Capítulo 3

Razonamiento Automático en OTTER

En este capítulo presentaremos una breve descripción de lo que es OTTER y su importancia. También, se presenta información sobre su estrategia de razonamiento, arquitectura y su proceso de inferencia, junto con algunos ejemplos de cómo trabaja. Cabe aclarar que no se pretende dar información muy detallada, ni un tutorial sobre el manejo de esta herramienta. Para conocer a fondo todas las funciones del programa consulte la referencia [McC94] y para encontrar un tutorial básico véase la referencia [Wos92]. Si el lector está interesado en un tutorial avanzado es recomendable la referencia [Wos96].

3.1 ¿Qué es OTTER?

OTTER es un demostrador automático que implementa el método de resolución para lógica de primer orden con igualdad. Fue desarrollado en el Laboratorio Nacional de Argonne, USA. Las siglas ”OTTER” significan Organized Techniques for Theorem-proving and Effective Research. Las condiciones primarias para su diseño fueron el desarrollo, la portabilidad y la extensibilidad [Amo98]. Para mas información consulte las referencias [McC94], [Amo98] y [Mir99].

3.2 Importancia de OTTER

OTTER tiene una efectividad extraordinaria, y ha resuelto problemas abiertos en diversas áreas como Teoría de grupos, Geometría Algebraica, Algebras Booleanas, Teoría de Retículos y Lógica Combinatoria [Amo98]. Con OTTER se han logrado obtener resultados que no se habían podido demostrar por métodos tradicionales.

Además, en lo que se refiere a concursos de razonamiento automático OTTER ha destacado entre los primeros lugares (fue ganador de la división ecuacional en CASC-13).

3.3 Aplicaciones relevantes

Se describen dos aplicaciones que consideramos relevantes debido a que han sido estudiadas desde hace muchos años y no se habían podido demostrar, y con OTTER es posible su demostración.

3.3.1 El problema de Robbins.

El **problema de Robbins**: "¿Todas las álgebras de Robbins son Booleanas?", ha sido resuelto: cada álgebra de Robbins es Booleana. Este problema se ha estudiado desde 1930, y hasta el año de 1996 se logró demostrar utilizando EQP (**Eq**uational **P**rover, un programa de demostración automática, muy similar a OTTER). La búsqueda exitosa tomó cerca de 8 días sobre un procesador RS/6000 y usó cerca de 20 megabytes de memoria.

Considerando que las pruebas encontradas por programas son siempre cuestionables, es importante contar con un demostrador automático que construya un objeto de prueba detallado, junto con un programa simple que controle que el objeto de prueba resulte ser correcto.

EQP aún no es capaz de construir objetos de prueba, así que la prueba de EQP fue utilizada para guiar a OTTER a una prueba del mismo teorema. OTTER produjo un objeto de prueba, el cual fue controlado a través de un inspector de prueba.

El archivo de entrada, la prueba, el objeto prueba e información adicional (con respecto a la demostración de OTTER) se encuentra disponible en la referencia [Http1].

3.3.2 Nuevas Resultados obtenidos a través de OTTER

En el Laboratorio Argonne, de Estados Unidos se han obtenido nuevos resultados en el área de razonamiento automático. La principal contribución de estos programas es que ayudan a pruebas manuales, encontrando una prueba sin guía del usuario. En la página [Http2] se encuentra un resumen de tales resultados, de los cuales varios fueron obtenidos a través de OTTER.

3.4 Estrategia de razonamiento en OTTER

Para que un programa de razonamiento automático sea efectivo, se requiere del uso de estrategias. Una estrategia es una serie de metarreglas que guían la aplicación de las reglas de inferencia. Si una estrategia decide que regla de inferencia debe aplicarse en cierto momento del proceso, decimos que es una *estrategia de dirección*. Si una estrategia evita el uso de ciertas reglas de inferencia entonces decimos que es una *estrategia de*

restricción. Ambos tipos de estrategia son necesarios para la efectividad de un programa de razonamiento automático [Mir99].

Entre las estrategias de restricción, una de las más poderosas es la estrategia de soporte, la cual le prohibe a un programa de razonamiento automático aplicar una regla de inferencia a menos que uno de los padres o premisas se haya deducido o sea miembro de un conjunto distinguido de cláusulas de entrada.

