Una aproximación a la programación en lógica rebatible

Alejandro J. García[1]

Grupo de Investigación en Programación Declaratica e Inteligencia Artificial (GIPDIA)
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) 553933 e-mail: ccgarcia@criba.edu.ar

Resumen

El objetivo de este trabajo es describir un lenguaje de programación en lógica rebatible como una extensión de la programación en lógica convencional. Para ello se utilizarán los *programas lógicos rebatibles* (PLR), los cuales están formados por dos tipos de cláusulas: las *cláusulas de programa extendido* [4] que permiten inferir tanto información positiva como negativa, y las *cláusulas de programa rebatible* [11] que a diferencia de las anteriores representan información tentativa que puede ser utilizada en la medida que no sea contradecida. Los PLR permiten la utilización de la negación clásica y la negación por falla, esto posibilita la representación de diferentes formas de "CWA" como cláusulas de programa.

En extensiones anteriores de la programación en lógica, al derivar dos literales complementarios se obtiene todo el lenguaje [4], o no hay forma de decidir entre ellos [5]. Para solucionar este tipo de problemas, nuestro desarrollo utiliza los conceptos de la argumentación rebatible para poder trabajar con información tentativa y potencialmente inconsistente. La noción de inferencia rebatible permite construir un *argumento* como un subconjunto de un PLR, para luego utilizar los conceptos de *derrota* y *justificación* de un sistema argumentativo como motor de inferencia del sistema. La semántica de un PLR queda caracterizada por los conjuntos de respuestas afirmativas, negativas, indecisas y desconocidas. Actualmente se encuentra en construcción una maquina abstracta para la ejecución de programas en lógica rebatible [3], basado en esta semántica, para lo cual se extenderá la máquina abstracta de Warren [15].

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

Una aproximación a la programación en lógica rebatible

1. Introducción

La programación en lógica es uno de los principales exponentes de la programación declarativa. Aunque se la ha utilizado como herramienta de representación de conocimiento, presenta limitaciones para adaptarse al razonamiento rebatible. En los últimos años se han desarrollado extensiones de la programación en lógica que incorporan algunos aspectos del razonamiento rebatible. Por ejemplo, Gelfond y Lifschitz en su trabajo "*Logic Programming with Classical Negation*" [4] introducen los programas lógicos extendidos (PLE) que permiten la representación de cláusulas de programa con antecedentes negados. Los PLE presentan ventajas importantes a la hora de representar conocimiento. Pero lamentablemente cuando dos literales complementarios (por ej. P y $\neg P$) pueden derivarse de un PLE, el *conjunto de respuestas* del programa es igual a todo el lenguaje. Los agentes inteligentes tienden a razonar en una forma rebatible: conclusiones previas son refutadas ante la presencia de mayor información. La argumentación rebatible [13, 1, 10, 14] es una formalización que intenta capturar este tipo de razonamiento. Como habitualmente el conocimiento maneja información tentativa (defaults), y la representación de información tentativa en la mayoria de los casos lleva a bases de conocimiento inconsistentes, en la mayoria de los casos el programa derivará todo el lenguaje.

Kowalsky y Sadri, en "*Logic Programming with Exceptions*" [6], utilizan los PLE para poder representar información negativa, y los extienden para la representación de excepciones a reglas generales. Por ejemplo, los pingüinos son una excepción a la regla "las aves vuelan". La idea que proponen es representar las reglas generales con cláusulas de la forma "$V \leftarrow A$" y las excepciones con cláusulas "$\neg V \leftarrow P$", luego a la hora de calcular la respuesta, las excepciones tienen prioridad sobre las reglas. Lamentablemente la forma en que diferencian una regla de una excepción, es que las primeras tienen un consecuente positivo y las últimas un consecuente negado. Esto representa una restricción muy grande, que trae aparejada confusiones, ya que la cláusula "los mamíferos no vuelan" representaría una excepción a la regla "los murcielagos vuelan".

