**Semántica Operacional para Lógicas No-Monótonas**

En este capítulo se plantean los algoritmos estudiados para la obtención de modelos de las semánticas no-monótonas WFS, CWFS y STABLES utilizando programas lógicos normales, que se les aplica: instanciación, re-escritura, completitud y forma clausular, y al final utilizar software de internet para llegar al resultado final o sea los modelos (Stables).

Primeramente veremos como se instancia un programa lógico normal, utilizando el método ground [12], y además se revisó otro método de instanciación que se detalla en [14], Consecuencias inmediatas en programas con variables.

Una vez que se instancia un programa lógico normal, se aplicaron las transformaciones básicas [9], y además se utilizo la completitud de Clark [30,29], para tratar de llegar al número mínimo de posibles modelos Stables.

Para llegar a la forma clausular se utilizaron dos métodos, el primero utiliza la FNC (Forma No Canónica) [25], para conseguir un crecimiento lineal al ser utilizado, y el segundo, utiliza OTTER (herramienta de internet) [26], que utiliza la forma canónica o tradicional (que son reglas de lógica clásica).

Y además SATO (herramienta de internet) que nos da el resultado final (obtención de modelos) a través del método de Davis-Putnam, para mayor detalle consultar [27].

Además se obtuvieron los modelos mínimos y se comprobaron los modelos STABLES, mediante el método Gelfond-Lifschitz[19].

Finalmente se muestra el algoritmo para la obtención de modelos de las semánticas WFS, STABLE y CWFS.

Además, como nota adicional, comentaremos que se reforzó la investigación con consultas por medio de E-mail con el Ph.D. Patrik Simons quien es el creador de S-models, para entender la funcionalidad de esta herramienta y entender el algoritmo de Ground como método de instanciación.

**4.1 Métodos de Intanciación**

*4.1.1 Algoritmo de Ground. PRELIMINARES[12]*

Consideremos programas lógicos normales, que consisten de reglas de la forma:

$c \leftarrow .. a_1,\ldots,a_n, not(b_1),\ldots, not(b_m).$

donde $a_i$, $b_j$ y $c$ son átomos. Dado un programa lógico $P$, primero tomaremos este conjunto de instancias ground que frecuentemente se denota $P_g$.

Reformulemos la definición de Stable Models (SM) en términos de operador de cerradura deductivo *Dcl*(.,.) que usaremos para describir una aproximación para calcular SM, llamaremos átomos y literales negadas (No-átomo de la forma *not*(*a*)) para un conjunto de literales *S*, por $S^+$ ( $S^-$) denotamos el conjunto de literales (literales negadas) en *S*.

**Ejemplo 20:** *S={a,not(b),not(c)}*,

entonces $S^+$ ={*a*} y $S^-$={*not*(*b*), *not*(*c*)}.

Para un conjunto de átomos S, sea not(*S*)= {not( $\emptyset$ )| $\emptyset \subset$ .. S}. Se denota por *Dcl*(*P*,*L*) la cerradura deductiva de un conjunto de reglas ground de *P*, siendo un programa lógico normal y un conjunto de literales *L*.

*Dcl*(*P*,*L*) es un conjunto más pequeño de átomos que contiene $L^+$ y que es cerrado bajo la inferencia de reglas *R*(*P*,*L*) donde

*R*(*P*,*L*)= { c $\leftarrow$ .. $a_1,\ldots,a_n$ | c $\leftarrow$ .. $a_1,\ldots,a_n$, *not*($b_1$),...., not($b_m$) $\subset$ .. P,{ *not*($b_1$),...., *not*($b_m$)} $\subset$ .. $L^-$}.

**Ejemplo 21:** Si P={q $\leftarrow$not(p); r $\leftarrow$ q},

entonces *R*(*P*,{p,not(p)})={q $\leftarrow$;r $\leftarrow$ .. q}y *Dcl*(*P*,{p,not(p)})={p,q,r}. Esto se ve directo, tal que un SM de un conjunto de reglas ground *P* se cumple si y sólo si

-

.

$\wedge$ =Dcl(P,not($\wedge$))
donde el segundo es el complemento de $\wedge$, es decir, los átomos que no están en $\wedge$. Esto es porque R(P,not($\wedge$ .. ))=P$\wedge$.

La caracterización de conjuntos para SM [13] son llamados conjuntos totales y estos conjuntos de átomos y el conjunto de literales negadas construidos de átomos en *NAnt*(*P*), antecedentes negativos en *P* , es decir átomos *b* tal que *not*(*b*) aparece en *P*.

**Ejemplo 22:** N Ant({q $\leftarrow$ not (p);r $\leftarrow$q})={p}.

Una observación crucial es que para la cerradura deductiva de *P*, una premisa *not*(*b*) juega un papel único cuando b$\subset$N ant(P), es decir,

–

*Dcl*(*P*,not($\wedge$ .. ))= *Dcl*(*P*, not(N Ant(P) - $\wedge$ .. ))

entonces un conjunto de átomos $\wedge$ .. es un SM de *P*

$\Leftrightarrow$ .. $\wedge$ .. = Dcl(P,not(N Ant (P)-$\wedge$ .. )).

