Capítulo 4

La Herramienta Models_WFS

La herramienta Models_WFS[1] se desarrolló con el fin de poder encontrar un modelo por cada teoría otorgada, a partir de ciertas transformaciones. En este capítulo se pretende explicar tales transformaciones junto con un ejemplo. También, se describe el criterio con que se aplican las transformaciones en Models_WFS, junto con una ejemplificación de cuando OTTER itera prolongadamente, y en donde Models_WFS obtiene una respuesta en tiempo polinomial (más rápido que con OTTER).

Para información técnica acerca de cómo utilizar la herramienta Models_WFS favor de consultar el Apéndice B.

La herramienta Models_WFS se desarrolló con el fin de poder encontrar un modelo por cada teoría otorgada. En nuestro caso, particular se busca construir un modelo (tan pequeño como sea posible en un tiempo polinomial), con el riesgo de que pueda fallar en ciertos casos. Dicho modelo (si es que existe) se construye a partir de transformaciones que tienen la característica de ser relativamente veloces, y fáciles de mecanizar. Asimismo, el modelo que resulta es una extensión parcial de las semánticas bien fundamentadas WFS (del inglés Well-Funded Semantics).

[1] Constructor de Modelos extensión parcial de WFS

ntación para Models_WFS

La herramienta Models_WFS utiliza como lenguaje de representación del conocimiento Lógica de Primer Orden. Pero considerando un número infinito de estructuras en tal lenguaje, se utiliza con ciertas restricciones:

- La primera de ellas es que no consideramos símbolos funcionales, por ejemplo, si F es un nuevo símbolo funcional, podemos remplazar $\forall x \ \forall y \ \exists z \ P(x,y,z)$ con la sentencia $\forall x \ \forall y \ P(x,y,F(x,y))$. Sin embargo, no trabajeremos con este tipo de cláusulas.
- La segunda es que cada cláusula debe contener por lo menos una literal positiva. De lo contrario no es posible derivar nada con la aplicación del conjunto de transformaciones.

  Por ejemplo, las siguientes cláusulas no se consideran:

  -p(x) :- -q(y).

  -t(x,z) :- -g(x,y), -h(y,z).

4.2 Definiciones

Antes de explicar cada una de las transformaciones, daremos algunas definiciones con el fin de facilitar el manejo de terminología que utilizaremos de aquí en adelante.

Definición de Atomo 4.1

Un átomo es un objeto sintáctico que consiste de un nombre de predicado que puede tener o no argumentos. Por ejemplo a, P(a), P(x), Z(e,y), son átomos.

Definicion de Literal 4.2

Una literal es un átomo o la negación de un átomo. Por ejemplo P(a), ~P(x), Z(e,y), son literales.

Definicion de Regla 4.3

Una regla esta conformada por una cabeza y un cuerpo. La cabeza de una regla es siempre un átomo y el cuerpo de la misma es una disyunción finita de literales. Por ejemplo:

Arriba(a,d) :- Encima(a,b), Encima(b,c), Encima(c,d).

Es una regla, donde “Arriba(a,d)” es la cabeza y “Encima(a,b), Encima(b,c), Encima(c,d)” es una disyunción finita de literales, donde cada “,” (coma) representa un ”o” lógico.

Nota: Las disyunciones se representarán por medio de comas.

Definición de Transformación 4.4

Una transformación es una relación binaria entre programas lógicos.

Definición de Cláusula 4.5

Una cláusula es una disyunción de literales.

Definición de Teoría 4.6

Una teoría es una conjunción de cláusulas.

4.3 Transformaciones aplicadas a Models_WFS

Una vez definida la terminología, y considerando como ***P*** la teoría a probar y a ***P´*** lo que resulte al aplicar cualquier transformación, se da una descripción de lo que realiza cada transformación junto con un ejemplo. Para más información sobre este conjunto de transformaciones y respectiva demostración de las mismas consulte las referencias de [Zep97], [Arr98], [Arr98a] y [Oso98].

La herramienta Models_WFS se programó, con un conjunto de transformaciones que permiten obtener un modelo (una solución) a cada teoría, si es que existe. Estas transformaciones son:

- C
- Red-
- Llc
- Casos
- Suc
- Red+
- Failure
- Loop

Tales transformaciones fueron comprobadas a través de OTTER para verificar si lo que resulta de aplicar determinada transformación es lógicamente equivalente con la teoría original. Los resultados obtenidos se listan en la tabla 4.1, donde la primera columna indica el nombre de la Transformación, la segunda columna indica si ***P⇒P´ (P implica a P´)***, la tercera columna indica si ***P⇐P´ (P´ implica a P)***, y la última columna indica si ***P⇔P´*** (***P*** y ***P´*** son lógicamente equivalentes, pues se implican comúnmente).

