# Un Operador General de Contracción para Programas Lógicos

Claudio A. Vaucheret
Departamento de Informática y Estadística
Universidad Nacional del Comahue

Guillermo R. Simari
Departamento de Ciencias de la Computación
Universidad Nacional del Sur[1]

e-mail cvaucher@uncoma.edu.ar, grs@criba.edu.ar

**Palabras clave**: Programacion Lógica, Teoría de Cambio de Creencias, Debugging Declarativo

**Resumen**

En este trabajo, presentamos un operador de contracción para el caso de programas lógicos. La clase de programas tratados es la de programas lógicos normales extendidos con la negación clásica. Es decir, dichos programas contemplan tanto la negación por falla (NAF) y la negación clásica. El operador de contracción está basado en la contracción Kernel[2] de la Teoría de Cambio de creencias. Para remover un literal de las inferencias de un programa, se reconocen cuales son los componentes del programa que contituyen las justificaciones de ese literal y se opera sobre ellos para obtener un nuevo programa contraído.

[1]Miembro de GIIA (Grupo de Investigación en Inteligencia Artificial) e ICIC (Instituto de Ciencias e Ingeniería de Computación), UNS, Bahía Blanca

# Un Operador General de Contracción para Programas Lógicos

## 1 Introducción

Un componente muy importante de los futuros sistemas de programación lógica serán los sistemas de debugging declarativos, es decir sistemas de depuración de errores que puedan ser usados sin necesidad de entender el funcionamiento computaciónal del sistema. Estos sistemas solo necesitan conocer la interpretación intentada de un programa incorrecto para ayudar a depurar un error. Una de las tareas a realizar es la detección y eliminación de alguna regla incorrecta que produzca que un hecho que no está en el significado intentado ocurra dentro del significado de un programa. Un programa lógico consiste de un conjunto de cláusulas que puede pensarse como una teoría lógica particular. La revisión de creencias en teorías lógicas brinda el esquema para modificar estas teorías en una forma que es similar a lo que en los programas depuración o debugging. Claramente, un agente que posea una representación interna del espacio en el que desarrolla esta tarea necesita modificar su base de conocimiento. Si un programa lógico se lo considera como un estado epistémico de un agente, la tarea de remover una regla incorrecta sería una operación de cambio de creencias, en este caso una contracción del estado epistemico para no producir lo que no está en el significado intentado.

La Teoría del Cambio de Creencias[1] estudia los aspectos formales del proceso de contraer una teoría para eliminar una proposoción, o revisar una teoría para introducir una proposición. Esta teoría modela estados de creencias por medio de conjunto de proposiciones clausurados logicamente y considera tres tipos de operaciones sobre los estados de creencia: Expansión, Contracción y Revisión. Expandir una teoria $A$ con $\alpha$, $A+\alpha$ significa adicionar $\alpha$ a $A$ requiriendo que el el resultado sea un nuevo estado de creencias posiblemente inconsistente. Contraer $A$ por $\alpha$, $A \div \alpha$ significa remover $\alpha$ de $A$ de tal modo que el resultado sea un nuevo estado de creencia. Revisar $A$ con $\alpha$, $A * \alpha$ significa adicionar $\alpha$ de tal modo de obtener un nuevo estado de creencias consistente. Contracción y revisión tienen el requerimiento de que los cambios hechos sean tan pequeños como sea posible de manera tal de minimizar la innecesaria pérdida de información.

En este trabajo, presentamos un operador de contracción cuando el estado epistémico, en lugar de ser una teoría lógica, es el significado de un programa lógico. Es decir definimos una operación sobre un programa que obtenga un nuevo programa con la característica de que su significado deje de contener una creencia no deseada. En el caso específico de la programación en lógica, dicha creencias será un hecho peerteneciente al significado del programa. Para realizar esta tarea el operador reconoce primero cuales son, dentro del programa, las justificaciones o explicaciones para ese literal. Para luego obtener un nuevo programa produciendo una incisión en cada una de las justificaciónes. De esta forma el significado del nuevo programa no contemplará a dicho literal. Esta aproximación, es similar a la contracción kernel[2] de la teoría de cambio de creencias, donde las justificaciones, llamadas *kernels*[2], son subconjuntos minimales de la teoría que infieren la sentencia a contraer.

La clase de programas tratados es la de programas lógicos normales extendidos con la negación clásica. Es decir, dichos programas contemplan tanto la negación por falla

[2]Se utiliza el vocablo en inglés por no encontrar uno en castellano que le correspondiese en la forma adecuada.

