Capítulo 1

Razonamiento automático

J. A. Alonso Jiménez
A. Fernández Margarit
M. J. Pérez Jiménez

1.1 Introducción

En el presente trabajo hacemos un resumen histórico del razonamiento automático y presentamos mediante un sistema de razonamiento automático distintas reglas, estrategias y aplicaciones.

1.2 Breve historia del razonamiento automático

Hacia finales del siglo XVII, Leibniz formula dos cuestiones que sintetizan viejas aspiraciones del hombre:

- la necesidad de disponer de un lenguaje universal (*lingua characteristica*) en el que poder expresar cualquier idea; y
- la posibilidad de mecanizar cualquier tipo de razonamiento (*calculus ratiocinator*).

Con el nacimiento de la Lógica Matemática moderna, en 1847, con los trabajos de G. Boole y A. De Morgan, aparece un primer "rational calculus", en el sentido de Leibniz. Posteriormente, en 1869, S. Jevons construye una máquina para verificar identidades booleanas.

La publicación en 1879 de los *Begriffsschrift* por parte de G. Frege, marca el inicio de la formalización de la Lógica. Dicho texto se considera precursor no sólo de los sistemas actuales de Lógica Matemática sino también de todos los lenguajes formales. Paradójicamente, a pesar de su importancia, estos trabajos pasaron desapercibidos.

Independientemente, G. Peano publica en 1889 sus *Arithmetices Principia* en el que utiliza una nueva notación simbólica para formalizar la teoría de números que R. Dedekind presentó el año anterior. La Aritmética de Peano (**PA**) es una teoría de primer orden que recoge las propiedades de los números naturales.

A principios del presente siglo, D. Hilbert propone usar las ideas de Frege y Peano para fundamentar las Matemáticas. En 1910, B. Russell y A. Whitehead publican el primer volumen de sus *Principia Mathematica* en la línea propuesta por Hilbert.

La primera cuestión formulada por Leibniz, la necesidad de construir un lenguaje universal, empezaba a tener respuestas parcialmente satisfactorias con los lenguajes formales. Hacia finales de los cincuenta, con la irrupción de los computadores electrónicos y con el soporte de los muchos logros obtenidos en el campo de la Lógica Matemática, se retoma con fuerza el segundo objetivo de Leibniz: la automatización del razonamiento, que se convierte en uno de los problemas iniciales de la Inteligencia Artificial.

En 1957, A. Newell, J.C. Shaw y H.A. Simon presentan su "máquina lógica" que consiste en un programa de computador capaz de probar 38 teoremas de la Lógica proposicional de los Principia Mathematica (tres años antes, M. Davis había escrito un programa para demostrar teoremas de la Aritmética de Presburger que no fue publicado en su momento). En 1960, Hao Wang con sus programas consiguió demostrar 9 capítulos de los Principia Mathematica en 9 minutos, incluyendo teoremas del cálculo de predicados usando el sistema deductivo de Gentzen.

En el mismo año, Gilmore presenta otra forma de tratar el problema de la deducción en la Lógica de primer orden basándose en un resultado de Skolem y Herbrand que reduce la demostración de una fórmula de primer orden, F, a la de una fórmula proposicional, F', asociada a ella. En esta reducción aparecen dos problemas: (1) La búsqueda de un "candidato" proposicional, F', y (2) la demostración de F'.

El artículo "*A computing procedure for quantification theory*", publicado por M. Davis y H. Putnam en 1960, supuso un avance cualitativo en la solución del problema (2) al introducir el concepto de cláusula y reducir la demostración de primer orden al estudio de la consistencia de conjuntos finitos de cláusulas proposicionales, proporcionando reglas que simplifican dicho estudio.

Los intentos de obtener una solución satisfactoria al problema (1) hacen emerger el concepto de unificación, que subyace en los trabajos de Prawitz, y culmina en el trabajo "*A machine oriented logic based on the resolution principle*" de J.A. Robinson publicado en 1965. En el mismo, presenta el principio de resolución en el que se combina la regla de corte proposicional y la unificación. Este trabajo supuso un hito en la demostración automática de teoremas y, en cierto modo, puede considerarse como el inicio de la deducción automática.

Desde entonces se han introducido diversos refinamientos del principio de resolución encaminados a mejorar la potencia del mismo en ciertos casos, y se han proporcionado estrategias para orientar el proceso y acotar el espacio de búsqueda. Entre los refinamientos y estrategias, destacamos los siguientes:

- **Estrategia del conjunto soporte** (L. Wos, D. Carson y G. Robinson, 1965).
- **Hiper–resolución** (J.A. Robinson, 1965).
- **Demodulación** (L. Wos, G. Robinson, D. Carson y L. Shalla, 1967).
- **Paramodulación** (G. Robinson y L. Wos, 1969).
- **Resolución UR** y **estrategia de pesos** (J. McCharen, R. Overbeek y L. Wos, 1976).

que comentaremos en la siguiente sección.