Una vez visto algunos preliminares se verá un método desarrollado por Niemella y Simons [12], que calcula una versión ground P' de un programa *P* de rango-restringido(veremos más adelante), tal que *P* y P' tienen exactamente el mismo SM correspondiente al conjunto dado de literales *B*, pero P' es frecuentemente más pequeño con respecto a todo el conjunto de instancias ground de *P*. Primero se define el conjunto P' y entonces se presenta un algoritmo para calcular directamente desde *P* sin considerar todas las instancias ground. Para definir P', primero sean

A=B$\cup$ .. $B_n$ $\cup$ .. B

donde $B_n$={not(c)|c$\subset$ .. N Ant ($P_g$) - *B* y no es una regla en *P* teniendo el mismo predicado como *c* en la cabeza} y $B_p$={c es una regla ground en *P* con *c* como cabeza y todas las literales en el cuerpo en $B_n$ $\cup$ .. B}.
Entonces sea

A'=Dcl($P_g$, A $\cup$ .. not(N ant ($P_g$)-A))

y finalmente

P'={c $\leftarrow$ .. $a_1$,...,$a_n$, not($b_1$),...,not($b_m$) $\subset$ .. $P_g$| {$a_1$,...,$a_n$} $\subseteq$ .. A',{$b_1$,...,$b_m$}
A= $\emptyset$

Se presentará un algoritmo posteriormente, que calcula directamente desde *P* y *A* sin *P* ground, primero cuando *P* es un programa de rango-restringido. Un programa es de rango-restringido cuando por cada regla toma una variable que aparece en la cabeza o es una literal negada en el cuerpo, y además aparece como una literal positiva en el cuerpo. Veremos el algoritmo Ground.

**Algoritmo 1: GROUND**

**función ground**(*P*,*A*) [12]

$P_2$ : = { r$\subset$ .. P | las literales positivas del cuerpo de *r* estan en *A* y las literales negativos estan en $A^-$ o no estan en *not*($A^+$) }

*P*:=*P*-$P_2$; U := $\emptyset$;

N:=$A^+$ .. {a| existe una regla r $\subset$ .. $P_2$ cuya cabeza es *a*}

**while** *N* es no vacío **do**

Elije una a$\subset$ .. N;N:=N-{a};U:=U .. {a};

**for** cada regla *r* en *P* **do**

**for** toda regla r' resultante de una sustitución aplicada a *r* tal que *a* esta en el cuerpo de la regla r' **do**

**if** literal negativa no esta en el cuerpo de r' y esta en *not*($A^+$)

**then**

**if** r' es ground y todas son literales positivas en el cuerpo de r' esta en N .. U **then**

$P_2$:=$P_2$ .. {r'}

**if** la cabeza de r'no esta en *U* **then**

*N*:=*N* .. {la cabeza de r'}

**endif**

**else**

P:=P .. {r'}

**endif**

**endif**

**end for**

**end for**

**end while**

**return** $P_2$.

**Ejemplo 23:** Considere un programa lógico normal:

*a*:-*not b*.

*b*:-*not a*.

*p*:-*not a*.

*p*:-*not p*,*not c*.

*c*:-*d*.

*d*:-*c*.

De acuerdo al **algoritmo 1**, $P_2$ es el conjunto de reglas resultantes o sea el Ground (*P*) = P'.

*P* = {programa inicial}, en el ejemplo anterior.

*A* = total de literales.{*a*, *b*, *p*, *c*, *d*}.

$A^+$= {*c*, *d*}. Literales positivas del cuerpo de cada regla.

$A^-$ = {*a*, *b*, *p*}. Literales negativas del cuerpo de cada regla.

Haciendo el **Algoritmo 1**, iterativamente me da como resultado:

El ground P':

*a*:-*not b*.

*b*:-*not a*.

*p*:-*not a*.

*p*:-*not p*.

Observe que, *p*:-*not p*, *not c*, en general no esta incluido porque c$\subset$ .. $A^+$ y *c*:- *d*, *d*:- *c*, no esta incluido porque c,d $A^-$

En la siguiente sección aplicaremos transformaciones a programas ground (*P*) para generar el programa resto del cual se obtendrá directamente el WFM. Sin embargo, la construcción de la instanciación ground de *P* es una operación muy costosa.

Antes de mencionar el otro método de instanciación, daremos dos definiciones importantes:

**Definición 8**[9]: Sea $\rightarrow$ .. x denota nuestro sistema final re-escrito.

$\rightarrow$ .. x :=$\rightarrow$ .. p [redacted] .. $\rightarrow$ .. N[redacted] .. $\rightarrow$ .. S[redacted] .. $\rightarrow$ .. F[redacted] .. $\rightarrow$ .. L(Son el conjunto de reglas de trasformaciones que veremos en la próxima sección).

$\rightarrow$ .. p = reducción positiva.
$\rightarrow$ .. N= Reducción negativa.
$\rightarrow$ .. S= Sucess (Éxito).
$\rightarrow$ .. F= Failure (falla).
$\rightarrow$ .. L= Loop (Detección de ciclo).

**Definición 9**: [9] (Programa Resto). Sea *P* un programa, sea el programa P satisface:

1. ground(*P*) (P)$\rightarrow$ .. xP,
2. P es irreducible con respecto $\rightarrow$ .. x, es decir, no existe un P' con P $\rightarrow$ .. xP.
3. entonces P es llamado programa resto de *P*.

Consecuencias inmediatas en programas con variables [14].

**Ejemplo 24** ( Números pares No-ground): Considere el siguiente programa lógico Numpar2. *par*(0).

*par*(*Y*).

*sucesor*(0,1). *sucesor*(*n*-1,*n*).

que define, los números pares entre 0 y algún número par fijo *n*.

El procedimiento de punto fijo produce la siguiente secuencia: $K_0$ = {*par*(0), *sucesor*(0,1), ... , *sucesor*(*n*-1,*n*)}

$U_0$ = $K_0$ .. {*par*(2), *par*(3), ... , *par*(*n*)}

$K_1$ = $K_0$ .. {*par*(2)} $U_1$ = 208#208 {*par*(2), *par*(4), *par*(5), ..., *par*(*n*)}

$K_2$ = $K_0$ .. {*par*(2), *par*(4)}

$U_2$ = $K_0$ .. {*par*(2), *par*(4), *par*(6),*par(7), ...,par(n)}*

$K_{n/2}$ = $U_{n/2}$ = $K_0$ .. {*par*(2), *par*(4), *par*(6), ... , *par*(*n*)}.

