse rebatiblemente "*not* anida_árbol(charo)".

Es fácil de ver que la negación en este lenguaje cumple con el *criterio de coherencia* [1], esto es, *si la meta "¬l" tiene éxito, entonces "not l" también lo tendrá.* La demostración es sencilla: si "¬l" tiene éxito, entonces "¬l" pertenece al conjunto de respuestas positivas, es decir existe una justificación para "¬l", luego, no existirá una justificación para "l", y por lo tanto "*not* l" tendrá éxito. También es fácil ver que la recíproca no es cierta: que "*not* l" tenga éxito no implica que el literal "¬l" lo tenga. En el ejemplo anterior aunque pueda derivarse "*not* anida_árbol(charo)", no ocurre lo mismo con "¬anida_árbol(charo)".

Es importante destacar que los dos tipos de negación son importantes, y como se verá a continuación, no pueden definirse uno en función del otro. Una cláusula como "$p \leftarrow$ *not* q" indica que al no poder justificarse q, entonces puedo derivar p. Mientras que utilizando "$p \leftarrow \neg q$", se podrá derivar p cuando se pueda derivar $\neg q$. No obstante, en un PLR puede escribirse una presuposición "¬q $-\prec$ TRUE" para indicar que "hay razones (rebatibles) para creer en ¬q". De esta manera se podría pensar que la cláusula "p $\leftarrow$ *not* q" puede reemplazarse por el par { p $\leftarrow$ ¬q ; ¬q $-\prec$ TRUE }, obteniendose un resultado equivalente; lo cuál significaría que los PLR $\mathcal{P}_1$ y $\mathcal{P}_2$ tendrían los mismos conjuntos de respuestas.

$\mathcal{P}_1$	$\mathcal{P}_2$
$p \leftarrow not\ q$	$p \leftarrow \neg q$
	$\neg$q $\prec$ TRUE

Si lo anterior fuera cierto, se podrían obtener PLR equivalentes, sin la utilización del operador de negación por falla "*not*", simplemente haciendo los reemplazos correspondientes. Aunque a primera vista parece que el resultado fuera equivalente, hay varias diferencias. En primer lugar, si se usan las cláusulas de $\mathcal{P}_2$ como reemplazo de $\mathcal{P}_1$, se pierde el significado buscado con el uso del operador "*not*", ya que ahora p podrá derivarse cuando exista una justificación de $\neg q$. Pero además, los conjuntos de respuestas son diferentes. El conjunto de respuestas positivas de $\mathcal{P}_1$ es $\{p\}$, mientras que el de $\mathcal{P}_2$ es $\{p, \neg q\}$. Al agregar "$q \prec$ TRUE" a ambos PLR, se acentúan las diferencias, ya que el conjunto de respuestas positivas para $\mathcal{P}_1 \cup \{\ q \prec$ TRUE $\}$ es $\{q\}$, mientras que el de $\mathcal{P}_2 \cup \{\ q \prec$ TRUE $\}$ es $\emptyset$. Por otro lado resulta imposible agregar "$\neg p \leftarrow$ TRUE" a $\mathcal{P}_1$ porque el conjunto de CPE se vuelve inconsistente, mientras que sí es posible agregarlo a $\mathcal{P}_2$ y el conjunto de respuestas positivas para $\mathcal{P}_2 \cup \{\ \neg p \leftarrow$ TRUE $\}$ es $\{\neg p\ ,\ \neg q\}$. Por último, el conjunto de respuestas positivas para $\mathcal{P}_1 \cup \{\ \neg p \prec$ TRUE $\}$ es $\{p\}$, mientras que el de $\mathcal{P}_2 \cup \{\ \neg p \prec$ TRUE $\}$ es $\{\ \neg q\}$, ya que los literales p y $\neg p$ pasan a pertenecer al conjunto de respuestas indecisas.