1.3 Ejemplos de razonamiento automático con OTTER

OTTER (**O**rganized **T**echniques for **T**heorem–proving and **E**ffective **R**esearch) es un demostrador automático de teoremas para la lógica de primer orden con igualdad, desarrollado por W. McCune en 1988. En esta sección mostraremos cómo puede usarse OTTER para resolver distintos problemas que nos servirán para presentar las reglas y estrategias que hemos comentado anteriormente.

1.3.1 Inconsistencia de cláusulas proposicionales y resolución binaria

El primer problema que vamos a considerar consiste en determinar si un conjunto dado de fórmulas proposicionales es inconsistente (es decir, carece de modelos).

Ejemplo 1 *Determinar si el conjunto*

$$\{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$$

es inconsistente.

Para resolver el problema con OTTER tenemos que precisar dos cuestiones: la primera es sintáctica (cómo se representa el problema en OTTER) y la segunda es deductiva (qué regla de inferencia se usa para resolverlo).

En OTTER se usa el lenguaje de cláusulas. Una **cláusula** es una disyunción de **literales positivos** (i.e. fórmulas atómicas) y **literales negativos** (i.e. negaciones de fórmulas atómicas). La negación se representa por `-` y la disyunción por `|`. Las cláusula terminan en un punto. El conjunto de las clásusulas se escriben entre las expresiones `list(sos)` y `end_of_list`.

La regla de inferencia que usaremos es la **resolución binaria proposicional**: a partir de dos cláusulas que contengan literales **complementarios** (i.e. que uno sea la negación del otro) obtener una nueva cláusula haciendo la disyunción de sus literales excepto los complementarios; es decir,

```
L1 |...| Li |   A | L{i+1} |...| Ln
M1 |...| Mj | - A | M{j+1} |...| Mk
-----------------------------------------------------------
L1 |...| Li | L{i+1} |...| Ln | M1 |...| Mj | M{j+1} |...| Mk
```

donde las dos primeras reglas son las premisas (o cláusulas padres) y la tercera es la conclusión (o resolvente). La regla de resolución binaria es **adecuada** (es decir, la conclusión es consecuencia lógica de las premisas). Por tanto, una forma de demostrar que el conjunto dado es inconsistente es obtener nuevas cláusulas por resolución hasta llegar a una contradicción.

En OTTER la expresión `set(binary_res)` indica que se usa la regla de resolución binaria.

Una vez que sabemos cómo se representa en OTTER el problema y cómo se indica la regla de inferencia que deseamos utilizar, vamos a ver cómo se obtiene la solución. Para ello, se crea un archivo con la representación del problema y la regla que se usa para resolverlo. En este caso, creamos el archivo `EJ1.IN`, cuyo contenido es el siguiente:

```
list(sos).
P | Q.
-P | Q.
P | -Q.
-P | -Q.
end_of_list.
set(binary_res).
```

La demostración de la inconsistencia se obtiene mediante la siguiente orden

```
otter <ej1.in >ej1.out
```

Con la cual se ejecuta OTTER tomando como archivo de entrada el `EJ1.IN` y dirigiendo la salida al archivo `EJ1.OUT`, en el que se encuentra la siguiente demostración:

.

```
1 [] P|Q.
2 [] -P|Q.
3 [] P| -Q.
4 [] -P| -Q.
5 [binary,2.1,1.1] Q.
6 [binary,3.2,5.1] P.
7 [binary,4.1,6.1] -Q.
8 [binary,7.1,5.1] $F.
```

Veamos cómo se interpreta la demostración. Las cuatro primeras cláusulas son cláusulas de entrada (las del archivo `EJ1.IN`). La cláusula 5 se obtiene de la 1 y la 2 aplicando resolución binaria sobre sus primeros literales. Análogamente se interpretan las líneas siguientes. En la 8 se obtiene la **cláusula vacía** (representada por `$F`), con lo que se prueba que el conjunto inicial es inconsistente.

1.3.2 Inconsistencia de cláusulas de primer orden y unificación

Vamos a generalizar el problema anterior al caso de la lógica de primer orden y comprobaremos cómo mediante sustituciones adecuadas (unificadores de máxima generalidad) se reduce al caso anterior. Para ello consideramos el siguiente

Ejemplo 2 *Demostrar que el conjunto*

$$\{\neg P(x) \vee Q(x), P(a), \neg Q(z)\}$$

es inconsistente.

Procediendo como en el Ejemplo 1, escribimos en el archivo de entrada

```
list(sos).
-P(x) | Q(x).
P(a).
-Q(z).
end_of_list.
set(binary_res).
```

y obtenemos la demostración

```
1 [] -P(x)|Q(x).
2 [] P(a).
3 [] -Q(z).
4 [binary,1.1,2.1] Q(a).
5 [binary,4.1,3.1] $F.
```

Nótese que para obtener la cláusula 4 se aplica a la 1 la sustitución `{x/a}` que es un **unificador de máxima generalidad** de los literales `P(x)` y `P(a)` (es decir, la sustitución más que general que aplicada a ambos literales los hace idénticos). La forma de la **resolución binaria de primer orden** es

```
L1 |...| Li |   A | L{i+1} |...| Ln
M1 |...| Mj | - B | M{j+1} |...| Mk
-----------------------------------------------------------------
(L1 |...| Li | L{i+1} |...| Ln | M1 |...| Mj | M{j+1} |...| Mk)s
```

donde `s` es un unificador de máxima generalidad de los literales `A` y `B`.