(NAF) y la negación clásica.

Para una mejor comprensión del paper primero presentaremos una introducción del operador de contracción para conjuntos clausurados de proposiciones dentro de la teoría de cambio de creencias. Especialmente la contracción kernel. Luego en la seccion 3 se definirá la contracción en el ámbito de los programas lógicos.

## 2 Contracción kernel

El sistema AGM [1] ofrece un excelente modelo para cambio de creencias. Un estado de creencias es representado como una teoría. Un operador de contracción, toma un estado de creencias y una sentencia particular y retorna un nuevo estado de creencias con la propiedad de que no infiere dicha sentencia. En este modelo, se define la *parcial meet* contracción, que dado un estado de creencias y una sentencia, devuelve el estado formado por la coincidencia entre los estados seleccionados desde un conjunto de candidatos maximales que no infieren la sentencia. Dicho conjunto de candidatos son los estados maximales en el sentido de perder la menor cantidad de creencias como sea posible. Se debe eliminar creencias solo cuando es imprescindible hacerlo.

Las siguientes definiciones formalizan la *parcial meet* contracción. Primero se define el conjunto de candidatos, llamado conjunto resto.

**Definición 2.1** *: Sea $A$ un conjunto de sentencias y $\alpha$ una sentencia. Sea $A \perp \alpha$ ("A resto alpha ") el conjunto de conjuntos tal que $B \in A \perp \alpha$ si y solo si:*

1. $B \subseteq A$
2. $\alpha \notin Cn(B)$
3. *No existe conjunto $B'$ tal que $B \subset B' \subseteq A$ y $\alpha \notin Cn(B')$*

Es decir que el conjunto $A$ resto $\alpha$ es el conjunto formado por todos los subconjuntos maximales de $A$ que no infieren a $\alpha$ . El tercer punto de la definición determina que son maximales.

El modelo AGM introduce una función que selecciona un subconjunto entre todos los candidatos posibles, dicha función se define como sigue.

**Definición 2.2** *Sea $A$ un conjunto de sentencias. Una función de selección para $A$ es una función $\gamma$ tal que para toda sentencia $\alpha$:*

1. *Si $A \perp \alpha$ es no vacío, entonces $\gamma(A \perp \alpha)$ es un subconjunto no vacío de $A \perp \alpha$ y*
2. *Si $A \perp \alpha$ es vacío. entonces $\gamma(A \perp \alpha) = \{A\}$.*

Por último, el nuevo estado de creencias será la intersección o coincidencia entre los seleccionados.

**Definición 2.3** *Sea $A$ un conjunto de sentencias y $\gamma$ una función de selección para $A$. La* parcial meet *contracción sobre $A$ que es generada por $\gamma$ es la operación $\sim_\gamma$ tal que para toda sentencia $\alpha$: $A \sim_\gamma \alpha = \bigcap \gamma(A \perp \alpha)$ Un operador $\div$ sobre $A$ es de* parcial meet *contracción si y solo si hay una función de selección $\gamma$ tal que para toda sentencia $\alpha$: $A \div \alpha = A \sim_\gamma \alpha$.*

En esta forma la contracción *Parcial Meet* está basada en la selección entre subconjuntos de la teoría que no implican la sentencia a ser descartada. Otra aproximación para definir una operación de contracción, propuesta por Alchourrón y Makinson [2] es seleccionar en cambio las sentencias a ser descartadas, es decir basar el operador de contracción en la selección entre los elementos de la teoría que contribuyen a hacer que esta implique la sentencia a contraer. En lugar de identificar conjuntos maximales que no infieren $\alpha$ la contracción kernel identifica que elementos de la teoría contibuyen para inferir $\alpha$ y seleccionar entre ellos cuales descartar.

Las sentencias que nosotros removemos con el fin de contraer $A$ por $\alpha$ deberían ser sentencias que contribuyen en que $A$ infiera $\alpha$. Por ejemplo si $p$, $q$ y $r$ son sentencias logicamente independientes y

$$A = \{p, q, p \to q, q \to p, r\}$$

Si estamos contrayendo $p \wedge q$ de $A$. Consideraremos los siguientes tres subconjuntos:

$$\{p, q\}$$
$$\{p, p \to q\}$$
$$\{q, q \to p\}$$

Cada uno de estos conjuntos implica $p \wedge q$, y cada uno de ellos es también minimal en el sentido de que no tienen subconjuntos propios que implican $p \wedge q$. Estos subconjuntos son llamados los $p \wedge q$-kernels de $A$.