Transformación	**P⇒P´ implicación**	**P⇐P´ implicación inversa**	**P⇔P´ doble implicación**
SUC	si	si	si
RED+	si	si	si
RED-	no	si	no
LOOP	si	si	si
CASOS	si	si	si
C	si	si	si
FAILURE	no	si	no
LLC	si	si	si

Tabla 4.1 Resultados obtenidos a través de OTTER para la comprobación de equivalencia lógica de transformaciones.

4.3.1 Transformación Transferencia (suc)

Si x es un hecho en ***P*** y encontramos alguna regla donde una literal positiva del cuerpo es x, borramos tal literal.

Por ejemplo, supongamos que tenemos lo siguiente en ***P***:

a
f : - a, b, ~c
d : - g

Al aplicarle **Transferencia (suc)** considerando ***a***, resultaría en ***P´*** lo siguiente:

a
f : - b, ~c
d : - g

4.3.2 Transformación RED+

Si x es un hecho en ***P*** y aparece como literal negada en el cuerpo de alguna regla, simplemente borramos la regla.

Por ejemplo, supongamos que en ***P*** tenemos:

c

f : - a, b,~c

d : - g

Al aplicarle **RED+** con ***c***, en ***P´*** resultaría lo siguiente:

c

d : - g

4.3.3 Transformación RED-

Si x$\not\in$ HEAD(P), pero x ocurre como literal negada en el cuerpo de una regla, entonces borramos ~x de dicha regla.

Por ejemplo, supongamos que tenemos en ***P***:

f : - a, b,~c
d : - g

Al aplicarle **RED-** con ***~c***, resultaría en ***P´*** lo siguiente:

f : - a, b
d : - g

4.3.4 Transformación Failure

Si x$\notin$ HEAD(P), pero x ocurre como literal positiva en el cuerpo de una regla, entonces borramos dicha regla.

Por ejemplo, supongamos que tenemos en ***P***:

f : - a, b
d : - f

Al aplicarle **Failure** con ***b***, resultaría en ***P´*** lo siguiente:

d : - f

4.3.5 Transformación LOOP

Sea ***P´´*** como P pero borramos todas las literales negadas

Sea ***P´´´*** el programa que resulta de reducir completamente ***P´´*** por medio de la operación S.

Sea A: = átomos en ***P***

Sea B: = hechos (***P´´´***)

Sea C: = A \ B

Sea ***P´*** el programa que resulta de ***P*** al borrar toda regla tal que x sea una literal positiva de su cuerpo y x pertenezca C (x∈ C).

Por ejemplo, supongamos que tenemos como ***P*** lo siguiente:

h.
a : - ~b
c : - a, h
d : - e
e : - d
m : - c

Por lo que en ***P´´*** tendríamos:

h
a
c : - a, h
d : - e
e : - d
m : - c

Así en ***P ´´´ :***

h
a
c
d : - e
e : - d
m

Sea A : = átomos en ***P :***

A= { **h, a, c, d, e, m** }

Sea B : = hechos ***P´´´:***

B= { **h, a, c, m** }

Sea C : = A \ B**:**

C= { **d, e** }

Por lo que ***P´*** quedaría como*:*

h
a : - ~b
c : - a, h
m : - c

4.3.6 Transformación Casos

La transformación Casos tiene dos formas:

1) Sustituye

2) Agrega

Por ejemplo, supongamos que tenemos como ***P*** lo siguiente:

d : - ~c
d : - c

Al aplicarle **CASOS**, resultaría lo siguiente para 1) en ***P´***:

d

Y para 2) en ***P´***:

d : - ~c
d : - c
d

Nota: Cabe hacer mención que en Models_WFS se utiliza la opción 1) Sustituir.

4.3.7 Transformación C

Si se encuentra en el cuerpo de una regla una litera1 tanto en forma negativa como positiva, entonces borramos la regla en ***P***, quedando ***P´*** sin esta regla.

Por ejemplo, supongamos que tenemos lo siguiente:

a : - ...., ...,b,...., ~b
f : - e, g

Al aplicarle **C**, resultaría en ***P´*** lo siguiente:

f : - e, g

4.3.8 Transformación LLC

Sea:

a una letra cualquiera en P.