Como en el ejemplo anterior *Numpar*2, da un número cuadrático para el cálculo del modelo bien fundamentado (WFM).

**Ejemplo 25:** Instanciación Ground de números pares.

Considere el programa lógico *Numpar*2, aplicamos las transformaciones que están definidas sólo para programas ground, y construimos la instanciación ground como sigue:

*ground*(*Numpar*2):

{par(0), sucesor(0,1), ... , sucesor(n-1, n),

par(0) $\leftarrow$ .. sucesor(0,0) $\wedge$ .. not par(0), ..., par(0) $\leftarrow$.. sucesor(n,0) $\wedge$.. not par(n),

par(1) $\leftarrow$ .. sucesor(0,1) $\wedge$ .. not par(0), ... , par(1) $\leftarrow$.. sucesor(n,1) $\wedge$.. not par(n),

par(n) $\leftarrow$ .. sucesor(0,n) $\wedge$ .. not par(0), ... , par(n) $\leftarrow$ .. sucesor(n,n) $\wedge$.. not par(n)}.

Usando el sistema de reescribir $\rightarrow$ .. x, entonces *ground*(*Numpar*) que es transformado dentro del programa residuo:

ground(Numpar2)$\rightarrow$ .. x {sucesor(0,1), ... , sucesor(n-1, n), par(0), par(2), ... , par(n) }.

Después de aplicar *ground*(*Numpar*2), se observa que tiene un número cuadrático de elementos y además, el número de transformaciones para obte-ner el modelo bien fundamentado es cuadrático.

El objetivo es construir un algorimo eficiente sin utilizar nuestro sistema de transformaciones, consideremos lo siguiente:

1. Fijando el orden de aplicación de nuestras transformaciones.
2. Eligiendo una apropiada estructura de datos para los hechos condicionales.
3. Evitando hacer el primer cálculo de la instanciación ground del programa.

Iniciamos con Detección de ciclos que aplicaremos al programa sin instanciar, después necesitamos calcular un pequeño subconcunto de *ground*(*P*), que se calcula eficientemente. La clave de esta idea, es generar desde el conjunto inicial de reglas instanciadas de *P*, que no tenga ninguna literal en el cuerpo que sea falsa. Consideramos una regla instanciada únicamente si hay razón de creer que todas las literales positivas del cuerpo, son posiblemente ciertas. Iniciaremos, con reglas instanciadas sin literales positivas en el cuerpo. Las cabezas de las reglas instanciadas son posiblemente ciertas, más adelante estas reglas instanciadas tienen a literales positivas en el cuerpo que pueden ser derivados sucesivamente.

Daremos unas definiciones importantes para este método y un ejemplo.

**Definición 10:** Consecuencias Inmediatas con variables [9].

Sea *P* un programa. *S* un conjunto de hechos condicionales, se da Tp que calcula el siguiente conjunto de hechos condicionales:

Tp(S):={A $\leftarrow$ .. B $\subset$ .. ground (P) | pos(B) $\subseteq$ .. cabezas(S)}.

Notesé que lfp(Tp)$\subseteq$ .. ground(*P*) y cabezas(lfp( Tp)) = lfp($T_{P,0}$).

**Definición 11:** (Instanciación de Programa relevante).

Sea *P* un programa lógico normal, definimos la instanciación de programa relevante $P_0$ $\cup$ .. ground(*P*) de *P* dado :

$P_0$ := lfp( Tp).

mostraremos que es correcto inicializar el cálculo con lfp( Tp) en lugar del programa residuo P

**Lema 1:** Sea *P* un programa lógico Normal y sea Tp definido anteriomente.

Entonces tenemos:

*ground*(*P*) $\rightarrow$ .. $_L$ *lfp*( Tp).

*lfp*( Tp) $\rightarrow$ .. $_x$ P

**Ejemplo 26:** Números pares revisados No-ground.

Consideramos nuevamente el programa *Numpar*2, siguiendo el Lema anterior, iniciamos el proceso de transformación en $Numpar2_0$ = lfp ($T_{Numpar2}$)en lugar de la instanciación *ground*(*Numpar*2) con :

$K_0$ = {*par*(0), *sucesor*(0,1), ... , *sucesor*(*n*-1,*n*)}

Tenemos,

$Numpar2_0 = K_0 \cup$ .. { par(1) $\leftarrow$ .. *sucesor*(0,1) $\wedge$ .. *not par*(0),

.

.

.

par(n) $\leftarrow$ .. *sucesor*(*n*-1,*n*) $\wedge$ .. *not par*(*n*-1)}.

Este conjunto inicial puede ser construido por *O*(*n*) pasos. La siguiente transformación *O*(*n*) convierte el conjunto inicial dentro del programa resto de *Numpar*2, del lema anterior.

**$Numpar2_0$ = lfp($T_{Numpar2}$)**

$\rightarrow$ .. $*_s$ $K_0 \cup$ .. {par(1) $\leftarrow$ .. not par(0),..., par(n) $\leftarrow$ .. not par(n-1)}

$\rightarrow$ .. $_N$ $K_0 \cup$ .. {par(2) $\leftarrow$ .. not par(1),..., par(n) $\leftarrow$ .. not par(n-1)}

$\rightarrow$ .. $_P$ $K_0 \cup$ .. {par(2), par(3) $\leftarrow$ .. not par(2),..., par(n) $\leftarrow$ .. not par(n-1)}

$\rightarrow$ .. $_N$ $K_0$ $\cup$ .. {par(2), par(4) $\leftarrow$ .. not par(3),..., par(n) $\leftarrow$ .. not par(n-1)}

$\rightarrow$ .. $_P$ $K_0$ $\cup$ .. {par(2), par(4) ,par(5) $\leftarrow$ .. not par(4),..., par(n) $\leftarrow$ .. not par(n-1)}

$\rightarrow$ .. $_N$ $K_0$ $\cup$ .. {par(2), par(4) ,par(6) $\leftarrow$ .. not par(5),..., par(n) $\leftarrow$ .. not par(n-1)}

$\rightarrow$ .. $_P$ $K_0$ $\cup$ .. {par(2), par(4) ,par(6), par(7) $\leftarrow$ .. not par(6),..., par(n) $\leftarrow$ not par(n-1)}

$\rightarrow$ .. $^*_x$ $K_0$ $\cup$ .. {par(2), par(4) ,par(6), ..., par(n)}.