El trabajo de Katsumi Inoue "*Extended Logic Programming with Default Assumptions*" [5] extiende el trabajo de Gelfond y Litfschitz para poder trabajar con información potencialmente inconsistente. También ataca el problema de los conjuntos de respuestas inconsistentes, pero lo hace separando la información absoluta, de la información tentativa o hipotética. De esta forma un "sistema de conocimiento" es un par (T, H) donde T es un conjunto de hechos que son verdaderos en el dominio en cuestión, y H es un conjunto de hipótesis. La tarea del sistema es hallar un subconjunto E de H tal que el conjunto de respuestas de $T \cup E$ sea consistente. El enfoque de Inoue tiene cierta semejanza con un sistema argumentativo. No obstante, presenta dificultades cuando se tiene que elegir

entre dos explicaciones contradictorias (en [2] puede encontrarse un análisis más detallado de estas extensiones).

El objetivo de este trabajo es describir un lenguaje de programación en lógica rebatible como una extensión de la programación en lógica convencional; de forma tal de capturar los aspectos del razonamiento rebatible, y solucionar los problemas de las extensiones mencionadas anteriormente. Para ello se utilizarán los *programas lógicos rebatibles* (PLR), los cuales están formados por dos tipos de cláusulas: las *cláusulas de programa extendido* [4] y las *cláusulas de programa rebatible* [11]. El lenguaje quedará separado en dos conjuntos disjuntos: uno que representa conocimiento estricto (seguro), y otro que contiene información rebatible (tentativa). Esto se verá en la sección 2.

En la sección 3 se definirá la noción de inferencia rebatible, la cuál permite construir un *argumento* como un subconjunto de un PLR. Una vez obtenido un argumento para una meta del programa, se podrán utilizar los conceptos de *derrota* y *justificación* de un sistema argumentativo como motor de inferencia del sistema. La semántica de un PLR quedará caracterizada entonces por cuatro conjuntos de repuestas: el conjunto de respuestas positivas, formadas por los literales positivos que pueden inferirse del programa; el conjunto de respuestas negativas, compuesto por la información negativa que puede derivar el programa; el conjunto de respuestas indecisas, formado por aquellos literales para los cuales no es posible decidir entre sus argumentos a favor y en contra; y por último el conjunto de respuestas desconocidas por el programa. Actualmente se encuentra en construcción una maquina abstracta para la ejecución de programas en lógica rebatible [3], basado en esta semántica, para lo cual se extenderá la máquina abstracta de Warren [15].

Los PLR permiten la utilización de la negación clásica y la negación por falla, esto posibilita la representación de diferentes formas de "CWA" como cláusulas de programa, esto se encuentra desarrollado en la sección 5.

2. Lenguaje de representación

En este lenguaje, un *literal* "l" es un átomo "a" o un átomo negado "$\neg a$", siguiendo la definición de Lloyd en [7]. El símbolo $\neg$ representará la negación clásica (*i.e.*, $\neg\neg a = a$), y el símbolo *not* la negación por falla. El símbolo "$\sim$" en cambio, se utilizará 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.

Definición 2.1 : *Cláusula de programa extendido* (CPE) [11]
Una CPE, es una cláusula de programa [7], que puede contener tanto la negación clásica como la negación por falla. Es una cláusula 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*. □

Definición 2.2 : *Cláusula de programa rebatible* (CPR) [11]
Una cláusula de programa rebatible 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 CPE de una CPR, 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 A son buenas razones para creer en l". □

Todas las variables en las cláusulas se asumen universalmente cuantificadas. Las cláusulas de programa no poseen un valor de verdad, y deben interpretarse como reglas de inferencia y no como condicionales. La semántica informal es " si el antecedente es verdadero, entonces el consecuente es verdadero". Las CPEs se utilizarán para representar conocimiento seguro (o no rebatible) como "ave(X) ← pigüino(X)", mientras que las CPRs se utilizarán para representar información rebatible como "vuela(X) $\prec$ ave(X)".

Como convención tipográfica, se utilizarán letras minúsculas para denotar literales (l), mayúsculas para conjuntos de literales (L), mayúsculas caligráficas para conjuntos de cláusulas ($\mathcal{L}$), y la convención habitual de PROLOG para cláusulas de programa. En algunas ocasiones una cláusula se notará "$c \leftarrow L$" donde L representará la conjunción de los elementos antecedentes.