P´´: = ***P*** U {~a}

P´´ en ***P´´´*** mediante la aplicación de S^E hasta que ya no se pueda

1) ***a*** no es un hecho en ***P´´*** entonces la regla no se aplica
2) ***a*** es un hecho en ***P´´´*** entonces ***P´*** : = ***P*** U {~a}, es decir se agrega el hecho a.

Explicación de S^E: Es como S pero también elimina ~a del cuerpo de una regla si ~a es hecho.

Por ejemplo, supongamos que tenemos como ***P*** lo siguiente:

b : - ~a
c : - b
a : - c

Elegimos ***a,*** por lo que tendríamos como ***P´´*** lo siguiente***:***

b : - ~a
c : - b
a : - c
~a

Y como ***P´´´:***

b
c
a
~a

Y como ***a*** es un hecho en ***P´´´*** se agrega a ***P´*** el hecho, resultando lo siguiente:

b : - ~a
c : - b
a : - c
a

Lema 4.3:

Lo robusto de las transformaciones anteriores es que si consideramos a ***L*** como el conjunto de literales que no aparecen como cabeza en ***P´*** y ***~L*** representa el conjunto de literales negadas de ***L***, entonces ***P*** es consecuencia lógica de ***P´ U ~L´.***

4.4 Criterio de aplicación de Transformaciones

Para decidir el criterio de aplicación de las transformaciones, se realizó una prueba con 10 programas y cuatro diferentes criterios (con respecto al orden de aplicación de transformaciones):

1er. Criterio	2do. Criterio	3er. Criterio	4to. Criterio
C	C	CASOS	RED+
RED-	RED-	RED-	CASOS
LLC	RED+	C	LLC
CASOS	CASOS	SUC	C
SUC	LLC	RED+	RED-
RED+	SUC	LLC	SUC
FAILURE	FAILURE	FAILURE	FAILURE
LOOP	LOOP	LOOP	LOOP

A continuación se listan de menor a mayor velocidad los resultados de los criterios:

- Criterio3
- Criterio2
- Criterio4
- Criterio1

El Criterio1 es con el que se consigue mayor velocidad y por lo tanto es el que consigue menor tiempo de ejecución.

4.5 Mejor Criterio de Aplicación de Transformaciones

Se eligió el Criterio1, el cual da respuesta en menor tiempo y a continuación se describe:

- Primero aplicamos la Transformación C, para eliminar las reglas que contengan en el cuerpo la misma literal tanto en forma positiva como negativa.
- Después aplicamos RED-, con el fin de eliminar a las literales negadas que aparezcan en el cuerpo de cada regla y que no aparezcan como cabezas.
- Después aplicamos LLC, con el fin de agregar un hecho.
- Luego aplicamos CASOS con el fin de reducir dos reglas por un hecho.
- Aplicamos SUC, con la finalidad de generar mas hechos.
- Después aplicamos RED+, para borrar todas las cláusulas que contengan dentro de su cuerpo una literal igual a un hecho en forma negada.
- Posteriormente aplicamos FAILURE con el propósito de eliminar reglas.
- Por último aplicamos LOOP, con el fin de eliminar más reglas, siendo ésta la última que aplicamos por ser una de las que más costosas (computacionalmente hablando).

Este conjunto de transformaciones se aplica iterativamente hasta que ya no se puede aplicar ninguna de ellas. Cuando ya no se puede aplicar ninguna transformación, entonces se busca el átomo que aparece más veces negado dentro del cuerpo de cada una de las reglas y entonces dicho átomo es agregado al final del conjunto de reglas, y se vuelven a aplicar el conjunto de transformaciones antes mencionado hasta que nuevamente ya no se pueda aplicar ninguna de ellas. Nuevamente se busca el átomo que aparece más veces negado y se aumenta como hecho al final del conjunto de transformaciones, repitiéndose el proceso hasta que ya no se pueda aplicar ninguna transformación y ya no quede ningún átomo negado dentro del conjunto de reglas.

4.5 Ejemplificación de Models_WFS

En el **Ejemplo 3.5.3** se presento un problema resuelto a través de OTTER, tal ejemplo se encuentra en [Gen88]. A continuación se retoma para visualizar cómo Models_WFS podría ayudar a OTTER a evitar la iteración prolongada (que presenta en algunas ocasiones) cuando se le introduce una teoría a probar que no es demostrable.
Por último, se explica y muestra como funciona Models_WFS.