1.3.3 Inconsistencia de fórmulas de primer orden y skolemización

Vamos a extender el problema al caso en que el conjunto de fórmulas no estén en forma de cláusula. Mediante el procedimiento de Skolem, dado un conjunto de fórmulas S se construye un conjunto de cláusulas S' tal que S es inconsistente precisamente si S' es inconsistente. Lo ilustraremos con el siguiente

Ejemplo 3 *Demostrar que el conjunto de fórmulas*

$$\{(\forall x)[P(x) \to Q(x)], P(a), \neg(\exists z)Q(z)\}$$

es inconsistente.

En OTTER también se puede usar el lenguaje lógico primer orden. En este caso, los símbolos lógicos que se necesitan son el condicional (que se representa por (`->`), el cuantificador universal (que se representa por `all`) y el cuantificador existencial (que se representa `exists`). Además, para indicar que se usan fórmulas en lugar de cláusulas, se escribe `formula_list(sos)` en lugar de `list(sos)`.

El archivo de entrada correspondiente al Ejemplo 3 es

```
formula_list(sos).
all x (P(x) -> Q(x)).
P(a).
-(exists z Q(z)).
end_of_list.
set(binary_res).
```

Al ejecutar OTTER con dicho archivo, lo primero que hace es transforma el conjunto de fórmulas en un conjunto de cláusulas equiconsistente (utilizando funciones y constantes de Skolem para eliminar cuantificadores existenciales). Una vez realizada la transformación el problema se reduce al anterior.

1.3.4 Consecuencia lógica

Hasta ahora hemos visto problemas de inconsistencia. El problema de consecuencia lógica (es decir, dado un conjunto de fórmulas S determinar si la fórmula F es consecuencia lógica de S) se reduce al de inconsistencia, ya que F es consecuencia lógica de S precisamente si $S \cup \{\neg F\}$ es inconsistente.

1.3.5 Validez de argumentaciones y representación del conocimiento

La decisión de la validez de una argumentación presenta dos nuevos problemas: uno es el problema de la **representación del conocimiento** y el otro es la explicitación del conocimiento implícito. Vamos a verlo en el siguiente

Ejemplo 4 *Demostrar la validez del siguiente argumento: Los caballos son más rápidos que los perros. Algunos galgos son más rápidos que los conejos. Lucero es un caballo y Orejón es un conejo. Por tanto, Lucero es más rápido que Orejón.*

Comenzamos determinando los símbolos de constantes que usaremos para la representación del argumento. Son los siguientes:

`Lucero`:	Lucero
`Orejon`:	Orejón
`CABALLO(x)`:	`x` es un caballo
`CONEJO(x)`:	`x` es un conejo
`GALGO(x)`:	`x` es un galgo
`PERRO(x)`:	`x` es un perro
`MAS_RAPIDO(x,y)`:	`x` es más rápido que `y`

Por lo que respecta al lenguaje, utilizamos por primera vez la conjunción (`&`). Con este lenguaje las cuatro primeras fórmulas representan las hipótesis del argumento; la quinta, la negación de la conclusión y las dos últimas, información implícita necesaria para la demostración. El archivo de entrada es

```
formula_list(sos).
all x y (CABALLO(x) & PERRO(y) -> MAS_RAPIDO(x,y)).
exists x (GALGO(x) & (all y (CONEJO(y) -> MAS_RAPIDO(x,y)))).
CABALLO(Lucero).
CONEJO(Orejon).
-MAS_RAPIDO(Lucero,Orejon).
all x (GALGO(x) -> PERRO(x)).
all x y z (MAS_RAPIDO(x,y) & MAS_RAPIDO(y,z) -> MAS_RAPIDO(x,z)).
end_of_list.
set(binary_res).
```

y la demostración obtenida es

```
1 [] -CABALLO(x)| -PERRO(y)|MAS_RAPIDO(x,y).
2 [] GALGO($c1).
3 [] -CONEJO(y)|MAS_RAPIDO($c1,y).
4 [] CABALLO(Lucero).
5 [] CONEJO(Orejon).
6 [] -MAS_RAPIDO(Lucero,Orejon).
7 [] -GALGO(x)|PERRO(x).
8 [] -MAS_RAPIDO(x,y)| -MAS_RAPIDO(y,z)|MAS_RAPIDO(x,z).
9 [binary,7.1,2.1] PERRO($c1).
10 [binary,3.1,5.1] MAS_RAPIDO($c1,Orejon).
11 [binary,1.1,4.1] -PERRO(x)|MAS_RAPIDO(Lucero,x).
16 [binary,11.1,9.1] MAS_RAPIDO(Lucero,$c1).
19 [binary,8.1,16.1] -MAS_RAPIDO($c1,x)|MAS_RAPIDO(Lucero,x).
36 [binary,19.1,10.1] MAS_RAPIDO(Lucero,Orejon).
37 [binary,36.1,6.1] $F.
```

Obsérvese que en las cláusulas 2 y 3 (correspondientes a la segunda fórmula) aparece la constante de Skolem `$c1`.