Las cláusulas de programa pueden contener dos tipos de negación. Por un lado, puede utilizarse la negación clásica ($\neg$) que permite expresar cláusulas como "¬carnívoro(X) ← caballo(X)" para indicar que los caballos no son carnivoros, o "trabajo(X) $\prec$ ¬feriado(X)" para explicar que habitualmente los días que no son feriados voy a trabajar (rebatible de muchas maneras). Por otro lado se tiene la negación por falla *not*, la cual es útil para casos como "busco(X) $\prec$ perdido(X),*not* muerto(X)" (si no se si X está muerto, y está perdido, entonces lo busco), o "¬cruzar_la_vía $\prec$ *not* ¬viene_el_tren" (no cruzar la via si no estoy seguro que no viene el tren).

3. Programas lógicos rebatibles

Definición 3.1 : *Programa lógico rebatible* (PLR)
Un programa lógico rebatible, es un conjunto finito de CPE y CPR. Si $\mathcal{P}$ es un PLR, por conveniencia se distinguirá con $\mathcal{S}$ al conjunto de CPE de $\mathcal{P}$ y con $\mathcal{D}$ al conjunto de CPR de $\mathcal{P}$. □

Definición 3.2 : *Meta definida*
Una meta definida es un literal m. □

Definición 3.3 : *Prueba rebatible para una meta definida m*
Dado un PLR $\mathcal{P}$, una prueba rebatible a partir de $\mathcal{P}$ para una meta definida m, es un conjunto finito de CPE y CPR definido recursivamente de la siguiente forma:

1. Si existe un hecho "$c \leftarrow$ TRUE" en $\mathcal{P}$, tal que m unifica con c (con unificador más general (umg) σ), el conjunto $\{c\sigma \leftarrow \text{TRUE}\}$ es una prueba rebatible para m.

2. Si existe una presuposición "$c \prec$ TRUE" en $\mathcal{P}$, tal que m unifica con c (con umg σ), el conjunto $\{c\sigma \prec \text{TRUE}\}$ es una prueba rebatible para m.

3. Si existe una CPE "$c \leftarrow L$" 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, $\{c\sigma \leftarrow L\sigma\} \cup (\bigcup_i \mathcal{F}_i)$ es una prueba rebatible para m. A fin de que no se produzcan ciclos, la cláusula "$c\sigma \leftarrow L\sigma$" no debe aparecer en ninguno de los conjuntos $\mathcal{F}_i$.

4. Si existe una 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, $\{c\sigma \prec L\sigma\} \cup (\bigcup_i \mathcal{F}_i)$ es una prueba rebatible para m. A fin de que no se produzcan ciclos, la cláusula "$c\sigma \prec L\sigma$" no debe aparecer en ninguno de los conjuntos $\mathcal{F}_i$.

5. Si m es una submeta que tiene el operador de negación por falla *not* (*i.e.*, $m = not\ l$), y no existe una prueba rebatible para el literal l, entonces el conjunto vacio es una prueba rebatible para m.

6. Si no se da ninguno de los casos anteriores, entonces no existe una prueba rebatible para m.

Si existe una prueba rebatible para m a partir de $\mathcal{P}$, se dirá además que $\mathcal{P}$ deriva rebatiblemente a m. □

Proposición 3.1 : *Dado un* PLR $\mathcal{P}$, *el proceso de hallar una prueba rebatible para una meta definida m termina en un tiempo finito.* □

Definición 3.4 : *Consistencia*
Un conjunto de cláusulas de programa $\mathcal{A}$ es consistente si no se puede derivar rebatiblemente un par de literales complementarios. Reciprocamente, un conjunto de cláusulas de programa $\mathcal{A}$ es inconsistente si puede derivarse rebatiblemente un par de literales complementarios. □

Dado un PLR $\mathcal{P}$, el conjunto $\mathcal{S}$ de $\mathcal{P}$ debe ser consistente, mientras que el conjunto $\mathcal{D}$ y el propio $\mathcal{P}$ (*i.e.*, $\mathcal{S} \cup \mathcal{D}$) pueden ser inconsistentes. Esta es la forma en que un PLR puede contener información potencilamente inconsistente. En el ejemplo 3.1, $\mathcal{S}$ es consistente, mientras que $\mathcal{P}$ no lo es.