Otra consideración importante es la siguiente: para que un conjunto de CPR sea un argumento (ver definición 6.2) se establece una condición de consistencia, que impide que dentro de un argumento existan dos literales complementarios con respecto a la negación clásica. Sin embargo con la negación por falla no ocurre lo mismo. Supóngase que se tiene el siguiente PLR: $\mathcal{P}$= { a $\prec$ b,c ; b $\prec$ p ; c $\prec$ *not* p ; p $\prec$ e ; $\neg$p $\prec$ d ; d $\leftarrow$ TRUE ; e $\leftarrow$ TRUE }. Aquí es posible construir el argumento $\mathcal{A}$= { a $\prec$ b,c ; b $\prec$ p ; p $\prec$ e ; c $\prec$ *not* p } para el literal "*a*". Obsérvese que en el mismo argumento se usaron como submetas a "p" y "*not* p". Esto es posible porque el literal "p" puede probarse rebatiblemente, pero no existe un justificación para "p", ya que es bloqueada por el argumento "{$\neg$p $\prec$ d }". La siguiente proposición muestra que aunque el argumento anterior pueda construirse, nunca podrá ser una justificación.

Proposición 3.1 *Un argumento $\mathcal{A}$ donde figuren las submetas "l" y "not l", no será nunca una justificación.* Demostración: (a) si "*l*" está justificado, entonces "*not l*" no podrá probarse, invalidando al argumento. (b) si "*not l*" está probado, entonces no existe una justificación para "*l*", esto es, existe un derrotador aceptable para todo argumento de l, y por lo tanto existirá un derrotador aceptable para $\mathcal{A}$. □

4. La CWA como cláusulas de programa

En un programa lógico tradicional, la respuesta a una consulta es '*sí*' o '*no*', ya que automáticamente se aplica la hipótesis de mundo cerrado (CWA), asumiendo como falso a

todo átomo instanciado que no pueda derivarse del programa. Esto resulta demasiado restrictivo y como en el caso de la negación por falla, puede llevar a situaciones no deseables. Por ejemplo, en el programa $\mathcal{P}= \emptyset$, al aplicar CWA, se asumen como falsos todos los átomos instanciados del lenguaje; y en el programa $\mathcal{P}= \{ p \leftarrow not\ q \}$, p y q se asumen falsos, por lo tanto puedo asumir como verdaderos a p, y a *not* p (observación: aquí "*not*" tiene el significado dado por la CWA, no el de la negación por falla).

Como se vió anteriormente, en la programación en lógica rebatible, hay cuatro respuestas posibles para una consulta: *si, no, indeciso* y *desconocido*. La respuesta a una consulta que no puede derivarse del programa es '*desconocido*', en lugar de '*no*', y por lo tanto no se aplica automáticamente CWA a ningún predicado.

No obstante, la hipótesis de mundo cerrado de un predicado particular p en un PLR, puede lograrse incluyendo en el PLR la cláusula "$\neg$p(X) $\leftarrow$ *not* p(X)" con lo cual se está indicando que: *si no se puede justificar p, entonces puede derivarse* $\neg p$; que es justamente lo que establece la hipótesis de mundo cerrado. De esta forma, es posible representar dentro del mismo lenguaje, algo que hasta ahora era una condición metalingüística. Por ejemplo en un PLR con la cláusula "$\neg$culpable(X) $\leftarrow$ *not* culpable(X)", si no se puede justificar que X sea culpable, se puede asumir que no lo es. Como las cláusulas de CWA se representan dentro del propio lenguaje, pueden representarse ahora otras formas de CWA:

p(X) $\leftarrow$ *not* $\neg$p(X)	peligroso(X) $\leftarrow$ *not* $\neg$peligroso(X)
p(X) $\leftarrow$ *not* p(X)	inocente(X) $\leftarrow$ *not* inocente(X)
$\neg$p(X) $\leftarrow$ *not* $\neg$p(X)	$\neg$culpable(X) $\leftarrow$ *not* $\neg$culpable(X)

Estas formas de CWA ya habían sido introducidas en [10], donde también se aprovechaba la facilidad de disponer de dos tipos de negación. No obstante, hasta el momento, todo el desarrollo sobre cláusulas de CWA se ha realizado utilizando cláusulas de programa extendido (*i.e.*, CPE). Como se verá a continuación esto puede traer algunos problemas. Considérese el siguiente PLR:

$\mathcal{P}_3$: pista-ocupada(X) $\leftarrow$ *not* pista-ocupada(X).
$\neg$pista-ocupada(norte) $\leftarrow$ TRUE.