Una sentencia en $A$ contribuye a que $A$ implique $p \wedge q$ si y solo si es un elemento de algún $p \wedge q$-kernel. Por lo tanto en el momento de seleccionar que elementos excluir de $A$ para obtener $A \div (p \wedge q)$, debemos elegir entre esas sentencias que son elementos de algún $p \wedge q$-kernel. Entre las sentencias seleccionadas para remover debe haber al menos un elemento de cada uno de los kernels.

Para expresarlo en un lenguaje formal, veamos las siguientes definiciones:

**Definición 2.4** *Sea $A \subseteq \mathcal{L}$ y $\alpha \in \mathcal{L}$. Entonces $A\|\alpha$ es el conjunto tal que $X \in A\|\alpha$ si y solo si:*

- $X \subseteq A$
- $X \vdash \alpha$ *y*
- *si* $Y \subset X$, *entonces* $Y \nvdash \alpha$

*$A\|\alpha$ es un conjunto kernel y sus elementos son los $\alpha$-kernels de $A$*

La función que selecciona las sentencias a remover será llamada una función de incisión ya que hace una incisión dentro de cada $\alpha$-kernel.

**Definición 2.5** *Una función de incision $\sigma$ para $A$ es una funcion tal que para todo $\alpha$*

- $\sigma(A\|\alpha) \subseteq \bigcup(A\|\alpha)$
- *si* $\emptyset \neq X \in A\|\alpha$, *entonces* $X \cap \sigma(A\|\alpha) \neq \emptyset$

Si la operación $\div$ de contracción es basada en la función de incisión $\sigma$, entonces $A \div \alpha = A - \sigma(A\|\alpha)$.

Este enfoque de considerar candidatos minimales que infieren $\alpha$ en lugar de candidatos maximales que no infieren $\alpha$ fue también utilizado en la actualización de base de conociminetos por medio de sistemas de mantenimiento de verdad basado en asumsiones en [4] y [7]

Los sistemas de mantenimiento de verdad basado en asumsiones (ATMS) fueron introducidos por[3]. Informalmente, la actividad de estos sistemas es, dada una base de conocimiento compuesta de conocimiento general y conocimiento particular y dada una sentencia, obtener y mantener las porciones de conocimiento particular necesarias para inferir dicha sentencia.

**Definición 2.6** *Supongamos que hemos dividido nuestra base de conocimiento en un conjunto $C$ de conocimiento común y un conjunto $A$ de otra información. Entendemos por una* explicación *para una sentencia $p$ un subconjunto minimal $E$ de $A$ que es consistente con $C$ y tal que*

$$E \cup C \models p$$

**Definición 2.7** *Un sistema de mantenimiento de verdad basado en Asumsiones, es todo sistema que toma una base de datos $D = C \cup A$ y una sentencia $q$ y retorna alguno o todas las explicaciones para $q$.*

Una de las aplicaciones de estos sistemas es la actualización consistente de base de datos. Supongamos que tenemos una base de datos lógica $D$ y queremos agregar un hecho $p$. Una actualización consistente debe remover lo necesario de la base de datos para que no infiera $\neg p$, es decir que debe contraer la base de datos por $\neg p$. El ATMS puede entonces encontrar todas las explicaciones para $\neg p$ y eliminar un elemento de cada explicación de manera que $D$ ya no infiera $\neg p$. Es decir para hacer una base de datos $D$ consistente con un nuevo hecho p:

1. Uso un ATMS para encontrar las explicaciones para $\neg p$ en $D$, llamamos esas explicaciones $e_1, \ldots, e_k$.
2. Encuentro un conjunto minimal H que tiene la propiedad de que $H \cap e_i \neq \emptyset$ para todo $e_i$
3. Remuevo $H$ de $D$

Cada una de las explicaciones es un subconjunto de $A$, es decir que consiste de conocimiento particular. Esta división de los datos en dos categorías permite a un conjunto de datos ser no revisables. Es decir el conocimiento general no formará parte del conjunto minimal $H$.

En la sección siguiente se aplicará este enfoque para la contracción de programas lógicos en lugar de base de datos y conjunto de creencias. Definiremos para el caso particular de programas lógicos, cuáles son las explicaciónes o kernels, de un hecho inferido por el programa y el modo de realizar una operación de contracción basado en inhabilitar las dichas explicaciones.