1.3.6 La estrategia del conjunto soporte

En la demostración anterior sólo intervienen 15 cláusulas, pero OTTER ha necesitado generar 37. Vamos a considerar cómo se puede reducir el número de cláusulas generadas; es decir, vamos a ver cómo orientar el sistema para evitar la generación de cláusulas innecesarias.

La primera forma de orientar el sistema será mediante la estrategia del conjunto soporte. Dicha estrategia consiste en dividir el conjunto de cláusulas S en dos subconjuntos T y $S-T$, tales que $S-T$ es consistente, y no considerar resolventes entre dos cláusulas de $S-T$. El conjunto T se llama el **soporte** y el $S-T$, conjunto **usable**. En OTTER se indica el comienzo del conjunto de fórmulas usables mediante `formula_list(usable)` y el del soporte mediante `formula_list(sos)` (en el caso de cláusulas se usan `list(usable)` y `list(sos)`, respectivamente).

Vamos a resolver de nuevo el Ejemplo 4, poniendo en el soporte sólo la cláusula correspondiente a la negación de la conclusión y en el conjunto de usables las restantes. El archivo de entrada es

```
formula_list(usable).
all x y (CABALLO(x) & PERRO(y) -> MAS_RAPIDO(x,y)).
exists x (GALGO(x) & (all y (CONEJO(y) -> MAS_RAPIDO(x,y)))).
CABALLO(Lucero).
```

```
CONEJO(Orejon).
all x (GALGO(x) -> PERRO(x)).
all x y z (MAS_RAPIDO(x,y) & MAS_RAPIDO(y,z) -> MAS_RAPIDO(x,z)).
end_of_list.
formula_list(sos).
-MAS_RAPIDO(Lucero,Orejon).
end_of_list.
set(binary_res).
```

y la demostración obtenida es

```
1 [] -CABALLO(x)| -PERRO(y)|MAS_RAPIDO(x,y).
2 [] GALGO($c1).
3 [] -CONEJO(y)|MAS_RAPIDO($c1,y).
4 [] CABALLO(Lucero).
5 [] CONEJO(Orejon).
6 [] -GALGO(x)|PERRO(x).
7 [] -MAS_RAPIDO(x,y)| -MAS_RAPIDO(y,z)|MAS_RAPIDO(x,z).
8 [] -MAS_RAPIDO(Lucero,Orejon).
9 [binary,8.1,7.3] -MAS_RAPIDO(Lucero,x)| -MAS_RAPIDO(x,Orejon).
16 [binary,9.2,3.2] -MAS_RAPIDO(Lucero,$c1)| -CONEJO(Orejon).
19 [binary,16.2,5.1] -MAS_RAPIDO(Lucero,$c1).
21 [binary,19.1,1.3] -CABALLO(Lucero)| -PERRO($c1).
22 [binary,21.1,4.1] -PERRO($c1).
24 [binary,22.1,6.2] -GALGO($c1).
25 [binary,24.1,2.1] $F.
```

en la que el número de las cláusulas generadas se ha reducido de 37 a 25.

1.3.7 Resolución UR

Otra forma de reducir el número de cláusulas generadas consiste en cambiar la regla de inferencia. Una regla de inferencia que concentra varias pasos en uno (eliminando las cláusulas intermedias y las que éstas generarían) es la **resolución UR** ("unit resulting"). La forma de dicha regla es

```
L_1 |...| L_n.
M_1.
...
M_{i-1}.
M_{i+1}.
...
M_n.
-------------
(L_i)s.
```

donde `s` es un unificador de máxima generalidad tal que para todo $j \in \{1, \dots, i-1, i+1, \dots, n\}$, `(L_j)s = (M'_j)s` y `M'_j` es el complementario de `M_j`.

En OTTER la expresión `set(ur_res).` indica que se usa la regla de resolución UR.

Para repetir el Ejemplo 4 utilizando la resolución UR, escribimos en el archivo de entrada

```
formula_list(sos).
all x y (CABALLO(x) & PERRO(y) -> MAS_RAPIDO(x,y)).
exists x (GALGO(x) & (all y (CONEJO(y) -> MAS_RAPIDO(x,y)))).
CABALLO(Lucero).
CONEJO(Orejon).
-MAS_RAPIDO(Lucero,Orejon).
all x (GALGO(x) -> PERRO(x)).
all x y z (MAS_RAPIDO(x,y) & MAS_RAPIDO(y,z) -> MAS_RAPIDO(x,z)).
end_of_list.
set(ur_res).
```

y se obtiene la demostración

```
1 [] -CABALLO(x)| -PERRO(y)|MAS_RAPIDO(x,y).
2 [] GALGO($c1).
3 [] -CONEJO(y)|MAS_RAPIDO($c1,y).
4 [] CABALLO(Lucero).
5 [] CONEJO(Orejon).
6 [] -MAS_RAPIDO(Lucero,Orejon).
7 [] -GALGO(x)|PERRO(x).
8 [] -MAS_RAPIDO(x,y)| -MAS_RAPIDO(y,z)|MAS_RAPIDO(x,z).
9 [ur,7,2] PERRO($c1).
10 [ur,3,5] MAS_RAPIDO($c1,Orejon).
12 [ur,1,4,9] MAS_RAPIDO(Lucero,$c1).
14 [ur,8,10,6] -MAS_RAPIDO(Lucero,$c1).
15 [binary,14.1,12.1] $F.
```

en la que el número de cláusulas generadas se ha reducido a 15 (ya no tenemos ninguna innecesaria). Nótese que la cláusula 12 se obtiene por resolución UR a partir de las cláusulas 1, 4 y 9 como se muestra en el siguiente esquema

```
-CABALLO(x)      | -PERRO(y)  |MAS_RAPIDO(x,y)
          |                 |
 CABALLO(Lucero)   PERRO($c1)
```

en el que se usa la sustitución `{x/Lucero, y/$c1}`.