En la propuesta de Gelfond y Lifschitz, un programa como $\mathcal{P}_3$ es "contradictorio", y por lo tanto a partir de él se puede inferir todo el lenguaje. Por otro lado, $\mathcal{P}_3$ tampoco puede ser un programa lógico rebatible, porque la definición de PLR establece que el conjunto de CPE debe ser consistente.

Afortunadamente, el lenguaje de programación en lógica rebatible dispone de dos tipos de cláusulas: las CPE, y las CPR. La diferencia entre utilizar una CPE o una CPR es importante, las primeras representan conocimiento seguro (estricto), mientras que las CPR representan información tentativa (rebatible). Los PLR, permitirán entonces, obtener un nuevo tipo de cláusula de CWA: las *cláusulas de CWA rebatibles*, escribiendo por ejemplo

"¬p(X) $-\prec$ *not* p(X)", esto es, " no poder justificar p(X) es una buena razón (rebatible) para derivar ¬p(X)". Al utilizar este nuevo tipo de CWA, se pueden eliminar los problemas de $\mathcal{P}_3$, escribiendo:

$\mathcal{P}_3'$: pista-ocupada(X) $-\prec$ *not* pista-ocupada(X).
¬pista-ocupada(norte) $\leftarrow$ TRUE.

En $\mathcal{P}_3'$ ahora sólo es posible inferir el literal "¬pista-ocupada(norte)", desapareciendo el problema de la inconsistencia. Obsérvese que aunque existe la derivación rebatible { pista-ocupada(norte) $-\prec$ *not* pista-ocupada(norte) } para el literal "pista-ocupada(norte)", ésta no es consistente con el conjunto de CPE, y por lo tanto no es un argumento.

En general para cualquier predicado P toda cláusula de la forma "$P \leftarrow$ *not* P" producirá un problema similar al del programa $\mathcal{P}_3$, ya que permite que el predicado P pueda derivarse rebatiblemente cuando no se pueda justificarlo. Esto hará que la inclusión de cualquier instancia de $\neg P$, produzca una inconsistencia. El mismo problema aparece con la cláusula "$\neg P \leftarrow$ *not* $\neg P$". Por lo tanto siempre será conveniente representar este tipo de reglas con cláusulas rebatibles.

Una situación distinta pero también problemática se presenta al utilizar una cláusula de CWA como "$\neg P \leftarrow$ *not* P". Considérese por ejemplo el siguiente PLR:

¬culpable(X) $\leftarrow$ *not* culpable(X). (*)
¬preso(X) $\leftarrow$ inocente(X).
inocente(X) $\leftarrow$ ¬culpable(X).
preso(X) $-\prec$ *not* culpable(X), alta-sospecha(X).
alta-sospecha(pepe) $\leftarrow$ TRUE.

En el PLR anterior aunque esté presente el hecho "alta-sospecha(pepe)", la meta "¬preso(pepe)" tiene una justificación, gracias a la cláusula (*). Contrario a las espectativas de que exista una justificación para "preso(pepe)", el conjunto { preso(pepe) $\leftarrow$ *not* culpable(pepe), alta-sospecha(pepe) } no es un argumento ya que es inconsistente con el conjunto de CPE.

Sin embargo, si la cláusula (*) es reemplazada por la cláusula rebatible r = "¬culpable(X) $-\prec$ *not* culpable(X) ", entonces es posible construir un argumento $\mathcal{A}$ para "¬preso(pepe)" y otro argumento $\mathcal{B}$ para "preso(pepe)".