En resumen, el modelo bien fundamentado del programa *Numpar*2 es calculado en $O(n)$ transformaciones.

El método de instanciación que utilizaremos es el de ground(P), además cabe señalar que utilizamos programas lógicos normales sin simbolos funcionales, si se utilizarán simbolos funcionales, provocaría que la instanciación se vuelva infinita.

**4.2 Métodos de Transformación (Compilación)**

***4.2.1Transformaciones Básicas***

Se introdujo un sistema para estudiar y calcular semántica negativa por medio de transformaciones básicas para programas ground(Red+, Red-, Success, Failure y Loop) [8], el cual es el que utilizaremos. Existe además, otro tipo de transformaciones que utilizan Red+, Red-, GPPE, Taut, Sub [1], que son utilizadas para el cálculo de WFS [11] y Stable, la diferencia de estos dos conjuntos de transformaciones, es que el primero borra menos modelos, lo cual al final del cálculo de modelos, es una desventaja, pero si se aplica reducción a través del primer conjunto de transformaciones y se le aplica el segundo conjunto de transformaciones, llegaríamos a programas muchos más reducidos y así dar posibles soluciones a través de los modelos encontrados, y el segundo conjunto de transformaciones tiene un crecimiento exponencial, es por esta razón, que utilizaremos el primer conjunto de transformaciones, que da un crecimiento lineal.

**Definición 12: Transformación de programa [9].** Un programa de transformación es una relación $\rightarrow$ .. entre programas lógicos ground. Una semántica $S$ permite una transformación $\rightarrow$ .. si y sólo si $S(P_1)=S(P_2)$ para todo $P_1$ y $P_2$ con $P_1 \rightarrow$ .. $P_2$'.

Las siguientes dos transformaciones describen la evaluación de literales

negativas en el cuerpo, en casos triviales.

**Definición 13: Reducción positiva [8].**

Sean $P_1$ y $P_2$ programas. $P_2$ resulta de $P_1$ por reducción positiva ( $P_1 \rightarrow$ .. $_P$ $P_2$) si y sólo si hay una regla $A \leftarrow$ .. B en $P_1$ y una literal negativa **not B** $\subset$ .. B tal que **B** not$\subset$ .. cabezas($P_1$ ), es decir, no hay alguna regla acerca de **B** en $P_1$ y

$P_2$ = ($P_1$- {A $\leftarrow$ .. B}) $\cup$ .. {A$\leftarrow$ .. (B-{**not B**})}

Podemos aplicar la transformación sin cambiar la semántica de $P_1$ .

**Ejemplo 27:** Sea un programa ground *P*

*a*:- *m*, $\neg$ .. *b,n*

*c*:-*h*

*h*:-.

De acuerdo a nuestra definición A={a}, B ={*m*, $\neg$ .. b,n}, **not B** = { $\neg$ .. *b}* donde $\neg$ .. *b* $\subset$ .. B en $P_1$; tal que *b* not $\subset$ .. cabezas($P_1$), donde P1={*a*:- *m*, $\neg$ .. *b,n*}; entonces,

$P_2$ = ($P_1$- {A $\leftarrow$ .. B}) $\cup$ .. {A$\leftarrow$ .. (B-{**not B**})} . Si sustituimos obtenemos lo siguiente:

La primera parte de la unión se hace cero, y la segunda parte se resta *m*, $\neg$ .. *b,n* de $\neg$ .. *b*, por lo tanto queda, $P_2$=*a*:-*m,n*. Aplicando directamente:

.

$$\begin{array}{lcl} a{:-}\ m,\ \neg b,n & & \\ c{:-}h, & & c{:-}h, \\ h{:-} & \Rightarrow\rightarrow_N & h{:-}, \\ b{:-}. & & b{:-}. \end{array}$$

**Definición 14: Reducción Negativa [8].**

Sean $P_1$ y $P_2$ programas. $P_2$ resulta de $P_1$ por reducción negativa ($P_1 \rightarrow$ .. $_N$ $P_2$) si y sólo si hay una regla $A \leftarrow$ .. B en $P_1$ y una literal negativa **not B** $\subset$ .. B tal que **B** $\subset$ .. hechos($P_1$), es decir, **B** aparece como un hecho en $P_1$ y

$P_2$=$P_1$ - {A $\rightarrow$ .. B}.

Podemos aplicar la transformación sin cambiar la semántica de $P_1$.

**Ejemplo 28:** Sea un programa ground *P*

*a*:- *m*,$\neg$ .. *b*,*n*

*c*:-*h*

*h*:-.

*b*:-.

De acuerdo a nuestra definición A={a}, B=*m*, $\neg$ .. *b*,*n* ; **not B** ={ $\neg$ .. *b*} donde {$\neg$ .. *b*} --> $\subset$ .. en $P_1$; tal que b $\subset$ .. hechos($P_1$); donde P1={ *a*:- *m*, $\neg$ .. *b*,*n*} entonces, $P_2$ $P_2$=$P_1$ - {A $\leftarrow$ .. B}. Si sustituimos nos queda $P_2$=0, Por lo tanto, directamente tenemos

.

$$\begin{array}{lcl} a{:-}\ m,\neg b,n & & \\ c{:-}h, & & c{:-}h, \\ h{:-} & \Rightarrow\rightarrow_{\mathrm{N}} & h{:-}, \\ b{:-}. & & b{:-}. \end{array}$$

Las proximas dos transformaciones describe la evaluación de literales positivas en el cuerpo en casos triviales.