# 3 Contracción de Programas

En esta sección definiremos un operador de contracción general para programas lógicos. Para ello debemos definir explicaciones y revisiones para el caso particular de programas lógicos. Un programa lógico será definido como un conjunto de reglas de la forma:

$$H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m.\ \ M >= 0, n >= 0$$

$H, B_1, \ldots, B_n, C_1, \ldots, C_m$ son literales clásicos. Un literal clásico es o un atomo $A$ o su negación clásica $\neg A$. El operador *not* es el común de negación por falla (NAF). Un literal clásico $L$ y el literal default *not* $L$ son literales complementarios. con $Comp(L)$ notaremos el literal complementario de $L$, es decir $Comp(notA) = A$ y $Comp(A) = notA$.

Para todo programa $P$, llamaremos el significado de $P$, $Sign(P)$, al modelo de $P$ asignado por una semántica determinada. Dicha semántica puede ser bi-valuada como la semántica de modelos estables [5] o tri-valuada como la semántica bien fundada [10]. Los programas lógicos que trataremos en este trabajo cuentan con dos tipos de negación, la negación por falla y la negación clásica $\neg$, esta clase de programas se llaman programas lógicos extendidos y el significado de estos programas estará dado por semánticas extendidas que pueden ser también bi-valuadas[6] o tri-valuadas [9] [8].

**Ejemplo 1** *consideremos el siguiente programa $P$ y para definir su significado utilizaremos la semántica bien fundada de programas lógicos.*

$$
\begin{array}{l}
p \leftarrow \neg r, not\ q. \\
p \leftarrow not\ b. \\
\neg r \leftarrow a. \\
b \leftarrow q, c. \\
a.
\end{array}
$$

*donde el significado de $P$ es $Sign(P) = \{a, \neg r, not\ c, not\ q, not\ b, p\}$*

### 3.1 Explicaciones de un Literal

Intuitivamente, la explicación de un literal, será aquella porción del programa que contribuye en inferir dicho literal. La explicación de un literal dentro de un programa lógico, es el equivalente al kernel de una sentencia en la teoría de Cambio de creencias o a la justificación de una sentencia en ATMS.

Tanto en la teoría de Cambio de Creencias, como en los sistemas de mantenimiento de verdad basado en asumsiones, el kernel o las justificaciones son un subconjunto de la teoría o la base de conocimiento original que infiere la sentencia. Para el caso de programas lógicos, una explicación de un literal es algo mas complejo que un subconjunto del programa. Debido al operador de negación por falla, un programa lógico puede inferir literales basado en suposiciones, es decir dicho operador permite que ante la ausencia de una información el programa pueda inferir literales. Por lo tanto una explicacion de un literal en el contexto de un programa debe estar estructurado por una porcion del programa, sumado a un conjuto de suposiciones constituído por información que el programa no debe inferir. La definición formal de una explicación es:

**Definición 3.1** *Dado un programa $P$ y un literal $c$ una* explicación *en $P$ es un triple $\langle I, O, c\rangle$ minimal donde se cumplen las siguientes condiciones:*

- $I \subseteq P$, $O \subseteq Lit(P)$,
- $c \in Sign(I)$,
- $c \notin Sign(I \cup O)$
- *para todo $x \in O$ se tiene que not $x \in Sign(P)$*

llamamos $Lit(P)$ a todos los literales clasicos que ocurren en alguna regla de P. La explicación es minimal, en el sentido de que no existe otro triple $\langle I', O', c\rangle$ que cumpla las condiciones de arriba y $I' \subset I$ y $O' \subset O$. El componente $I$ de una explicación contiene las reglas del programa necesarias para inferir $c$ y el componente $O$ contiene los elementos que no deben estar en el significado del programa de manera de poder inferir $c$ al que llamamos el consecuente de la explicación. De esta manera, una explicación se puede invalidar con la ausencia de un elemento de $I$ o con la presencia de un elemento de $O$ en el significado del programa.

**Ejemplo 2** *Si tomamos el programa del ejemplo 1 con su significado, las siguientes son ejemplos de explicaciones en P para el literal p:*

- $\langle\{p \leftarrow \neg r, not\ q., \neg r \leftarrow a., a.\}, \{q\}, p\rangle$
- $\langle\{p \leftarrow not\ b.\}, \{q, b\}, p\rangle$
- $\langle\{p \leftarrow not\ b.\}, \{c, b\}, p\rangle$

Teniendo en cuenta un programa y su significado, se puede definir un procedimiento que obtenga las explicaciones de un literal dado. Dicho procedimiento es el siguiente:

***entrada**: un programa P , $Sign(P)$ su significado asociado y L un literal.*
***salida**: una explicación $e = \langle I, O, L\rangle$ en P*