$\mathcal{A}$= { ¬culpable(pepe) $-\prec$ *not* culpable(pepe) }
$\mathcal{B}$= { preso(pepe) $\leftarrow$ *not* culpable(pepe), alta-sospecha(pepe) }

Utilizando el criterio de especificidad, $\mathcal{B}$ es un derrotador propio de $\mathcal{A}$. Por lo tanto utilizando la CPR r, $\mathcal{B}$ se convierte en una justificación para "preso(pepe)", obteniéndose el resultado esperado.

Una de las ventajas más importantes de los PLR es su adaptabilidad al cambio. Al manejar adecuadamente la inconsistencia, permite que se agregue información en forma dinámica, sin presentar problemas. Por ejemplo, en el PLR $\mathcal{P}_4$, puede obtenerse tanto una justificicación para "pista-ocupada(norte)", como para "pista-ocupada(sur)". Sin embargo, si se agregan dos nuevos hechos (ver $\mathcal{P}_4'$), entonces ya no es posible justificar los literales anteriores. Esto se debe a que los argumentos que se habían construído antes, con el agregado de los nuevos hechos, no cumplen con la condición de consistencia de la definición 6.2. En el nuevo PLR $\mathcal{P}_4'$ ahora existe una justificación para " ¬pista-ocupada(norte)", "pista-libre(sur)" y "¬pista-ocupada(sur)".

$\mathcal{P}_4$:	$\mathcal{P}_4'$:
pista-ocupada(X) $\dashrightarrow$ *not* pista-ocupada(X)	pista-ocupada(X) $\dashrightarrow$ *not* pista-ocupada(X)
¬pista-ocupada(X) $\leftarrow$ pista-libre(X)	¬pista-ocupada(X) $\leftarrow$ pista-libre(X)
	¬pista-ocupada(norte) $\leftarrow$ TRUE
	pista-libre(sur) $\leftarrow$ TRUE

Observación sobre los ciclos: Al utilizar una regla como "¬a $\leftarrow$ *not* a" el mecanismo de derivación no incurre en un ciclo, ya que la definición de derivación rebatible lo prevee explicitamente. No obstante, al encontrar una submeta "*not l*" en una cláusula C se necesita saber si "*l*" tiene o no una justificación. A fin de no incurrir en un ciclo entre justificaciones, al intentar justificar "*l*", no debe permitirse que se utilice nuevamente la cláusula C que disparó el proceso. Por ejemplo en el PLR { a $\dashrightarrow$ *not* b ; b $\dashrightarrow$ *not* a }, no existe una justificación para "a", ni para "b".

5. Conclusiones y desarrollos futuros

La semántica de los PLR permite el manejo de información potencialmente inconsistente, ya que pueden ser derivados literales complementarios, y luego sólo uno de ellos es elegido como una inferencia válida. La expresividad del lenguaje de programación en lógica rebatible, y la disponibilidad de dos tipos de negación, posibilitan la representación de diferentes formas de CWA dentro del lenguaje. Esto permite el manejo de información incompleta en una forma adecuada. Según lo expuesto en la sección 4 las cláusulas de CWA rebatible han resultado más adecuadas que las utilizadas hasta el momento.

Actualmente se encuentra en desarrollo una máquina abstracta para los PLR, como una extensión de la maquina abstracta de Warren (WAM), y un interprete para los PLR, en base a la semántica definida por los conjuntos de respuestas.

6. Apéndice: programación en lógica rebatible

Las siguientes definiciones fueron extraidas de [16, 6, 14], y corresponden a los principales conceptos de la programación en lógica rebatible, y de la argumentación rebatible (por más detalles referirse a [15, 14, 8, 9]).