**Definición 15: Exito(Success) [8].**

Un programa Ground (sin variables) $P_2$ resulta de un programa Ground $P_1$ por medio de éxito ( P1 $\rightarrow$ .. $_S$ P2) si y sólo si, hay una regla *A* $\leftarrow$ .. B y un hecho **B** en $P_1$ tales **B** $\subset$ .. B y

$P_2$=($P_1$-{A $\leftarrow$ .. B}) .. {A $\leftarrow$ .. ( B-{**B**})}

Podemos aplicar la transformación sin cambiar la semántica de $P_1$.

**Ejemplo 29:** Sea un programa ground P

*b*:-

*h*:-*c*,*b*.

De acuerdo a nuestra definición A={*h*}, B={*c*,*d*}; **B** ={*b*} donde *b* $\subset$ .. B en

$P_1$; donde P1={*h*:-*c*,*b*}, entonces, P2=(P1-{A $\leftarrow$ .. B}) $\cup$ .. {A $\leftarrow$ .. (B-{**B**})}. La primera parte de la unión se hace cero, y la segunda parte se resta { c,b} de {b}, por lo tanto queda: $P_2$={h: -c},directamente, resulta lo siguiente:

$$\begin{array}{l} b{:}- \\ h{:}-c,b. \end{array} \Rightarrow\rightarrow_S \begin{array}{l} b{:}- \\ h{:}-c. \end{array}$$

**Definición 16: Falla (Failure) [8].** Sea $\rightarrow$ .. $_F$ *(falla)* la transformación del programa definida por: ( $P_1 \rightarrow$ .. $_F$ $P_2$) si y sólo si hay una regla *A* $\leftarrow$ .. B en $P_1$ y una literal positiva **B** $\subset$ .. B tal que no hay una regla acerca de **B** en $P_1$, entonces

$P_2$=$P_1$-{A $\leftarrow$ .. B}.

Podemos aplicar la transformación sin cambiar la semántica de $P_1$ .

**Ejemplo 30:** Sea un programa ground P

*a*:- *m*,*b*,*n*

*h*:-*c*,

*a*:-*c*.

De acuerdo a nuestra definición A={a}, B={ *m*,*b*,*n*}; B={b} ,donde {b} $\subset$ .. B en $P_1$; donde $P_1$ ={*a*:- *m*,*b*,*n*} entonces, $P_2$ =P1-{A $\leftarrow$ .. B}. Si sustituimos nos queda $P_2$=0, Por lo tanto, directamente nos da:

.

$$\begin{array}{l} a{:}-\ m,b,n \\ h{:}-c, \\ a{:}-c. \end{array} \Rightarrow\rightarrow_F \begin{array}{l} h{:}-c, \\ a{:}-c. \end{array}$$

Las cuatro transformaciones definidas anteriormente, no pueden remover ciclos positivos, lo que hacen que sean débiles.

**Definición 17: Detección de ciclo [8].**

Sean $P_1$ y $P_2$ programas ground. $P_2$ resulta del programa $P_1$ por medio de detección de ciclo ( $P_1 \rightarrow$ .. $_L$ $P_2$) si y sólo si hay un conjunto **A** de átomos ground tales que:

1. Para cada regla **A** $\leftarrow$ .. **B** en $P_1$ si $A \subset$ .. **A**, entonces **B** $\cap$ .. **A** diferente de $\emptyset$,

2. $P_2$: = {$A \leftarrow$ ..**B** $\subset$ .. $P_1$ | **B** $\cap$ .. **A** diferente de $\emptyset$},

3. *$P_1$ diferente de $P_2$.*

Veremos un ejemplo, donde involucra Ciclo:

**Ejemplo 31:** Sea el programa Ground *P*,

*p*:- .

*q*:-*r*.

*r*:-*q*.

Se tiene que A={p,*q*,r} - {p} ,de acuerdo a nuestra definición, las dos últimas reglas son eliminadas porque sus literales del cuerpo positivas *q* y *r* están contenidas en **A**. Entonces el programa queda, $P_2$={p},
directamente da:

.

$$\begin{array}{l} p{:-}. \\ q{:-}r. \\ r{:-}q. \end{array} \Rightarrow\rightarrow_L \; p{:-}.$$

El conjunto **A**(visto en la definición de ciclo) es el conjunto Unfounded más grande[17] que consiste de todos los átomos ground positivos para los cuales es posible que no sean ciertos. Es decir, no se puede asumir que toda literal negativa es cierta.

Después de analizar los métodos de instanciación y de reducción de reglas a través de transformaciones básicas, y ver cada una de las semánticas no-monótonas, estamos listos para plantear el algoritmo que hará el cálculo de modelos (Stables) de las semánticas WFS, CWFS y STABLE.

**4.3 Algoritmo para el Cálculo de Modelos de WFS, CWFS, y STABLE**

En esta sección comentaremos algunos aspectos del **algoritmo final**, para el cálculo de modelos WFS, CWFS y Stables.

Utilizaremos programas lógicos normales (PLN), como primer paso, es instanciar este programa por medio de ground(P), si el programa lógico P no esta instanciado, debe emplearse algún algoritmo de instanciación

(vistos al menos dos en la sección anterior).

La salida del programa instanciado, es utilizado por un aplicación hecha en lenguaje C, que re-escribe el programa utilizando las transformaciones (red+, red-, success, failure y loop) básicas, y así obtener los modelos WFS. Si queremos obtener los modelos CWFS, se utiliza otra aplicación hecha(en un curso de maestría) en lenguaje C, que re-escribe el programa utilizando las transformaciones (red+, red-, success y loop) y completando el programa (completitud de Clark), y así dejarlo con salida para ser utilizado por OTTER (Forma Canónica o tradicional) [26] y por Forma No Canónica [25] (un programa hecho en C), que dejan el programa de salida en su forma clausular.