*Si L es un literal clásico:*
*Selecciono una regla $L \leftarrow B_1, \ldots, B_n$ donde cada $B_i \in Sign(P)$*
*asigno a e el triple $\langle I, O, L\rangle$*
*inicializado con los siguientes valores*
*$I = \{L \leftarrow B_1, \ldots, B_n.\}$ y $O = \emptyset$*
*para cada $B_i$:*
*obtengo recursivamente una explicación $\langle I'_i, O'_i, B_i\rangle$,*
*asigno $I \leftarrow I \cup I'_i$ y $O \leftarrow O \cup O'_i$*
*Retorno e*

*Si L es un literal default notA:*
*Selecciono de cada regla $A \leftarrow B_1, \ldots, B_n$, un elemento*
*de su cuerpo $B_j$ tal que $B_j \notin Sign(P)$*
*asigno a e el triple $\langle I, O, L\rangle$*
*inicializado con los siguientes valores*
*$I = \emptyset$ y $O = \{A\}$*
*para cada $B_i$:*
*obtengo recursivamente una explicación $\langle I'_j, O'_j, Comp(B_j)\rangle$,*
*asigno $I \leftarrow I \cup I'_j$ y $O \leftarrow O \cup O'_j$*
*Retorno e*
*Si no existe ninguna regla con esa condición retorno $e = \langle\{\}, \{A\}, L\rangle$*

El procedimiento para encontrar una explicación $\langle I, O, L\rangle$ busca una regla que contribuya en inferir $L$, es decir una regla cuya cabeza sea $L$ y todos los literales del cuerpo pertenezcan al significado del programa, dicha regla pasa a formar parte del componente $I$ de la explicación. Luego esta explicación debe completarse con una explicación para cada

literal del cuerpo. Los componentes de estas explicaciones se agregan respectivamente a los componentes $I$ y $O$ de la explicación para $L$. Para el caso de que $L$ sea un literal default, es decir *not* $A$, Todas las reglas con $A$ en la cabeza están inactivas por algún elemento que no pertenece al significado del programa. El procedimiento construye una explicación para $L$ formada por las explicaciones de los complementos de dichos elementos.

**Ejemplo 3** *Tomando el programa $P$ y su sugnificado del ejemplo 1, veamos ejemplos de explicaciones calculadas por el procedimiento anterior:*

- $\langle\{a.\},\{\},a\rangle$
  dado que $a.$ es la unica regla para $a$

- $\langle\{\neg r \leftarrow a., a.\},\{\},\neg r\rangle$
  como $a \in Sign(P)$ se selecciona la regla $r \leftarrow a.$. Primero se asigna a $e$,

  $$e = \langle\{\neg r \leftarrow a.\},\{\},\neg r\rangle$$

  y luego de obtener la explicación de $a$ el procedimiento devuelve

  $$e = \langle\{\neg r \leftarrow a., a.\},\{\},\neg r\rangle$$

- $\langle\{\},\{q\},not\ q\rangle$
  dado que no hay reglas para $q$

- $\langle\{\},\{b,q\},not\ b\rangle$
  De la única regla para $b$, selecciono $q$ ya que $q \notin Sign(P)$ . El procedimiento primero asigna a $e$,

  $$e = \langle\{\},\{b\},not\ b\rangle$$

  y luego de obtener la explicación de $not\ q$ devuelve

  $$e = \langle\{\},\{b,q\},not\ b\rangle$$

- $\langle\{\},\{b,c\},not\ b\rangle$
  igual que en el caso anterior pero seleccionando $c \notin Sign(P)$

- $\langle\{p \leftarrow \neg r, not\ q., \neg r \leftarrow a., a.\},\{q\},p\rangle$
  seleccionando la primera regla para $p$ provisto que $not\ q$ y $\neg r$ pertenecen a $Sign(P)$ el procedimiento sucesivamente asigna a $e$
  $e = \langle\{p \leftarrow \neg r, not\ q.\},\{\},p\rangle$
  $e = \langle\{p \leftarrow \neg r, not\ q.\},\{q\},p\rangle$ al obtener la explicación de $not\ q$
  $e = \langle\{p \leftarrow \neg r, not\ q., \neg r \leftarrow a., a.\},\{q\},p\rangle$ al obtener la explicación de $\neg r$