Definición 6.1 Dado un PLR $\mathcal{P}$, una *prueba rebatible* para una meta definida m a partir de $\mathcal{P}$, es un conjunto finito de cláusulas de programa instanciadas (CPE y CPR), definido recursivamente de la siguiente forma:

1. Si existe un hecho "$c \leftarrow$ TRUE" (presuposición "$c \prec$ TRUE") en $\mathcal{P}$, tal que m unifica con c, con unificador mas general σ), el conjunto $\{c\sigma \leftarrow \text{TRUE}\}$ ($\{c\sigma \prec \text{TRUE}\}$) es una prueba rebatible para m.
2. Si existe una CPE "$c \leftarrow L$" (CPR "$c \prec L$") tal que c unifica con m (con umg σ), y existe un prueba rebatible $\mathcal{F}_i$ para cada uno de los elementos de $L\sigma$, entonces, $\{c\sigma \leftarrow L\sigma\} \cup (\bigcup_i \mathcal{F}_i)$ ($\{c\sigma \prec L\sigma\} \cup (\bigcup_i \mathcal{F}_i)$) es una prueba rebatible para m. A fin de que no se produzcan ciclos, la cláusula "$c\sigma \leftarrow L\sigma$" ("$c\sigma \prec L\sigma$") no debe aparecer en ninguno de los conjuntos $\mathcal{F}_i$.
3. Si m es una submeta que tiene el operador de negación por falla *not* (*i.e.*, m es *not* l), y no existe una *justificación* para el literal l, entonces el conjunto vacio es una prueba rebatible para m.
4. Si no se da ninguno de los casos anteriores, entonces no existe una prueba rebatible para m.

Definición 6.2 Dado un PLR, formado por el conjunto $\mathcal{S}$ de CPE, y el conjunto $\mathcal{D}$ de CPR, un *argumento* $\mathcal{A}$ para un literal h, es un subconjunto de CPR instanciadas, tal que: (1) Existe una prueba rebatible de m a partir de $\mathcal{S} \cup \mathcal{A}$ (denotado $\mathcal{S} \cup \mathcal{A} \mid\sim m$), (2) $\mathcal{S} \cup \mathcal{A}$ es consistente, y (3) $\mathcal{A}$ es el menor subconjunto (con respecto a la inclusión de conjuntos) que cumple las dos condiciones anteriores.
Si $\mathcal{A}$ es un argumento para h, también se dirá que $\langle A, h\rangle$ es una *estructura de argumento*. Además $\langle B, q\rangle$ es un *subargumento* de $\langle A, h\rangle$ si y sólo si, $B \subseteq A$.

Definición 6.3 Sea $\mathbf{L} = \{l : l$ es un literal básico para el cuál existe una prueba rebatible $\}$, y sean $\langle A_1, h_1\rangle$,$\langle A_2, h_2\rangle$ dos estructuras de argumento. Sea $\mathcal{S}_P$ el conjunto de hechos básicos de $\mathcal{S}$, y $\mathcal{S}_G$ el conjunto de reglas generales de $\mathcal{S}$, ($\mathcal{S}_G \cup \mathcal{S}_P = \mathcal{S}$). Se dirá que $\langle A_1, h_1\rangle$ es *estrictamente más específico* que $\langle A_2, h_2\rangle$, si y sólo si,
i) $\forall E \subseteq \mathbf{L}$, si $\mathcal{S}_G \cup E \cup A_1 \mid\sim h_1$ y $\mathcal{S}_G \cup E \not\mid\sim h_1$, entonces $\mathcal{S}_G \cup E \cup A_2 \mid\sim h_2$.
ii) $\exists E' \subseteq \mathbf{L}$ tal que $\mathcal{S}_G \cup E' \cup A_2 \mid\sim h_2$, $\mathcal{S}_G \cup E' \not\mid\sim h_2$, y $\mathcal{S}_G \cup E' \cup A_1 \not\mid\sim h_1$.

Definición 6.4 : Se dirá que $\langle A_1, h_1\rangle$ *contraargumenta a* $\langle A_2, h_2\rangle$ en un literal h, sssi, (1) existe un subargumento $\langle A, h\rangle$ de $\langle A_2, h_2\rangle$ tal que $\mathcal{S} \cup \{h, h_1\}$ es inconsistente, y (2) para todo subargumento propio $\langle B, q\rangle$ de $\langle A_1, h_1\rangle$, no es el caso que $\mathcal{S} \cup \{h_2, q\}$ sea inconsistente.