Ejemplo 3.1 :

perro(chicho) $\leftarrow$ TRUE
toro(pepe) $\leftarrow$ TRUE
$\neg$carnivoro(X) $\leftarrow$ toro(X)
tiene-cuernos(X) $\prec$ toro(X)
$\neg$peligroso(X) $\prec$ $\neg$carnivoro(X)
peligroso(X) $\prec$ $\neg$carnivoro(X), tiene-cuernos(X)
peligroso(X) $\leftarrow$ *not* $\neg$peligroso(X)

Aquí es posible derivar rebatiblemente perro(chicho), toro(pepe), $\neg$ carnivoro(pepe), tiene-cuernos(pepe), $\neg$ peligroso(pepe), y peligroso(pepe). Además, utilizando la regla (de prudencia) "peligroso(X) $\leftarrow$ *not* $\neg$peligroso(X)" podemos derivar "peligroso(chicho)", ya que no sabemos nada acerca de su peligrosidad. □

La noción de prueba rebatible no prohibe que de un PLR $\mathcal{P}$ puedan probarse rebatiblemente dos literales complementarios. En el ejemplo 3.1, pueden derivarse: "$\neg$ peligroso(pepe)", y "peligroso(pepe)". Resulta necesario entonces un criterio de inferencia, a fin de que solo una de las metas complementarias sea aceptada como una nueva creencia. Para ello se definirá un *argumento* como un subconjunto de CPR de un programa lógico rebatible, y una vez hecho esto se podrán utilizar los conceptos de: *justificación* y *derrotador* definidos en [12]. Esto se desarrollará en la sección siguiente.

4. Los PLR y la argumentación rebatible

Un *sistema argumentativo* [13, 1, 10, 14] es una formalización del razonamiento rebatible. Un *argumento* $\mathcal{A}$ para una hipótesis h es un pieza tentativa de razonamiento que un agente está dispuesto a aceptar como una explicación para h. Se seguirá aquí el sistema propuesto en [13] y [1], que acepta un argumento $\mathcal{A}$ como una razón rebatible para una conclusión h, cuando $\mathcal{A}$ es una *justificación* de h.

El proceso de obtención de una justificación para h, involucra la construcción de un *argumento aceptable* para h, a partir de la base de conocimiento del sistema. Para decidir la aceptabilidad de un argumento, se construyen *contraargumentos* que son posibles *derrotadores* (ver apéndice definiciones 7.2 y 7.3). 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. Ver el apéndice para obtener mayores detalles sobre estos conceptos.

En este lenguaje, se definirá *argumento* como un subconjuto de un PLR, y luego se utilizarán los conceptos de derrotador y justificación, para determinar el comportamiento de un interprete para el lenguaje.

Definición 4.1 : *Argumento*
Sea $\mathcal{P}$ un PLR, $\mathcal{S}$ el conjunto de CPE de $\mathcal{P}$, y $\mathcal{D}$ el conjunto de CPR de $\mathcal{P}$. Un argumento $\mathcal{A}$ para un literal h, es un subconjunto de CPR instanciadas de $\mathcal{D}$, tal que:

1. Existe una prueba rebatible de m a partir de $\mathcal{S} \cup \mathcal{A}$.
2. $\mathcal{S} \cup \mathcal{A}$ es consistente.
3. $\mathcal{A}$ es el menor subconjunto 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. □

En el ejemplo 3.1, es posible construir el argumento $\mathcal{A}$={ peligroso(pepe) $-\prec$ ¬carnivoros(pepe), tiene-cuernos(pepe); tiene-cuernos(pepe) $-\prec$ toro(pepe) } que es una justificación para "peligroso(pepe)". Aunque existe otro argumento $\mathcal{B}$= {¬peligroso(pepe) $-\prec$ ¬carnivoros(pepe) } para "¬peligroso(pepe)", $\mathcal{B}$ no es una justificación porque $\mathcal{A}$ derrota a $\mathcal{B}$.

Dado un PLR, es posible identificar un conjunto finito de literales que tendrán una justificación. Estos literales corresponden a los que un razonador respondería afirmativamente ante una consulta.