- $\langle\{p \leftarrow not\ b.\},\{q,b\},p\rangle$
  seleccionando la segunda regla para $p$ provisto que $not\ b$ pertenece a $Sign(P)$ el procedimiento sucesivamente asigna a $e$
  $e = \langle\{p \leftarrow not\ b.\},\{\},p\rangle$
  $e = \langle\{p \leftarrow not\ b.\},\{b,q\},p\rangle$ al obtener la explicación de $not\ b$

Con el fin de realizar una operación de contracción de un programa $P$ por un literal $L$, se debe herir cada una de las explicaciones de $P$ cuyo consecuente sea $L$. El programa resultante de la operación de contracción $P \div L$ no debe contener explicaciones con $L$ como consecuente. La forma de invalidar una explicación $\langle I, O, L\rangle$ es *revisar* alguno de los

elementos de $I$ o de $O$. En el primer caso $P \div L$ no debe contener algún elemento de $I$, y en el segundo para algún elemento $x$ de $O$ *not* $x$ no debe pertenecer a $Sign(P \div L)$.

En lugar de considerar a todas las reglas de un programa o sus literales default como revisables se pueden clasificar los elementos de un programa distinguiendo aquellos que uno desea proteger de la revisión y aquellos que son revisables. En la actualización de bases de conocimiento por medio de ATMS, se divide la base de conocimiento en conocimiento común y conocimiento adicional, y las justificaciones son subconjuntos de esta última clase, por lo tanto el conocimiento a ser dejado de lado en el momento de la actualización. En la siguiente subsección definiremos teniendo en cuenta esta aproximación, las explicaciones revisables y las revisiones para un programa.

### 3.2 Explicaciones Revisables y Revisiones

Cuando para un atomo $L$ no hay reglas en un programa $P$, *not* $L$ es inferido por el programa debido a la asumsión de mundo cerrado dado por la negación por falla. En el momento de decidir que elementos de una explicación se deben revisar en una operación de contracción sería coherente considerar primero estas asumsiones realizadas por la negación por falla que por ejemplo alguna regla del programa. Dentro de las cláusulas de un programa, los hechos pueden ser considerados mas revisables que las reglas. Este criterio es similar a la actualización de base de conocimientos, donde el conocimiento particular es mas revisable que el conocimiento general. En este caso los hechos serían un conocimiento particular de una aplicación y las reglas de un programa se considerarían un conocimiento mas general. Para distinguir los elementos de un programa factibles de revisión introduciremos la definición de *revisables* de un programa.

**Definición 3.2** *Dado un programa $P$ los* revisables *de $P$ $Rev(P)$ es un par $\langle \mathcal{I}, \mathcal{O} \rangle$ donde $\mathcal{I} \subseteq P$ y si $x \in \mathcal{O}$ entonces not $x \in Sign(P)$*

**Ejemplo 4** *Si consideramos por ejemplo de la política de tomar a los hechos de un programa y sus asumsiones por negacion por falla, los elementos revisables del programa $P$ del ejemplo 1 son:*

$$Rev(P) = \langle \{a.\}, \{q, c\} \rangle$$

*$a.$ es el único hecho del programa y los literales $q$ y $c$, no tienen reglas en $P$*

Dejando establecido que no todos los elementos de un programa son revisables, surge la posibilidad de que la operación de contracción no pueda realizarse para algún literal dado. La definición de revisables de un programa nos permite definir un operador de semi-contracción, es decir que la contracción $P \div L$ se realizará solo si se pueden invalidar todas las explicaciones de $L$.

**Definición 3.3** *Dado un programa $P$ donde $Rev(P) = \langle \mathcal{I}, \mathcal{O} \rangle$. Sea $e = \langle I, O, c \rangle$ una explicación en $P$. Si $\mathcal{I} \cap I = \emptyset$ y $\mathcal{O} \cap O = \emptyset$ entonces la explicación $e$ es no revisable. En caso contrario, $e$ es revisable.*

Si ninguno de los elementos de una explicación es revisable, entonces dicha explicación no se puede invalidar en una operación de contración. Esto nos permite definir cuando se puede aplicar una operación de contracción a un programa. Para poder contraer un programa $P$ por $L$, todas las explicaciones $\langle I, O, L \rangle$ del programa, deben ser revisables, porque si alguna explicación se mantuviera vigente, $L$ seguiría pertenecientdo al significado de $P$.

**Definición 3.4** *Un programa $P$ se puede contraer por $L$ si todas las explicaciones de $P$ con consecuente $L$ son revisables.*

Para la operación de contracción nos interesa considerar solo los elementos revisables. Definiremos entonces un tipo de explicaciones que contenga solo los elementos plausibles de revisión.