1.3.8 Hiper–resolución

Otra regla de inferencia que concentra varias pasos en uno es la regla de **hiper–resolución**. La forma de dicha regla es

```
-A_1 |...| -A_n | B_1 |...| B_m.
M_1.
...
M_n.
-------------
(B_1 |...| B_m)s.
```

donde `A_i` y `B_j` son átomos y `s` es un unificador de máxima generalidad tal que para todo $i \in \{1, \dots, n\}$, `(A_j)s = (M_j)s`.

En OTTER la expresión `set(hyper_res)` indica que se usa la regla de hiper–resolución.

Para repetir el Ejemplo 4 utilizando la regla de hiper–resolución, escribimos en el archivo de entrada

```
formula_list(sos).
all x y (CABALLO(x) & PERRO(y) -> MAS_RAPIDO(x,y)).
exists x (GALGO(x) & (all y (CONEJO(y) -> MAS_RAPIDO(x,y)))).
CABALLO(Lucero).
CONEJO(Orejon).
-MAS_RAPIDO(Lucero,Orejon).
all x (GALGO(x) -> PERRO(x)).
all x y z (MAS_RAPIDO(x,y) & MAS_RAPIDO(y,z) -> MAS_RAPIDO(x,z)).
end_of_list.
set(hyper_res).
```

y se obtiene la demostración

```
1 [] -CABALLO(x)| -PERRO(y)|MAS_RAPIDO(x,y).
2 [] GALGO($c1).
3 [] -CONEJO(y)|MAS_RAPIDO($c1,y).
4 [] CABALLO(Lucero).
5 [] CONEJO(Orejon).
6 [] -MAS_RAPIDO(Lucero,Orejon).
7 [] -GALGO(x)|PERRO(x).
8 [] -MAS_RAPIDO(x,y)| -MAS_RAPIDO(y,z)|MAS_RAPIDO(x,z).
9 [hyper,7,2] PERRO($c1).
10 [hyper,3,5] MAS_RAPIDO($c1,Orejon).
11 [hyper,1,4,9] MAS_RAPIDO(Lucero,$c1).
12 [hyper,8,11,10] MAS_RAPIDO(Lucero,Orejon).
13 [binary,12.1,6.1] $F.
```

1.3.9 Obtención de respuestas

El problema que vamos a estudiar a continuación es el siguiente: Dado un conjunto de fórmulas S y una fórmula $F(x_1,\dots,x_n)$, cuyas variables libres son $x_1,\dots,x_n$, encontrar términos $t_1,\dots,t_n$ tales que $F(t_1,\dots,t_n)$ sea consecuencia de S. Para resolverlo introducimos un nuevo símbolo de predicados (`$ANS`), consideramos el conjuntos de las cláusulas correspondientes a las fórmulas de

$$S \cup \{(\forall x_1)\dots(\forall x_n)[F(x_1,\dots,x_n) \to \$ANS(x_1,\dots,x_n)]\}$$

y aplicamos el procedimiento de resolución hasta encontrar una cláusula cuyo único literal contenga el predicado `$ANS`; los términos que aparecen en dicho literal forman una respuesta a la cuestión planteada.

Veamos un ejemplo y su solución en OTTER:

Ejemplo 5 *Dado* $\{\forall(P(x) \to Q(x)), P(a)\}$ *determinar un* z *tal que* $Q(z)$ *sea consecuencia del conjunto.*

El contenido del archivo de entrada es

```
list(sos).
all x (P(x) -> Q(x)).
P(a).
all z (Q(z) -> $ANS(z)).
end_of_list.
set(binary_res).
```

y la demostración obtenida es

```
1 [] -P(x)|Q(x).
2 [] P(a).
3 [] -Q(z)|$ANS(z).
4 [binary,1.1,2.1] Q(a).
5 [binary,4.1,3.1] $ANS(a).
```

que indica que la respuesta es `a`.

1.3.10 Validez de argumentaciones con igualdad

Hasta ahora sólo hemos considerado la lógica de primer orden sin igualdad. Una forma de resolver problemas en los que interviene la igualdad es añadiendo los axiomas propios de la igualdad comunes (es decir, los de reflexividad, simetría y transitividad) y los específicos (es decir, los de sustitución correspondientes a los símbolos de funciones y predicados que estemos considerando). Vamos a verlo mediante el siguiente ejemplo:

Ejemplo 6 *Demostrar que si Luis es el padre de Juan, entonces Luis es mayor que Juan.*

Empezamos por fijar el lenguaje.