Se termina utilizando SATO [27], que es un programa obtenido a través de internet, que aplica el método de Davis-Putnam (satisfactibilidad) al programa que esta en su forma clausular, obteniendo los modelos finales o resultantes.

Para comprobación de los modelos STABLES obtenidos por SATO, se hace lo siguiente: una vez que se tienen los modelos finales, que son los "candidatos" a ser Modelos Stables se utiliza el método de Gelfond-Liftchitz, de SM [19], para reproducir los que serán al final, los modelos Stables.

Presentaremos el algoritmo para calcular modelos de WFS, CWFS y STABLE de una base de datos deductiva o un programa lógico normal *P*.

**Algoritmo 2:**

1. Calcular los modelos WFS.
    a. Ground(*P*) = P'.

    b. Reducciones con transformaciones básicas (red+, red-, suc., fai, loop).

2. Calcular los modelos CWFS.
    a. Completamos el programa utilizando completitud de Clark, obteniendo P''.

3. Calcular los modelos STABLES.

    a. i. Utilizamos forma Tradicional o canónica con OTTER = P''', y llegamos a su forma clausular. ii.Utilizamos FNC, para llegar a la forma clausular.

    b. Utilizamos SATO (satisfactibilidad) para obtener los modelos Finales (STABLES).

    c. Verificamos los modelos STABLES utilizando el método Gelfond-Lifschitz y reproducir los Modelos Stables.

Recordemos que para las semánticas WFS, CWFS y STABLE, se obtendrán modelos resultantes que algunas veces serán los modelos mínimos, los modelos resultantes, o el modelo intended que sería el modelo correcto o ideal que le corresponde al programa.

Consideremos el programa Lógico normal *P* :

*a*:-*not b*.

*b*:-*not a*.

*p*:-*not a*.

*p*:-*not p*, *not c*.

*c*:-*d*.

*d*:-*c*.

Del **algoritmo 2**, iniciamos con el paso 1 y sus respectivos incisos, al programa *P*, se le aplica el método de instanciación Ground del **algoritmo 1**, la idea de ground es generar y empezar con el conjunto de instanciaciones de reglas de *P* que no contengan literales en el cuerpo que sean obviamente falsas, Consideramos una regla instanciada solo si tenemos razones para creer que todas las literales positivas de su cuerpo tienen posibilidad de ser ciertas, Podemos empezar con las instancias de reglas sin literales positivas de su cuerpo, Las cabezas de estas reglas instanciadas son posiblemente ciertas, así que además las reglas instanciadas que tengan a estos átomos como literales positivos de sus cuerpos pueden ser derivados exitosamente, además, si ya sabemos que un átomo *A* es cierto, no debemos hacer caso de aquellas instancias de reglas que tienen *not A* en su cuerpo como literal negativa[8], al final obtenemos P'.

Haciendo el **Algoritmo 1** (Ground, Visto en la sección anterior), iterativamente me da como resultado:

El ground P':

*a*:-*not b*.

*b*:-*not a*.

*p*:-*not a*.

*p*:-*not p*.

Al programa resultante P', se le aplican las transformaciones básicas (red+, red-, Success y Loop), y obtenemos los modelos de la semántica WFS, el programa resultante de P' nos podría servir para verificar si su semántica es verdadera o falsa , continuamos con el paso 2, del **algoritmo**

**2**, de utilizar la completitud de Clark (CWFS, [29]) obtenemos un P'', que sería un programa reducido y completado, quedando de la siguiente forma:

*a* $\Leftrightarrow$ .. *(-b).*

*b* $\Leftrightarrow$ .. *(-a).*

*p* $\Leftrightarrow$ .. *(-a V -p).*

En este ejemplo, no hubo reducción, fue solo completitud.

Después de obtener P'', instanciado y reducido, continuamos con el paso 3, inciso a, paso (i) del **algoritmo 2**, que se encarga de obtener el programa resultante en su forma cláusular mediante su forma tradicional que son leyes de Morgan y leyes de la lógica clásica, del ejemplo anterior quedaría de la siguiente forma:

de *a* $\Leftrightarrow$ .. *(-b*) obtenemos lo siguiente:

*( a* $\leftarrow$ .. *b)* y *( -b* $\leftarrow$ .. *a)*, y nos da como resultado, (*b* V *a*) y( *-b*V *-a*),respectivamente,

de *b* $\Leftrightarrow$ .. (-*a*), obtenemos :

( *b* $\leftarrow$.. -*a*), y ( -*a* $\leftarrow$ .. *b*),dandonos (*b* V *a*) y ( -*a* V -*b*), respectivamente,

de la expresión, *p* $\Leftrightarrow$.. (-*a* V -*p*), tenemos:

*p* $\rightarrow$ .. (-*a* V -*p*) y (-*a* V -*p*) $\rightarrow$ .. *p*, lo cual da como resultado lo siguiente: -*p* V(-*a* V -*p*) y *(-(-a* V -*p))* V *p*, resolviendo por leyes de Morgan obtenemos:

(-*p* V-*a*) y *p*, respectivamente.

al final, obtenemos P''' que contiene las reglas siguientes:

(*a* V *b*).

(-*b* V -*a*).

(*b* V *a*).

(-*a* V -*b*).

(-*p* V -*a*).

*p*.

Lo cual nos da, un conjunto de clausulas.

En esta parte del **algoritmo 2**, punto 3, inciso a, paso (i) , la forma

cláusular también se puede obtener a través de su Forma No Canónica [25], por ejemplo, la fórmula ( *y1*∧ .. *y2*)V *y3*, donde V significa "*y*", y puede ser escrita como (*y1* V *y3*) ∧ .. (*y2* V *y3*). También la implicación *y 1* → .. *y 2* o ( *y1*⊃ .. *y2*) puede ser escrita como ¬ .. *y1* V *y2*, y la equivalencia *y1* identico *y2* se puede reescribir como (¬ .. *y1* V *y2*)∧ .. (*y1* V ¬ .. *y2*).