Por otra parte, una de las estrategias más poderosas de dirección (que son las que dictan hacia donde dirigir la atención) es la estrategia de peso. Un programa de razonamiento automático que emplea pesos elige la cláusula con el peso más favorable; en el caso de OTTER, se prefieren cláusulas más ligeras, es decir, cláusulas con menor peso.

OTTER también ofrece una técnica efectiva para búsqueda de pruebas cortas conocida como la estrategia de la razón, que fue formulada para escoger que cláusula debe ser la siguiente en procesar. Aunque el peso es de gran utilidad para guiar la búsqueda de un programa, su uso retrasa y hasta evita el procesamiento de cláusulas muy pesadas.

Una manera para seleccionar con anticipación cláusulas pesadas es utilizar la búsqueda a lo ancho. Sin embargo, una búsqueda a lo ancho obliga al programa a escoger en cada nivel un número mayor de cláusulas. Para combinar las dos estrategias tenemos la estrategia de la razón, que dado un entero k causa que el programa elija para procesar k cláusulas por peso y después una por cada nivel.

El estudio de nuevas y mejores estrategias es de gran importancia y es uno de los principales temas de investigación actualmente [Mir99].

3.5 Arquitectura de OTTER

OTTER está escrito en lenguaje de programación C, el cual proporciona velocidad y portabilidad. Consta aproximadamente de 35,000 líneas de código (incluyendo comentarios). Está diseñado para ejecutarse en ambiente UNIX aunque existen versiones limitadas que se pueden ejecutar bajo los sistemas operativos DOS o Macintosh.

OTTER lee un archivo de entrada que contiene un conjunto de fórmulas o cláusulas de lógica de primer orden con igualdad, dividido en listas y alguna información extra para controlar la ejecución. El conjunto de fórmulas incluye las hipótesis y la negación de la conclusión (todas las pruebas son por contradicción); la información de control consiste de diversos parámetros y banderas que especifican las reglas de inferencia que se utilizarán y las estrategias de búsqueda. Mientras OTTER busca, va escribiendo la información obtenida a un archivo de salida, incluyendo la refutación buscada si ésta ocurrió. Las fórmulas se construyen con los símbolos listados en la tabla 3.1.

Interpretación	**Símbolo**
Negación	-
Conjunción	&
Disyunción	|
Implicación	–>
Equivalencia	<–>
Cuantificación existencial	exists
Cuantificación universal	all
Igualdad	=
Desigualdad	!=

Tabla 3.1 Símbolos con que se construyen las fórmulas en OTTER

Las reglas de inferencia que OTTER conoce son: resolución binaria, hiperresolución, hiperresolución negativa y UR-resolución. Mediante el comando `set(r)` se le indica al programa que regla de inferencia usar y mediante el comando `clear(r)` se le indica que regla dejar de usar. Aquí `r` puede ser simbolizada por: `bynary_res`, `hyper_res`, `neg_hyper_res`, `ur_res`, `para_into`, `para_from` y `demod_info`, para habilitar la bandera de resolución binaria, hiperresolución, hiperresolución negativa, resolución–UR, paramodulación o demodulación, respectivamente.

3.6 El proceso de Inferencia de OTTER

OTTER trabaja con cláusulas únicamente pero es posible proporcionarle fórmulas cualesquiera como entrada. Estas fórmulas serán transformadas automáticamente a forma clausular. El mecanismo básico de inferencia de OTTER es el algoritmo de la cláusula dada que se puede ver como una implementación de la estrategia del conjunto de soporte. OTTER mantiene cuatro listas de cláusulas que se encuentran en la tabla 3.2.

Tipo de Lista de Cláusulas	**Descripción**
`usable`	En la lista están disponibles para hacer inferencia.
`sos`	Es la lista del conjunto de soporte (set of support), las cláusulas de esta lista no están disponibles para hacer inferencias por sí solas; están esperando para participar en búsqueda de una refutación.
`passive`	Las cláusulas de esta lista no participan directamente en la búsqueda; se usan únicamente para subsunción y conflicto de unidades. Esta lista está fija desde el inicio y no cambia durante el proceso.
`demulators`	Aquí están los demuladores que se usan para reescribir las cláusulas inferidas.