`juan`:	Juan
`luis`:	Luis
`padre(x)`:	el padre de `x`
`x = y`:	`x` e `y` son iguales
`x != y`:	`x` e `y` son distintos
`MAYOR(x,y)`:	`x` es mayor que `y`

Usando dichoo lenguaje, el contenido del archivo de entrada es

```
list(sos).
padre(juan)=luis.
-MAYOR(luis,juan).
MAYOR(padre(x),x).
x=x.
x!=y | y=x.
x!=y | y!=z | x=z.
x!=y | padre(x)=padre(y).
x1!=x2 | -MAYOR(x1,y) | MAYOR(x2,y).
y1!=y2 | -MAYOR(x,y1) | MAYOR(x,y2).
end_of_list.
set(binary_res).
```

en el que las últimas cláusulas son los axiomas de sustitución del símbolo de función `padre` y del símbolo de predicado `MAYOR`. La demostración obtenida es

```
1 [] padre(juan)=luis.
2 [] -MAYOR(luis,juan).
3 [] MAYOR(padre(x),x).
4 [] x=x.
5 [] x!=y|y=x.
6 [] x!=y|y!=z|x=z.
8 [] x1!=x2| -MAYOR(x1,y)|MAYOR(x2,y).
29 [binary,6.1,1.1] luis!=x|padre(juan)=x.
53 [binary,29.1,5.2] padre(juan)=x|x!=luis.
303 [binary,8.3,2.1] x!=luis| -MAYOR(x,juan).
312 [binary,303.1,53.1,unit_del,3,4] $F.
```

De la demostración llama la atención que de las 312 cláusulas generadas por OTTER sólo se necesitan 11 en la demostración. Esto indica que la solución trivial es poco eficiente.

Una forma de reducir el número de cláusulas innecesarias es usar la estrategia del conjunto soporte. Vamos a repetir el problema, pero dejando en el conjunto soporte sólo la negación de la conclusión. El contenido del archivo de entrada es

```
list(usable).
padre(juan)=luis.
MAYOR(padre(x),x).
x=x.
x!=y | y=x.
x!=y | y!=z | x=z.
x!=y | padre(x)=padre(y).
x1!=x2 | -MAYOR(x1,y) | MAYOR(x2,y).
y1!=y2 | -MAYOR(x,y1) | MAYOR(x,y2).
end_of_list.
list(sos).
-MAYOR(luis,juan).
end_of_list.
set(binary_res).
```

y la demostración obtenida es

```
1 [] padre(juan)=luis.
```

```
2 [] MAYOR(padre(x),x).
7 [] x1!=x2| -MAYOR(x1,y)|MAYOR(x2,y).
9 [] -MAYOR(luis,juan).
11 [binary,9.1,7.3] x!=luis| -MAYOR(x,juan).
17 [binary,11.1,1.1] -MAYOR(padre(juan),juan).
18 [binary,17.1,2.1] $F.
```

El número de cláusulas generadas se ha reducido de 312 a 18 y la longitud de la prueba se ha reducido de 11 a 7.

Otra forma de reducir el número de cláusulas innecesarias es usar como regla de inferencia la resolución UR. Para ello, el contenido del archivo de entrada es

```
list(sos).
padre(juan)=luis.
-MAYOR(luis,juan).
MAYOR(padre(x),x).
x=x.
x!=y | y=x.
x!=y | y!=z | x=z.
x!=y | padre(x)=padre(y).
x1!=x2 | -MAYOR(x1,y) | MAYOR(x2,y).
y1!=y2 | -MAYOR(x,y1) | MAYOR(x,y2).
end_of_list.
set(ur_res).
```

y la demostración obtenida es

```
1 [] padre(juan)=luis.
2 [] -MAYOR(luis,juan).
3 [] MAYOR(padre(x),x).
8 [] x1!=x2| -MAYOR(x1,y)|MAYOR(x2,y).
17 [ur,8,3,2] padre(juan)!=luis.
18 [binary,17.1,1.1] $F.
```

en la que el número de cláusulas generadas es 18, como en la anterior, pero la longitud de la prueba es 6 (una menos que en la anterior).

1.3.11 Paramodulación

En esta sección vamos a introducir una nueva regla de inferencia especialmente adaptada para tratar la igualdad. Dicha regla es la **paramo-**

dulación, cuya forma es

```
L1 |...| Li[t1] |...| Ln.
M1 |...| M{j-1} | t2=t3 |M{j+1} |...| Mk.
------------------------------------------------------------
(L1 |...|Li[t3] |...|Ln |M1 |...|M{j-1}|t2=t3 |M{j+1}|...|Mk)s
```

donde `Li[t1]` indica que el literal `Li` contiene el término `t1` que es unificable con el término t2, `s` es un unificador de máxima generalidad de `t1` y `t2`, y `Li[t3]` es el literal obtenido sustituyendo en `L1` la ocurrencia de `t1` por `t3`. Usando la regla de paramodulación, el único axioma de la igualdad que se necesita añadir es el axioma de reflexividad.