La Forma normal Conjuntiva es expresiva porque cualquier fórmula en variables bivalentes puede interpretarse como una función booleana (verdadera-falsa), *f*(*y*) = *f(y1,...,ym)*. Si *y* = $v^1,...,v^N$ son los valores de *y* que hacen *f*(*y*) falsa, entonces la siguiente fórmula es equivalente a *f*(*y*).

$$\bigwedge_{i=1}^{N} \bigvee_{j=1}^{m} y_j(v^i_j),$$

donde $y_j(v^i_j)$ es ¬ .. $y_j$ si ( $v^i_j$ )=verdadera y es $y_j$ si ( $v^i_j$ )= falsa.

Esta conversión a la Forma Normal Conjuntiva requiere tiempo y espacio exponencial en el peor de los casos. Sin embargo, una fórmula que involucra las conectivas (¬ .. , ∧ .. ¸V,→ .. , identidad) pueden ser convertidas a la Forma Normal Conjuntiva en tiempo lineal agregando nuevas variables [25]. El algoritmo le llamaremos (Forma No Canónica) FNC y es el siguiente, si una fórmula dada *F* tiene la forma A ∧ .. B, donde *A* y *B* son subfórmulas, entonces se aplica el algoritmo a *A* y a *B* separadamente, y se juntan los resultados.

Si *F* tiene la forma A→ .. B entonces hay que reescribirla como ¬ .. A V B y aplicar el algoritmo a la fórmula resultante y similarmente para A identica a B. Si *F* tiene la forma A V B, entonces se escribe como ($y_{m+1}$ identico A) ($y_{m+2}$ identico B) ($y_{m+1}$ V $y_{m+2}$), donde $y_{m+1}$, $y_{m+2}$ son variables que no ocurren en *F*, y se aplica el algoritmo a la fórmula resultantes. El algoritmo termina cuando se alcanza la FNC.

**Ejemplo 32:** tenemos el mismo ejemplo que utilizamos con la forma tradicional, que esta completado y reducido

a< >( b)

b < > ( a)

p < > (a V p)

es equivalente a:

( -y 1 -b)

( a b )

( y 1 b )

(-a y 1)

( -y2 -a)

( p a )

( y2 a )

( -y3 -p)

( p p )

( y3 p )

(-p y2 y3)

( -y4 -a)

( b a )

( y4 a )

(-b y4)

aplicando la Forma No Canónica**,** se comprobó que son equivalentes**.**

Una vez teniendo el programa P''' continuamos con el inciso b), del paso 3 del **algoritmo 2**, que por medio del método de Davis-Putnam (satisfactibilidad) llegamos al resultado final (vemos ejemplos en el **Apéndice 7.2 y Apéndice 7.3**):

un modelo que es : *b, p*.

que representa el número de modelos, donde cada modelo me representa una solución del programa lógico normal, que al final de cuentas serían los modelos que nos interesa obtener para efectos de comparación, con los modelos Stables obtenidos con S-models.

En este momento, del **algoritmo 2**, paso 3, inciso c), tenemos los modelos STABLES, estos modelos se verifican utilizando la definición de SM (método de Gelfond- Lifschitz) para así tener los probables Modelos Stables.

Existen otras herramientas en internet que también podemos hacer uso, para llegar a los modelos finales o resultados, que por nombrar algunos de ellos: Grasp, NTAB, Posit, Satz, Satz-rand, etc.

Cabe mencionar, que con la herramienta S-MODELS, se obtuvo un mo-delo, y ese modelo fue el mismo: *b, p.*

El siguiente diagrama de bloques, mostrado en la figuras, 4.1 y 4.2, muestran el funcionamiento general del programa que fue implementado en lenguaje C y utilizando herramientas para solución de problemas obtenidas en internet, para la obtención de modelos.