Definición 4.2 : *Conjunto de respuestas positivas*
Sea $\mathcal{P}$ un PLR. El conjunto de respuestas positivas de $\mathcal{P}$ es un conjunto finito de literales L tal que para todo $h \in L$, existe un argumento $\mathcal{A}$ que es una justificación para h. □

Aplicando la definición anterior, el conjunto de respuestas positivas del ejemplo 3.1 es { toro(pepe), ¬carnivoro(pepe), tiene-cuernos(pepe), peligroso(pepe), perro(chicho), peligroso(chicho)}. Por otro lado, en el mismo ejemplo no es posible inferir "¬peligroso(pepe)", porque existe un argumento más específico para "peligroso(pepe)" (*i.e.*, un derrotador propio). En este caso, la respuesta para la consulta "¬peligroso(pepe)" debería ser negativa. También debería darse una respuesta negativa a la consulta "peligroso(ico)" en el ejemplo 4.1, ya que solo es posible construir una justificación para "¬peligroso(ico)".

Ejemplo 4.1 :

¬carnivoro(X) $\leftarrow$ caballo(X)
caballo(ico) $\leftarrow$ TRUE
¬peligroso(X) $-\prec$ ¬carnivoro(X)

□

De esta forma, el conjunto de respuestas negativas de un PLR queda caracterizado de la siguiente manera:

Definición 4.3 : *Conjunto de respuestas negativas*
Sea $\mathcal{P}$ un PLR. El conjunto de respuestas negativas de $\mathcal{P}$ es un conjunto finito de literales L tal que para todo $h \in L$, se cumple una de ls siguientes condiciones:

a) Para cada argumento $\mathcal{A}$ de h, existe un derrotador propio de $\mathcal{A}$

b) No existe un argumento para h, pero existe un argumento $\mathcal{B}$ que es una justificación para $\sim h$

□

Cabe destacar que el inciso (b) de la definición anterior es un caso novedoso, en el cuál el sistema argumentativo anteriormente no podía dar una respuesta. Lo que se intenta capturar aquí es la siguiente situación: aunque no puedo formular un argumento para h si tengo una justificación para $\sim h$ (como es el caso de" peligroso(ico)" en el ejemplo 4.1). En esta situación un razonador respondería que no vale h, lo cuál no quiere decir que con más conocimiento la respuesta podría ser otra.

Además de las respuestas positivas y negativas, existen consultas ante las cuales un razonador puede manifestarse indeciso. Este tipo de respuestas queda caracterizado por la siguiente definición.

Definición 4.4 : *Conjunto de repuestas indecisas*
Sea $\mathcal{P}$ un PLR. El conjunto de respuestas indecisas de $\mathcal{P}$ es un conjunto finito de literales L tal que para todo $h \in L$, se cumple una de estas condiciones:

a) para todo argumento $\mathcal{A}$ de h, $\mathcal{A}$ no tiene derrotadores propios, pero si tiene al menos un derrotador de bloqueo, (es decir, posee derrotadores que interfieren la justificación pero no puede probarse que sean mejores que $\mathcal{A}$).

b) no existe un argumento para h pero si ocurre lo siguiente: (i) existe un conjunto mínimo de literales J que no pueden derivarse de $\mathcal{P}$, (ii) J y $\mathcal{P}$ permiten la formación de un argumento $\mathcal{A}$ para h, y (iii) para algún $l \in J$, $\sim l \in \mathcal{S}$.

□

Aquí el inciso (b) de la definición anterior también es un caso novedoso, en el cuál el sistema argumentativo anteriormente no podía dar una respuesta. Lo que se intenta capturar aquí es la siguiente situación: no puedo formular un argumento para h con la información disponible en $\mathcal{P}$, pero aunque asuma los hechos faltantes, tampoco podría dar una respuesta afirmativa ya que todo argumento que podría construir sería inconsistente con $\mathcal{S}$.

Por último existe un conjunto de respuestas, que por no tener ningún tipo de información al respecto, son totalmente desconocidas. Por ejemplo, que número saldrá en la loteria mañana.

Definición 4.5 : *Conjunto de respuestas desconocidas*
Sea $\mathcal{P}$ un PLR. El conjunto de respuestas desconocidas de $\mathcal{P}$ es un conjunto posiblemente infinito de literales L tal que para todo $h \in L$, h no pertenece a los conjuntos de respuestas anteriores. □

Observación 1 : Dado un PLR $\mathcal{P}$, los conjuntos de respuestas positivas y negativas de $\mathcal{P}$, son ambos conjuntos consistentes (*i.e.*, no contienen un par de literales complementarios). En cambio, el conjunto de respuestas indecisas, es normalmente un conjunto inconsistente.