Tabla 3.2 Lista de cláusulas que mantiene OTTER

Algunas de estas listas pueden estar vacías. El ciclo de inferencia principal es el algoritmo de la cláusula dada que opera en las listas `sos` y `usable`.

```
While sos != {}
     Sea Cd la cláusula mas ligera de sos;
     sos := sos - {Cd};
     usable := usable ∪ {Cd};
     Infiere (Cd);
End While
```

Donde el proceso `Infiere(Cd)` genera nuevas cláusulas utilizando las reglas de inferencia en acción, con la condición de que cada cláusula nueva debe tener a `Cd` como uno de sus padres y elementos de la lista `usable` como sus padres restantes. Además las nuevas cláusulas se someten a pruebas de retención y aquéllas que sobreviven se agregan a la lista `sos`.

3.7 Ejemplificación de OTTER

A continuación se muestra, una ejemplificación básica de como funciona la herramienta OTTER. Para información sobre cómo utilizarla consultar el Apéndice A.

Primeramente se da una redacción de cada ejemplo a demostrar, después se traducen los enunciados de cada prueba a su forma clausular, para después introducirlos dentro del archivo de entrada de OTTER, y por último se muestra la parte de prueba del archivo de salida que infiere OTTER.

Ejemplo 3.1 [**Cha87**]

Supongamos que los precios bajan si la tasa de interés sube. Suponga también que la mayoría de la gente no es feliz cuando los precios bajan. Asuma que la tasa de interés sube. Muestre que se puede concluir que la mayoría de la gente no es feliz.

Permítanos denotar los enunciados mencionados como sigue:

P = Sube la tasa de interés.

S = Los precios bajan.

U = La mayoría de la gente no es feliz.

Hay cuatro enunciados en este ejemplo, los cuales son:

(1) Si la tasa de interés sube, los precios bajan.

(2) Si los precios bajan, la mayoría de la gente no es feliz.

(3) La tasa de interés sube.

(4) La mayoría de la gente no es feliz.

Ahora si nosotros queremos demostrar que (4) es verdadera si (1)$\wedge$(2)$\wedge$(3) son verdaderas.

Por lo que formalizamos nuestras premisas y establecemos las sentencias relevantes:

1. $P \Rightarrow S$
2. $S \Rightarrow U$
3. P

Necesitamos derivar:

4. U

Por lo que se requiere realizar un archivo de tipo texto con lo siguiente:

```
set(binary_res).
set(process_input).
clear(print_kept).
clear(print_back_sub).
assign(max_mem, 1500). % 1.5 Megabytes
formula_list(sos).
% problem
P -> S.
S -> U.
P.
% goal
-U.
end_of_list.
```

Dentro del archivo de salida contiene la siguiente demostración:

```
---------------- PROOF ----------------
```

```
1 [] -P|S.
2 [] -S|U.
3 [] P.
4 [] -U.
5 [binary,1.1,3.1] S.
6 [binary,2.1,5.1] U.
7 [binary,6.1,4.1] $F.
------------ end of proof -------------
```

Por lo que podemos concluir que la mayoría de la gente no es feliz.

Ejemplo 3.2 Problema de Síntesis Química [**Cha87**]

Supongamos que podemos ejecutar las siguientes reacciones químicas:

- $MgO + H_2 \Rightarrow Mg + H_2O$
- $C + O_2 \Rightarrow CO_2$.
- $CO_2 + H_2O \Rightarrow H_2CO_3$

Suponga que tenemos algunas cantidades de MgO, H_2, O2 y C. Muestre que podemos hacer H_2CO_3.

Para este problema, podemos considerar MGO, H2, O2 y C como fórmulas atómicas. Luego las reacciones químicas de arriba pueden ser representadas por las siguientes fórmulas:

A1: $(MGO \wedge H2) \Rightarrow (MG \wedge H2O)$
A2: $(C \wedge O2) \Rightarrow CO2$
A3: $(CO2 \wedge H2O) \Rightarrow H2CO3$

Dado que tenemos MGO, H2, O2 y C, estos factores pueden ser representados por las siguientes fórmulas:

A4: MGO
A5: H2
A6: O2
A7: C

Ahora el problema puede ser visto de la siguiente manera: ¿cómo probar que H2CO3 es una consecuencia lógica de A1,....,A7?