Definición 6.5 $\langle A_1, h_1\rangle$ *derrota a* $\langle A_2, h_2\rangle$ en un literal h, sssi, existe un subargumento $\langle A, h\rangle$ de $\langle A_2, h_2\rangle$ tal que: $\langle A_1, h_1\rangle$ contraargumenta a $\langle A_2, h_2\rangle$ en el literal h y
(1) $\langle A_1, h_1\rangle$ es estrictamente más específico que $\langle A, h\rangle$ (derrotador propio), o
(2) $\langle A_1, h_1\rangle$ no puede compararse con $\langle A, h\rangle$ (derrotador de bloqueo).

Definición 6.6 : Un *árbol de dialéctica* para $\langle A, h\rangle$, denotado $\mathcal{T}_{\langle A, h\rangle}$, se define recursivamente como sigue:

1. Un nodo que contiene una estructura de argumento $\langle A, h\rangle$ sin derrotadores (propios o de bloqueo), es un árbol de dialéctica para $\langle A, h\rangle$, y es también la raíz del árbol.
2. Supóngase que $\langle A, h\rangle$ es una estructura de argumento con derrotadores (propios o de bloqueo) $\langle A_1, h_1\rangle$, $\langle A_2, h_2\rangle$, ..., $\langle A_n, h_n\rangle$. El árbol de dialéctica $\mathcal{T}_{\langle A, h\rangle}$, para $\langle A, h\rangle$ se construye poniendo a $\langle A, h\rangle$ como nodo raíz, y haciendo que este nodo sea el padre de las raices de los árboles de dialéctica de $\langle A_1, h_1\rangle$, $\langle A_2, h_2\rangle$, ..., $\langle A_n, h_n\rangle$, *i.e.*, $\mathcal{T}_{\langle A_1, h_1\rangle}$, $\mathcal{T}_{\langle A_2, h_2\rangle}$, ..., $\mathcal{T}_{\langle A_n, h_n\rangle}$.

Definición 6.7 Los nodos de un árbol de dialéctica $\mathcal{T}_{\langle A, h\rangle}$ se etiquetan recursivamente como *nodos no-derrotados* (nodos-U) y *nodos derrotados* (nodos-D) como sigue:

1. Las hojas de $\mathcal{T}_{\langle A, h\rangle}$ son *nodos-U*.
2. Sea $\langle B, q\rangle$ un nodo interno de $\mathcal{T}_{\langle A, h\rangle}$. $\langle B, q\rangle$ será un *nodo-U* sssi todo hijo de $\langle B, q\rangle$ es un *nodo-D*. $\langle B, q\rangle$ será un *nodo-D* sssi tiene al menos un hijo que es *nodo-U*.

Definición 6.8 : $\langle A, h\rangle$ es una *justifición* para h sssi la raíz de $\mathcal{T}_{\langle A, h\rangle}$ es un nodo-U.

Definición 6.9 *Conjuntos de respuestas de un* PLR $\mathcal{P}$

1. El *conjunto de respuestas positivas* de $\mathcal{P}$ es un conjunto finito de literales L tal que para todo $h \in L$, existe una justificación para h.
2. El *conjunto de respuestas negativas* de $\mathcal{P}$ es un conjunto finito de literales L tal que para todo $h \in L$, se cumple: (a) o bien para cada argumento $\mathcal{A}$ de h, existe un derrotador propio de $\mathcal{A}$; o bien (b) no existe un argumento para h, pero existe un argumento $\mathcal{B}$ que es una justificación para $\sim h$
3. El *conjunto de respuestas indecisas* de $\mathcal{P}$ es un conjunto finito de literales L tal que para todo $h \in L$, se cumple que para todo argumento $\mathcal{A}$ de h, $\mathcal{A}$ no tiene derrotadores propios, pero si tiene al menos un derrotador de bloqueo.
4. El *conjunto de respuestas desconocidas* de $\mathcal{P}$ es un conjunto posiblemente infinito de literales que no pertenecen a los conjuntos de respuestas anteriores.