Ejemplo 4.5.1

Conocemos que los caballos son más rápidos que los perros y que hay un galgo que es más rápido que cada conejo. Conocemos que Harry es un caballo y que Ralph es un conejo. Lo que se va a pretender derivar es el hecho que Ralph es más rápido Harry.

Necesitamos primero formalizar nuestras premisas. Las sentencias relevantes son las siguientes:

$\forall x \forall y$ Caballo(x) $\wedge$ Perro(y) $\Rightarrow$ Mas_Rapido(x,y)
$\exists y$ Galgo(y) $\wedge$ ($\forall z$ Conejo(z) $\Rightarrow$ Mas_Rápido(y,z))
$\forall y$ Galgo(y) $\Rightarrow$ Perro(y)
$\forall x \forall y \forall z$ Mas_Rápido(x,y) $\wedge$ Mas_Rápido(y,z) $\Rightarrow$ Mas_Rápido(x,z)
Caballo(Harry)
Conejo(Ralph)

Note que estamos adicionando dos hechos acerca del mundo, no establecidos explícitamente en el problema: que los galgos son perros y que nuestra velocidad de relación es transitiva. Lo que ahora se intenta probar es:

Mas_Rápido(Ralph, Harry).

Lo cual resulta ser obviamente falso. Sin embargo, esto lo concretaremos más adelante concretaremos

4.6 Solución usando Models_WFS

Usando Models_WFS se construye un modelo en el que no se encuentra posibilidad alguna de que tal cláusula sea cierta (sin iteración prolongada). Por lo que a continuación vamos a ejemplificar este problema con Models_WFS.

OTTER para probar determinada teoría, requiere que ésta se encuentre en forma clausular (como se explico en la sección 2.1.2), por lo que realiza tal conversión y la escribe dentro del archivo de salida. Para efecto del archivo de entrada a Models_WFS se utiliza esta parte del archivo de salida que genera OTTER:

```
-caballo(x)| -perro(y)|mas_rapido(x,y).
galgo($c1).
-conejo(z)|mas_rapido($c1,z).
-galgo(y)|perro(y).
-mas_rapido(x,y)| -mas_rapido(y,z)|mas_rapido(x,z).
caballo(Harry).
conejo(Ralph).
```

Lo anterior se especifica en la forma normal, resultando el siguiente programa:

```
mas_rapido(X,Y):-caballo(X), perro(Y).
galgo(c1).
mas_rapido(c1,X):-conejo(X).
perro(X):-galgo(X).
mas_rapido(X,Z):-mas_rapido(X,Y), mas_rapido(Y,Z).
caballo(harry).
conejo(ralph).
```

Posteriormente debe hacerse la construcción de la ***instanciación ground***, lo que significa que en el programa no deben aparecer variables. Esto se logra sustituyendo cada una de las variables en las reglas del programa por las constantes que también forman parte de la misma teoría.

Tal instanciación la realizamos por medio de una herramienta de software que se denomina ***ground***. Actualmente ya se cuentan con versiones más completas de tal

herramienta, denominadas con diferentes versiones de ***lparse***, para más información sobre estas herramientas consulte la referencia [Http3].

Una vez realizada tal instanciación, ésta se aplica como archivo de entrada al programa Models_WFS resultando:

```
 El tiempo es Mon May  1 17:11:06.295 2000

 El Modelo (extención parcial WFS) es
caballo(harry).
galgo(c1).
conejo(ralph).
mas_rapido(c1,ralph).
perro(c1).
mas_rapido(harry,c1).
mas_rapido(harry,ralph).

 El tiempo es Mon May  1 17:11:06.302 2000
```

Donde se obtiene el modelo que la herramienta Models_WFS construyó, tal modelo conforma una explicación coherente de la teoría de entrada. Si existe la posibilidad de que las conclusiones sean ciertas no es posible que tales átomos resulten ser falsos (no demostrables) y viceversa con los átomos que no existen dentro del modelo.

Con Models_WFS encontramos un ***modelo*** donde `mas_rapido(ralph,harry)` es falso (no existe dentro del modelo) por lo que no es demostrable. Dicho ***modelo,*** Models_WFS lo construyó en sólo 7 milésimas de segundo, y con OTTER cuando le introducimos este problema itero prolongadamente durante 5 horas sin dar respuesta alguna, por lo que se abortó el programa ya que se consideró que ya había tardado demasiado y que no iba a terminar.