Una vez definido los posibles conjuntos de respuestas que puede generar un PLR, se establecerá como debe comportarse un interprete de este lenguaje ante una consulta.

Definición 4.6 : 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.
- INDEFINIDO, en el caso que m pertenezca al conjunto de respuestas indecisas.
- NO SE, en el caso que m pertenezca al conjunto de respuestas desconocidas.

□

5. Cláusulas de "CWA"

La mayoria de los lenguajes de programación en lógica trabajan bajo la *hipótesis de mundo cerrado* (CWA [2]) para todos los predicados de un programa como una propiedad metalinguística. Esto es, cada vez que falla la prueba de un predicado p, se asume $\neg p$, y esto vale para todos los predicados. Como ya se indicó anteriormente, el lenguaje definido en este trabajo posse tanto la negación por falla, como la negación clásica. Esto permite que la hipótesis de mundo cerrado para un predicado particular p en un PLR pueda lograrse incluyendo en el PLR, la cláusula "$\neg$p(X) $\leftarrow$ *not* p(X)", con lo cual le estamos indicando al programa que si no puedo probar p, entonces vale $\neg p$.

La CWA puede ser útil para algunos predicados en algunos contextos, por ejemplo "$\neg$muerto(X) $\leftarrow$ *not* muerto(X)", (esto es, si no puedo probar que X esta muerto, asumo que no está muerto). No obstante exiten numerosas ocasiones donde no es conveniente que se aplique CWA. Por ejemplo si un piloto de avión utiliza la siguiente regla "$\neg$pista-ocupada(X) $\leftarrow$ *not* pista-ocupada(X)", la vida de los pasajeros correría mucho peligro cuando no se posea información sobre la pista. Por lo tanto, parecería ser conveniente que CWA sea utilizada solo para algunos predicados específicos, y en determinados contextos.

Como las cláusulas de CWA se representan dentro del propio lenguaje, puede utilizarse otra forma de CWA: "p(X) $\leftarrow$ *not* $\neg$p(X)". Esto es, si no puedo probar que $\neg p$ es válido, asumo que vale p. En el caso del piloto de avión sería preferible utilizar la regla "pista-ocupada(X) $\leftarrow$ *not* $\neg$pista-ocupada(X)" en lugar de la citada anteriormente. En el ejemplo 3.1 se utilizó la regla "peligroso(X) $\prec$ *not* $\neg$peligroso(X)" también como una forma de ser prudente. Además de representar prudencia, este tipo

[2]En inglés closed world assumption o CWA

de reglas pueden expresar optimismo o esperanza por parte del razonador, por ejemplo: "gané-la-lotería $\leftarrow$ *not* ¬gané-la-lotería". También existen otros casos donde no es conveniente utilizar ninguna de las dos versiones. Por ejemplo ninguna de las siguientes cáusulas resulta adecuada: "¬ama(X,Y) $\leftarrow$ *not* ama(X,Y)", o "ama(X,Y) $\leftarrow$ *not* ¬ama(X,Y)".

Como contrapartida a las dos versiones anteriores, cuyo consecuente es complementario del antecedente, existen dos versiones más que pueden representarse: "p(X) $\leftarrow$ *not* p(X)" y "¬p(X) $\leftarrow$ *not* ¬p(X)", las cuales serían equivalentes entre si, porque conservan el mismo literal.

Por otro lado, el nuevo lenguaje definido nos permite representar un nuevo tipo de CWA, la CWA rebatible: "¬p(X) $\prec\!\!\!-$ *not* p(X)", es decir, no poder probar "p(X)" es una buena razón (rebatible) para asumir "¬p(X)".

6. Conclusiones

En el presente trabajo se ha definido un lenguaje de programación en lógica rebatible lo suficientemente expresivo como para capturar la mayoría de los ejemplos que se intentan representar con este tipo de razonamiento.