**Definición 3.5** *Dado un programa $P$ donde $Rev(P) = \langle \mathcal{I}, \mathcal{O} \rangle$. Sea $e = \langle I, O, c \rangle$ una explicación en $P$. Si $e$ es revisable, entonces $\langle \mathcal{I} \cap I, \mathcal{O} \cap O, c \rangle$ es un* soporte revisable *en $P$.*

Un soporte revisable $\langle I, O, L \rangle$ de P contiene componentes revisables del programa $P$ que contribuyen a que $L$ pertenezca a $Sign(P)$. El concepto de soporte revisable corresponde al de kernel para una sentencia en la definición de contracción kernel en la teoría de cambio de creencias.

**Ejemplo 5** *Sea $P$ el programa del ejemplo 1, suponga $Rev(P) = \langle \{a\}, \{q, c\} \rangle$. todas las explicaciones de p son revisables y los soportes revisables son:*

- $\langle \{a.\}, \{q\}, p \rangle$
- $\langle \{\}, \{q\}, p \rangle$
- $\langle \{\}, \{c\}, p \rangle$

Dado un programa $P$ y un literal $L$, vamos a definir una revisión de $L$ en ese programa como un conjunto de elementos de $P$ que deben ser removidos con el fin de contraer $L$. Por lo tanto dicha revisión debe contener elementos de cada uno de los soportes revisables con consecuente $L$

**Definición 3.6** *Sea $P$ un programa contraíble por $L$, una* revisión *es un triple $\langle I, O, L \rangle$ minimal tal que para todo soporte revisable en $P$ $\langle I', O', L \rangle$ se cumple ya sea que $I \cap I' \neq \emptyset$ o $O \cap O' \neq \emptyset$*

Una revisión es una selección de elementos de un programa a ser retirados con el fin de realizar la operación de contracción.

**Ejemplo 6** *Si tomamos los soportes revisables del ejemplo anterior, $\langle \{\}, \{q, c\}, p \rangle$ es una revision de $P$, para el caso de $Rev(P) = \langle \{a\}, \{q, c\} \rangle$.*

## 3.3 Operador de Contracción

Para un programa $P$ y un literal $L$ que pertenece a $Sign(P)$ debemos definir un operador de contracción $\div$ de manera que $P \div L$ sea un nuevo programa cuyo significado no contenga a $L$. En la teoría de cambio de creencias esta propiedad del operador es llamado el postulado de éxito. Es decir

$$L \notin Sign(P \div L)$$

Otra propiedad deseable del operador de contracción es la de *inclusión* que postula que la contracción no debe agregar nuevos elementos a la teoría. En el caso de los programas lógicos, dicha propiedad sería

$$Sign(P \div L) \subseteq Sign(P)$$

Otros postulados establecen la minimalidad del cambio en el sentido de cambiar lo menos posible de la teoría original con el fin de lograr el éxito de la contracción. Para los programas lógicos, se deben considerar dos casos, según si el significado de un programa está basado en una semántica tri-valuada o bi-valuada. En el caso de una semántica tri-valuada, la operación de contracción puede hacer que le valor de un literal pase de verdadero a indefinido, mas que de verdadero a falso como en el caso bi-valuado. Para realizar estos cambios mínimos, se pueden utilizar reglas de inhibición que prohiben la falsedad de un átomo en algún modelo.

**Definición 3.7** *La regla de inhibición para un átomo $A$, es $A \leftarrow not\ A$.*

En una semántica tri-valuada, para todo programa $P$ conteniendo una regla de inhibición para un átomo $A$, el valor de verdad de $A$ en todo modelo de $P$ no es falso. Y si esta es la única regla para A, el valor de verdad de $A$ es indefinido en todos los modelos.

La operación de contracción sobre un programa $P$ dada una revisión $\langle I, O, L\rangle$ debe eliminar las reglas pertenecientes a $I$ y provocar que los elementos de $O$ dejen de tener el valor falso. Es decir debe agregar al programa una regla de inhibición por cada elemento de $O$. En el caso bi-valuado debe agregar al programa un hecho por cada elemento de $O$.

Veamos primero el operador de contracción para el caso de semánticas trivaluadas, donde las reglas de inhibición realizan un mínimo cambio en el programa.