En OTTER las expresiones `set(para_from)` y `set(para_into)` indican que se usa la regla de paramodulación. El uso de dos indicadores se explica por el procedimiento de demostración de OTTER, según la ecuación ocurra o no en la cláusula elegida.

Para resolver el Ejemplo 6 con la regla de paramodulación, tenemos que escribir en el archivo de entrada

```
list(sos).
padre(juan)=luis.
-MAYOR(luis,juan).
MAYOR(padre(x),x).
x=x.
end_of_list.
set(para_into).
set(para_from).
```

y la demostración obtenida es

```
1 [] padre(juan)=luis.
2 [] -MAYOR(luis,juan).
3 [] MAYOR(padre(x),x).
5 [para_into,3.1.1,1.1.1] MAYOR(luis,juan).
6 [binary,5.1,2.1] $F.
```

1.3.12 Demodulación

En esta sección presentaremos otra estrategia que permite reducir el número de cláusulas innecesarias en problemas que utilizan igualdades. Lo haremos resolviendo el siguiente ejemplo.

Ejemplo 7 *Sea G un grupo y e su elemento neutro. Demostrar que si, para todo x de G, $x^2 = e$, entonces G es conmutativo.*

Los axiomas de grupo que utilizaremos son los siguientes:

$$\begin{array}{l}
(\forall x)[e.x = x] \\
(\forall x)[x.e = x] \\
(\forall x)[x.x^{-1} = e] \\
(\forall x)[x^{-1}.x = e] \\
(\forall x)[(x.y).z = x.(y.z)]
\end{array}$$

Tenemos que demostrar que si $(\forall x)[x.x = e]$, entonces $(\forall x)(\forall y)[x.y = y.x]$.

Para obtener la demostración con OTTER escribimos en el archivo de entrada las cláusulas correspondientes a los axiomas de grupo y al axioma reflexivo (en la lista de usables) y las correspondientes a la hipótesis y a la negación de la conclusión (en la lista soporte). Además, elegimos como regla de inferencia la de paramodulación. El contenido del archivo de entrada es

```
list(usable).
e * x = x. % Ax. 1
x * e = x. % Ax. 2
x^ * x = e. % Ax. 3
x * x^ = e. % Ax. 4
(x * y) * z = x * (y * z). % Ax. 5
x = x.
end_of_list.

list(sos).
x * x = e.
a * b != b * a.
end_of_list.

set(para_into).
set(para_from).
```

y la demostración obtenida es

```
1 [] e*x=x.
2 [] x*e=x.
5 [] (x*y)*z=x*y*z.
```

```
7 [] x*x=e.
8 [] a*b!=b*a.
19 [para_from,7.1.2,2.1.1.2] x*y*y=x.
20 [para_from,7.1.2,1.1.1.1] (x*x)*y=y.
31 [para_into,19.1.1,5.1.2] (x*y)*y=x.
167 [para_into,20.1.1,5.1.1] x*x*y=y.
170 [para_from,20.1.1,5.1.1] x=y*y*x.
496 [para_into,167.1.1.2,31.1.1] (x*y)*x=y.
755 [para_into,496.1.1.1,170.1.2] x*y=y*x.
756 [binary,755.1,8.1] $F.
```

Vamos a ver cómo reducir el número de cláusulas generadas. Analizando el archivo de salida, se observa que aparecen cláusulas con términos en los que figura el elemento neutro como factor. Estas cláusulas pueden simplificarse teniendo en cuenta los axiomas del neutro. La estrategia de demodulación consiste en usar igualdades como reglas de simplificación. Las igualdades que se usen de esta forma se llaman **demoduladores**.

En OTTER la expresión `list(demodulators).` indica el principio de la lista de demoduladores.

Vamos a repetir la prueba usando como demoduladores los axiomas de grupo. El contenido del archivo de entrada es

```
list(usable).
e * x = x.                      % Ax. 1
x * e = x.                      % Ax. 2
x^ * x = e.                     % Ax. 3
x * x^ = e.                     % Ax. 4
(x * y) * z = x * (y * z).      % Ax. 5
x = x.
end_of_list.

list(sos).
x * x = e.
a * b != b * a.
end_of_list.

list(demodulators).
e * x = x.                      % Ax. 1
x * e = x.                      % Ax. 2
x^ * x = e.                     % Ax. 3
x * x^ = e.                     % Ax. 4
(x * y) * z = x * (y * z).      % Ax. 5
```

```
end_of_list.

set(para_into).
set(para_from).
```

y la demostración obtenida es

```
1 [] e*x=x.
5 [] (x*y)*z=x*y*z.
7 [] x*x=e.
8 [] a*b!=b*a.
10 [] x*e=x.
13 [] (x*y)*z=x*y*z.
14 [para_into,7.1.1,5.1.2,demod,13,13,13] x*y*x*y=e.
20 [para_from,7.1.2,1.1.1.1,demod,13] x*x*y=y.
494 [para_from,14.1.1,20.1.1.2,demod,10] x=y*x*y.
540 [para_from,494.1.2,20.1.1.2] x*y=y*x.
541 [binary,540.1,8.1] $F.
```

con lo que se consigue reducir el número de cláusulas generadas de 756 a 541.