Para lo que se requiere realizar un archivo de tipo texto con lo siguiente:

```
set(binary_res).
set(process_input).
```

```
clear(print_kept).
clear(print_back_sub).
assign(max_mem, 1500). % 1.5 Megabytes
formula_list(sos).
% problem
(MGO & H2) -> (MG & H2O).
(C & O2) -> CO2.
(CO2 & H2O) -> H2CO3.
MGO.
H2.
O2.
C.
% goal
-H2CO3.
end_of_list.
```

Dentro del archivo de salida se encuentra la siguiente demostración:

```
---------------- PROOF ----------------
2 [] -MGO| -H2|H2O.
3 [] -C| -O2|CO2.
4 [] -CO2| -H2O|H2CO3.
5 [] MGO.
6 [] H2.
7 [] O2.
8 [] C.
9 [] -H2CO3.
11 [binary,2.1,5.1,unit_del,6] H2O.
12 [binary,3.1,8.1,unit_del,7] CO2.
13 [binary,4.1,12.1,unit_del,11,9] $F.
------------ end of proof -------------
```

Por lo que podemos concluir que. H_2CO_3 es una consecuencia lógica de A1,....,A7.

Ejemplo 3.3 [**Gen88**]

Consideremos el siguiente problema. 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. Nuestro trabajo va a ser demostrar el hecho que Harry es más rápido que Ralph.

Primero necesitamos formalizar nuestras premisas. Las sentencias relevantes son las siguientes:

$\forall$ x $\forall$ y Caballo(x) & Perro(y) $\Rightarrow$ Mas_Rápido(x,y)

$\forall$ y Galgo(y) & ($\forall$ 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) & 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. Y que, lo que necesitamos derivar es lo siguiente:

Mas_Rápido(Harry, Ralph)

Por lo que tenemos que realizar una archivo de tipo texto con lo siguiente:

```
set(binary_res).
set(process_input).
clear(print_kept).
clear(print_back_sub).
assign(max_mem, 1500). % 1.5 Megabytes
formula_list(sos).
% problem
all x all y (caballo(x) & perro(y) -> mas_rapido(x,y)).
exists y (galgo(y) & (all z (conejo(z) ->
mas_rapido(y,z)))).
all y (galgo(y) -> perro(y)).
all x all y all z ( (mas_rapido(x,y) & mas_rapido(y,z) ) -
> mas_rapido(x,z) ).
caballo(Harry).
conejo(Ralph).
% goal
-mas_rapido(Harry,Ralph).
end_of_list.
```

El cual al indicarlo como archivo de entrada en OTTER obtenemos la prueba del teorema en el archivo de salida, el cual contiene entre otras cosas la derivación de cómo el teorema fue demostrado, tal como se muestra a continuación:

```
---------------- PROOF ----------------
1 [] -caballo(x)| -perro(y)|mas_rapido(x,y).
2 [] galgo($c1).
3 [] -conejo(x)|mas_rapido($c1,x).
4 [] -galgo(x)|perro(x).
5 [] -mas_rapido(x,y)| -mas_rapido(y,z)|mas_rapido(x,z).
```

```
6 [] caballo(Harry).
7 [] conejo(Ralph).
8 [] -mas_rapido(Harry,Ralph).
9 [binary,4.1,2.1] perro($c1).
10 [binary,3.1,7.1] mas_rapido($c1,Ralph).
11 [binary,1.1,6.1] -perro(x)|mas_rapido(Harry,x).
16 [binary,11.1,9.1] mas_rapido(Harry,$c1).
19 [binary,5.1,16.1] -
mas_rapido($c1,x)|mas_rapido(Harry,x).
36 [binary,19.1,10.1] mas_rapido(Harry,Ralph).
37 [binary,36.1,8.1] $F.
------------ end of proof -------------
```

Por lo que podemos concluir que Harry es más rápido que Ralph.