Definición 6.10 Dado un PLR $\mathcal{P}$ y una meta definida m, un interprete de programas lógicos rebatibles, responderá:

- SI, en el caso que m pertenezca al conjunto de respuestas positivas.
- NO, en el caso que m pertenezca al conjunto de respuestas negativas.
- INDECISO, en el caso que m pertenezca al conjunto de respuestas indecisas.
- DESCONOCIDO, en el caso que m pertenezca al conjunto de respuestas desconocidas.

7. Referencias

[1] Alferes José J. Pereira Luis M. *Contradiction: when avoidance equals removal (part I and II).* Proc. of Extensions of Logic Programming, 4th International Workshop ELP'93 St. Andrews U.K. March 1993.

[2] Dung Phan M. *Negations as Hypotheses: an Abductive Foundation for Logic Programming.* Proc. of 8th. Int. Conf. on Logic Programming (ICLP) 1991.

[3] Dung Phan M. *An Argumentation Semantics for Programming with Explicit Negation.* Proc. of 10th. Int. Conf. on Logic Programming (ICLP) 1993.

[4] Dung Phan M. *On the Acceptability of Argumets and its Fudamental Role in Nonmonotonic Reasoning and Logic Programming.* Proc. of Int. Join Conf. on Artificial Intelligence (IJCAI) 1993.

[5] Fillottrani, Pablo R. *Sistemas de razonamiento no monótono y su relación con la semántica de las bases de datos deductivas* Tesis de Magister en Ciencias de la Computación, Bahía Blanca, Universidad Nacional del Sur, Octubre de 1995.

[6] García A. J. *Una aproximación a la programación en lógica rebatible.* 2do. Workshop en Aspectos Teoricos de la Inteligencia Artificial (ATIA'95). Bahía Blanca, Octubre de 1995.

[7] García A. J. y Simari G. R. *Compilación de programas lógicos que utilizan la negación por falla. Una extensión de la máquina abstracta de Warren* 1er. Congreso Argentino de Ciencias de la Computación. Bahía Blanca, Noviembre de 1995.

[8] García A. J., Chesñevar C. I., and Simari G. R. *Making Argument Systems Computationally Attractive.* Proc. XIII Int. Conf. of the Chilean Society for Computer Science, October 1993.

[9] García A. J., Chesñevar C. I., and Simari G. R. *Bases de Argumentos: su mantenimiento y revisión.* XIX Conferencia Latinoamericana de Informática. Buenos Aires, Agosto 1993.

[10] Gelfond M. and Lifschitz V. *Logic Programs with Classical Negation.* Proc. of 7th. Int. Conf. on Logic Programming (ICLP) 1990

[11] Inoue K. *Extended Logic Programming with Default Assumptions.* Proc. of 8th. Int. Conf. on Logic Programming (ICLP) 1991.

[12] Lloyd J. W. *Foundations of Logical Programming.* 2nd. edition, Springer-Verlag 1987

[13] Nute Donald, *Basic defeasible logic*, in *Intensional Logics for Programming*, Ed by Luis Fariñas del Cerro, Claredon Press – Oxford (c) 1992.

[14] Simari G. R. , Chesñevar C. I., and García A. J. *The Role of Dialectics in Defeasible Argumentation.* XIV Int. Conf. of Chilean Computer Science Society, November 1994.

[15] Simari G. R. and Loui R. P. *A Mathematical Treatment of Defeasible Reasoning and its Implementation.* Artificial Intelligence, 53: 125–157,1992.

[16] Simari G. R. y García A. J. *A Knowledge Representation Language for Defeasible Argumentation.* XXI Conferencia Latinoamericana de Informática. Canela, Brasil, Agosto 1995.