Otra forma de reducir más cláusulas es adoptando la estrategia de considerar como demoduladores las igualdades que se van generando. En OTTER se indica la adopción de esta estrategia mediante la expresión `set(dynamic_demod)`. Si añadimos dicha expresión al anterior archivo de entrada, obtenemos la siguiente demostración

```
5 [] (x*y)*z=x*y*z.
7 [] x*x=e.
8 [] a*b!=b*a.
9 [] e*x=x.
10 [] x*e=x.
13 [] (x*y)*z=x*y*z.
14 [para_into,7.1.1,5.1.2,demod,13,13,13] x*y*x*y=e.
19 [para_from,7.1.1,5.1.1.1,demod,9] x*x*y=y.
31 [para_from,14.1.1,19.1.1.2,demod,10] x*y*x=y.
36 [para_from,31.1.1,19.1.1.2] x*y=y*x.
37 [binary,36.1,8.1] $F.
```

en la que sólo se generan 37 cláusulas frente a las 756 de la primera prueba.

1.4 Conclusiones

En este trabajo hemos mostrado distintas problemas que pueden resolverse mediante un demostrador automático. Evidentemente no son las únicas ni las más importantes, pero sirven para ilustrar los principales problemas que hay que resolver cuando se utiliza un demostrador:

- Representación del conocimiento: cuál es la información de que se dispone (explícita e implícita) y cuál es la mejor manera de representarla.
- Sistema de inferencia: cuáles son las reglas que convienen aplicar para la solución del problema que estamos estudiando.
- Estrategias: cómo orientar la aplicación de las reglas y simplificar la información generada.

Para profundizar en el estudio del razonamiento automático puede hacerse mediante el libro de Chang y Lee (que es el clásico del campo) y el de Wos y otros (que es una aproximación más aplicada que el anterior).

Finalmente, desamos comentar que para el desarrollo del razonamiento automático se necesita seguir experimentando y abriendo nuevos campos en los que se apliquen los sistemas, lo que planteará nuevos problemas y ampliará el repertorio de representaciones, inferencias y estrategias disponibles.

1.5 Bibliografía.

1. Davis, M. *A computer program for Presburger's algorithm.* S. Inst. for Symbolic Logic, Cornell U.V. (1957), 215–233.

2. Davis, M. *The prehistory and early of automated deduction.* Automation of Reasoning: Classical Papers on Computational Logic, Vol 1. Ed. by J. Siekmann and G. Wrightson. Springer Verlag (1983), 1–28.

3. Chang, C.L.; Lee, R.C.T *Symbolic logic and mechanical theorem proving.* Academic press, 1973.

4. Davis, M.; Putnam, H. *A computing procedure for quantification theory.* Journal of the ACM, 7, 3 (1960), 201–215.

4. Gilmore, M. *A proof method for quantification theory, its justification and realization.* IBM J. Res. Develop. (1960), 28–35.

5. Loveland, D.W.. *Automated theorem proving: a quarter century review.* Contemporary Mathematics. Automated theorem proving: after 25 years. American Mathematical Society, Vol 29 (1983), 1–46.

6. McCharen, J.; Overbeek, R.; Wos, L *Complexity and related enhancement for automated theorem–proving program.* Computer and Mathematics with applications. (1976), 1–16.

7. McCune, W. *OTTER 3.0 Reference Manual and Guide".* **ANL** (1994).

8. Newell, A.; Shaw, J.C.; Simon H.A. *Empirical explorations with the Logic Theory Machine: A case study in heuristics.* Proc. Western Joint Computer Conf. (1956), 218–239.

9. Prawitz, D. *An improved proof procedure.* Theoria (1960), 102–139.

10. Robinson, G; Wos, L. *Paramodulation and theorem–proving in first–order theories with equality.* Machine Intelligence (B. Meltzer and D. Michie, eds.), Edinburg University Press (1969), 135–150.

11. Robinson, J.A. *A machine oriented logic based on the Resolution Principle.* Journal of the ACM 12 (1965) 23–41.

12. Robinson, J.A. *Automatic deduction with Hyper–Resolution.* Journal of Computer Mathematics, 1 (1965), 227–334.

13. Wang, H. *Proving theorem by pattern recognition, Part I..* Comm. ACM (1960), 220–234.

14. Whitehead, A.N.; Russell, B.. Prinicipia Mathematica. Vol 1. Cambridge University Press (1910).

15. Wos, L.; Carson, D.; Robinson, G.. *Eficiency and completeness of the support strategy in theorem proving.* Journal of the ACM (1965), 536–541.

16. Wos, L.; Henschen, L.. *Automated theorem proving 1965–1970.* Automation of reasoning: classical papers on computational logic, Vol 2. Ed. by J. Siekmann and G. Wrightson. Springer Verlag (1983), 1–24.

17. Wos, L.; Overbeek, R.; Lusk, E.; Boyle, J. *Automated Reasoning: Introduction and Applications, revised edition.* McGraw-Hill, 1992.

18. Wos, L.; Robinson, G.; Carson, D.; Shala, L.. *The concept of demodulation in theorem proving.* Journal of the ACM (1967), 698–709.