Se ha mostrado además que un argumento puede verse como un subconjunto de un PLR, y por lo tanto los PLR definen un lenguaje de representación de conocimiento adecuado para la argumentación rebatible. Por último los conjuntos de respuestas proveen una forma sencilla de definir el comportamiento de un interprete para los PLR.

La facilidad de disponer de la negación clásica y la negación por falla dentro del lenguaje, han abierto la posibilidad de representar diferentes reglas de "CWA" dentro de un PLR, y solo para aquellos predicados que fuera adecuada.

7. Apéndice: Conceptos de la argumentación rebatile

Las siguientes definiciones fueron extraidas de [12], y corresponden a los principales conceptos de un sistema argumentativo (por más detalles referirse a [13, 12, 1, 2]).

Definición 7.1 : Sea $\langle A, h\rangle$ una estructura de argumento, $\langle S, j\rangle$ es un subargumento de $\langle A, h\rangle$ si y solo si, $S \subseteq A$. □

Definición 7.2 : Se dirá que $\langle A_1, h_1\rangle$ *contraargumenta a* $\langle A_2, h_2\rangle$ en un literal h, si y solo si,

1. Existe un subargumento $\langle A, h\rangle$ de $\langle A_2, h_2\rangle$ tal que el conjunto $\mathcal{S} \cup \{h, h_1\}$ es inconsistente.
2. Para todo subargumento propio $\langle S, j\rangle$ de $\langle A_1, h_1\rangle$, no es el caso que $\langle A_2, h_2\rangle$contraargumenta a $\langle S, j\rangle$.

□

Definición 7.3 : $\langle A_1, h_1\rangle$ *derrota a* $\langle A_2, h_2\rangle$ en un literal h, si y solo si, 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 mejor[3] que $\langle A, h\rangle$, o
2. $\langle A_1, h_1\rangle$ no puede compararse con $\langle A, h\rangle$.

En el caso (1) se dirá que $\langle A_1, h_1\rangle$ es un *derrotador propio*, y en el caso (2) un *derrotador de bloqueo.* □

Definición 7.4 : 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 $\langle A, h\rangle$. Este nodo también es 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.*, $\mathcal{T}_{\langle A_1,h_1\rangle}$, $\mathcal{T}_{\langle A_2,h_2\rangle}$, ..., $\mathcal{T}_{\langle A_n,h_n\rangle}$. Si se forma una linea de argumentación que no es aceptable (ver [12]) durante la construcción de este árbol, se poda el subárbol cuya raíz sea la culpable de violar la condición de aceptabilidad de la linea de argumentación.

□

Definición 7.5 : 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 7.6 : Sea $\langle A, h\rangle$ una estructura de argumento, y sea $\mathcal{T}_{\langle A,h\rangle}$ su árbol de dialéctica asociado. Se dirá que A es una *justifición* para h sssi el nodo raíz de $\mathcal{T}_{\langle A,h\rangle}$ es un U-node. □

[3]Habitualmente se utiliza la especificidad como criterio de comparación, pero cualquier otro orden parcial definido sobre el conjunto de los argumentos puede ser utilizado.

8. Referencias

[1] 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.

[2] García A. J. *Extensiones de la programación en lógica.* Reporte Técnico GIPDIA-UNS-1995. Universidad Nacional del Sur.

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

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

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

[6] Kowalski R. and Sadri F. *Logic Programs with Exceptions.* Proc. of 7th. Int. Conf. on Logic Programming (ICLP) 1990

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

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

[9] J. L. Pollock. *Defeasible Reasoning. Cognitive Science*, 11:481–518, 1987.

[10] Prakken H. *Logical Tools for Modelling Legal Arguments (Ph.D. Thesis).* Vrije University, Amsterdam, Holanda. January 1993.

[11] Simari G. R. and García A. J. *A Knowledge-Representation Language for Defeasible Argumentation.* XXI Latin American Conference on Informatics. Canela, Brazil, Agosto de 1995.

[12] 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.

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

[14] Vreeswijk G. *Studies in Defeasible Argumentation (Ph.D. Thesis).* Vrije University, Amsterdam, Holanda. March 1993.

[15] Warren David H. D. *An Abstract Prolog Instruction Set* Thechnical Note 309, SRI International, Menlo Park, CA, October 1983.