**Definición 3.8** *Sea $P$ un programa contraíble por $L$ y sea $\langle I, O, L\rangle$ una revisión, entonces $P \div L = (P - I) \cup R$. donde $R$ es un conjunto de regals de inhibición $r \leftarrow not\ r$ donde*

- *$r \in O$ o*
- *$r$ es la cabeza de una regla perteneciente a $I$.*

**Ejemplo 7** *En nuestro ejemplo, $P \div p$ es el programa:*

$$
\begin{array}{l}
p \leftarrow \neg r, not\ q.\\
p \leftarrow not\ b.\\
\neg r \leftarrow a.\\
b \leftarrow q, c.\\
a.\\
q \leftarrow not\ q.\\
c \leftarrow not\ c.
\end{array}
$$

*cuyo significado es $Sign(P \div p) = \{a, \neg r, \}$. El valor de verdad de $q, c$ es indefinido y por lo tanto también lo son los valores de $b$ y $p$.*

Esta operación cumple los postulados de éxito e inclusión. En el caso de semánticas bi-valuadas, el carácter no monotónico de la negación por falla, hace que las propiedades de inclusión no se cumpla. Para definir un operador de contracción bi-valuado, que cumpla la condición de éxito, utilizaremos el siguiente procedimiento. Dado un programa $P$ y un literal $L$ se inhabilitan todas las explicaciones con consecuente $L$ , obteneinedo un nuevo programa $Q$. Si $Q$ no contiene en su significado a $L$, la operación de contracción estará cumplida. Puede ocurrir sin embargo que nuevas explicaciones surjan para $L$, en ese caso se vuleve a obtener una revisión y se modifica el programa, hasta que la condición de exito sea alcanzada o hasta que el programa no pueda ser contraído. El procedimiento es el siguiente:

*Dados $P$ un programa y $L$ un literal clásico.*
*Si $P$ es contraíble por $L$ , sea $\langle I, O, L\rangle$ una revisión*
*obtengo un nuevo programa $Q = (P - I) \cup O$.*
*Si $L \notin Sign(Q)$, retorno $Q$*
*en caso contrario repito lo siguiente hasta que $L \notin Sign(Q)$:*
*si $Q$ no es contraíble por $L$ termino diciendo que $P$ no es contraíble.*
*en caso contrario sea $\langle I, O, L\rangle$ una revision de $Q$,*
*a $Q$ le asigno un nuevo programa $(Q - I) \cup O$*

En cada paso el programa nuevo se obtiene retirando las reglas del componente $I$ y agregando como hechos los elementos del componente $O$. Esto produce que se eliminen todas las explicaciones encontradas hasta entonces. Si surgen nuevas, el procedimiento continúa.

**Ejemplo 8** *En nuestro ejemplo, $P \div p$ es el programa:*

$$
\begin{array}{l}
p \leftarrow \neg r, not\ q. \\
p \leftarrow not\ b. \\
\neg r \leftarrow a. \\
b \leftarrow q, c. \\
a. \\
q. \\
c.
\end{array}
$$

*el significado de este programa es $Sign(P \div p) = \{a, q, c, b, \neg r, not\ p\}$*

# 4 Conclusiones

Se presentó un operador de contracción para el caso de programas normales extendidos. Dicho operador es dependiente de la selección a priori de cuales son los elementos revisables de un programa. Dicha selección define cuando un programa es contraíble. El operador de contracción retira de un programa todas las explicaciones para el literal a contraer. Para el caso de semánticas trivaluadas el operador cumple con los postulados de cambio de creencia de éxito, inclusión y minimalidad.

# Referencias


[1] C. Alchourrón, P. Gärdenfords, and D. Makinson. On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[2] C. E. Alchourrón and D. Makinson. On the Logic of Theory Change: Safe Contraction. *Studia Logica*, 44:405–422, 1985.

[3] J. de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence*, 28:127–162, 1986.

[4] R. Fagin, J. D. Ullman, and M. Y. Vardi. On the Semantics of updates in databases. In *Proceedings Second ACM Symposium on Principles of Database Systems*, pages 352–365, Atlanta, Georgia, 1988.

[5] M. Gelfond and V. Lifschitz. The Stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *5th Int. Conf. on LP*, pages 1070–1080. MIT Press, 1988.

[6] M. Gelfond and V. Lifschitz. Logic progams with classical negation. In *ICLP'90*, pages 579–597, 1990.

[7] M. L. Ginsberg. Counterfactuals. *Artificial Intelligence*, 30:35.

[8] L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *European Conf. on AI*, pages 102–106. John Wiley & Sons, 1992.

[9] T. Przymusinski. Extended stable semantics for normal and disjunctive programs. In *ICLP'90*, pages 459–477, 1990.

[10] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.