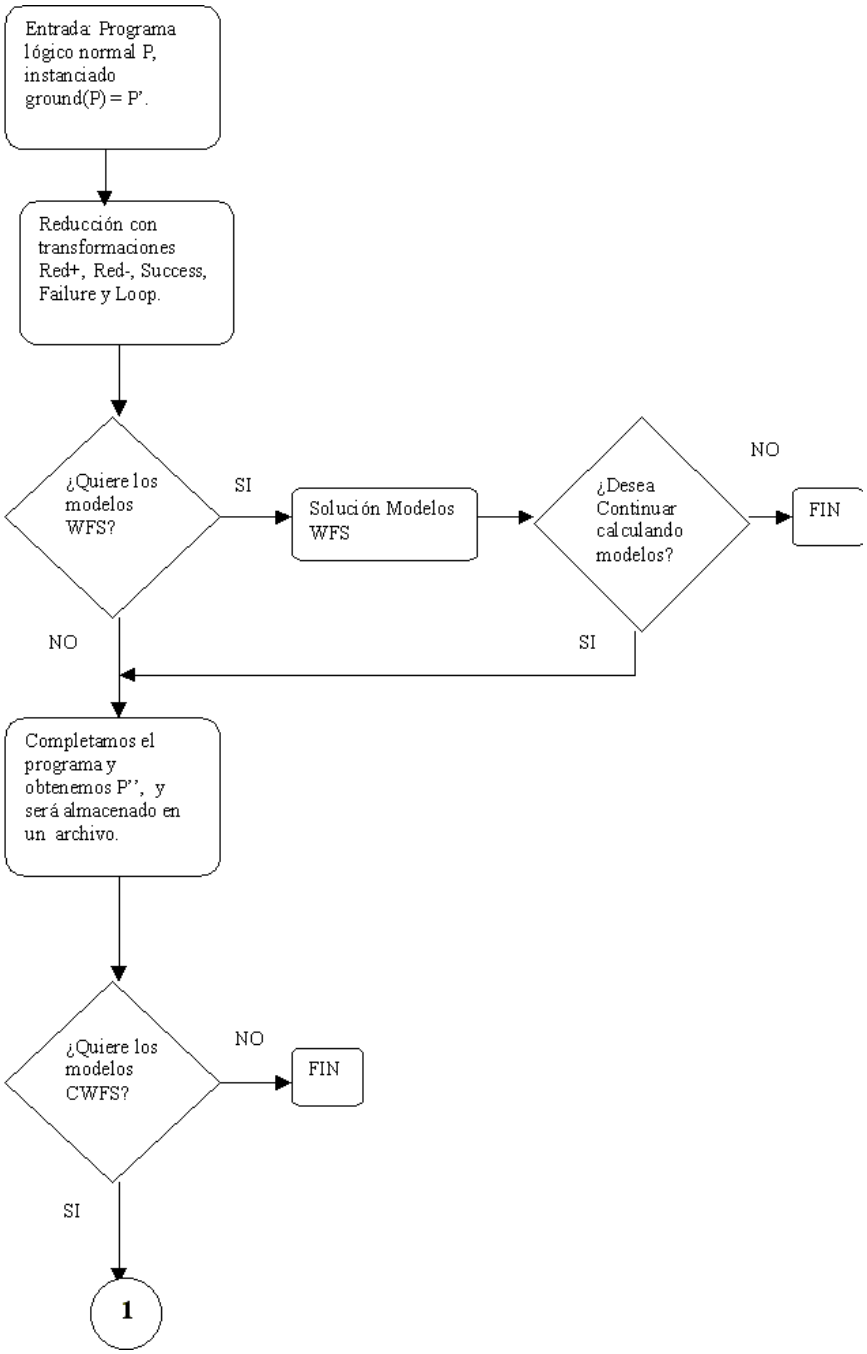


Figura 4.1: Diagrama de bloques final

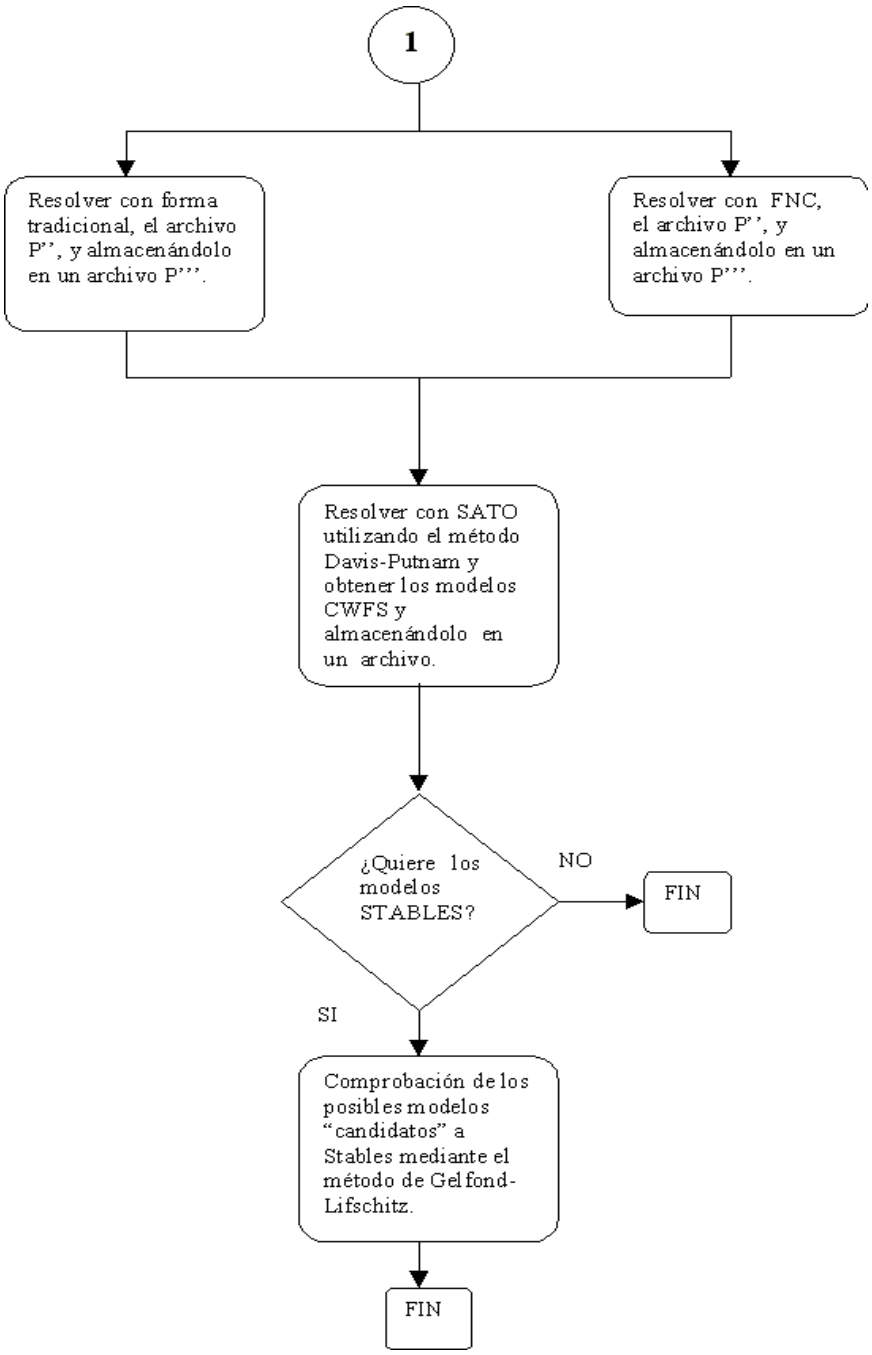


Figura 4.2: Continuación de diagrama de bloques final