Razonamiento automático en lógica de primer orden

Juan J. Arrabal, José A. Alonso, Delia Balbontín y Javier Herrera
Departamento de Álgebra, Computación, Geometría y Topología
Facultad de Matemáticas
Universidad de Sevilla

12 de Julio de 1992

1 Introducción

El objetivo del presente trabajo consiste en una demostración de las posibilidades del programa OTTER [McCune 90] para la demostración automática de teoremas de Lógica de primer orden. Para dicha demostración hemos elegido como campo de experimentación los ejercicios propuestos por Jesús Mosterín en su libro "Lógica de primer orden". En la resolución de los ejercicios nos encontraremos con los problemas centrales del razonamiento automático: representación del conocimiento, elección de reglas de inferencia y utilización de estrategias de prueba. Como es corriente en la experimentación en razonamiento automático, a lo largo del experimento aparecen nuevos problemas; en nuestro experimento nos hemos encontrado con el problema del tratamiento de los descriptores (ejercicios 25–35) para los cuales hemos introducidos nuevos axiomas, pero queda pendiente un estudio más detallado que posiblemente demuestre la conveniencia de introducir una nueva regla de inferencia.

2 Ejercicios

En esta sección presentamos las soluciones a algunos de los ejercicios de [Mosterín 70] (pp. 58–87) usando la misma numeración del libro. La selección se ha hecho considerando los ejercicios que implican la introducción de nuevas reglas de inferencia o estrategias de búsquedas.

Ejercicio 1.
$\vdash \forall x(P \to P)$

Para resolverlo, elegimos como regla de inferencia la resolución binaria y ponemos en el conjunto soporte la negación de la fórmula a probar. El fichero de entrada para el Ejercicio 1 es

```
set(binary_res).
formula_list(sos).
- (all x (P -> P)).
end_of_list.
```

El programa OTTER transforma la fórmula a cláusulas y realiza la demostración, dando como fichero de salida

```
-------> sos clausifies to:
1 [] P.
2 [] -P.
-------- PROOF --------
1 [] P.
2 [] -P.
3 [binary,2,1] $F.
```

En el proceso de clausificación se obtienen las cláusulas `P` y `-P` (la negación de `P`) que inmediatamente generan la contradicción (`$F`).

Ejercicio 2.
$\bigwedge xy(Rxy \vee Ryx) \vdash \forall xRxx$

Para este ejercicio no es suficiente la regla de resolución binaria. Lo resolveremos añadiéndole la regla de factorización. Además, colocamos en el conjunto soporte la hipótesis y la negación de la conclusión. El fichero de entrada es

```
set(binary_res).
set(factor).
formula_list(sos).
all x y (R(x,y) | R(y,x)).
- (all x R(x,x)).
```

```
end_of_list.
```

y el de salida es

```
-------> sos clausifies to:
1 [] R(x,y)|R(y,x).
2 [] -R($c1,$c1).
-------- PROOF --------
1 [] R(x,y)|R(y,x).
2 [] -R($c1,$c1).
3 [binary,1,2] R($c1,$c1).
4 [binary,3,2] $F.
```

La clausificación de la hipótesis genera la cláusula 1 (donde | es el símbolo de disyunción). La negación de la conclusión indica que existe un elemento x que no verifica la fórmula Rxx; en el proceso de clausificación, se introduce la constante de Skolem `$c1` para representar un elemento verificando la negación de Rxx.

Para el Ejercicio 3 basta resolución binaria.

Ejercicio 4.
$\vdash \bigvee y(Fy \to \bigwedge xFx)$

El fichero de entrada es

```
set(binary_res).
formula_list(sos).
- (exists y (F(y) -> (all x F(x)))).
end_of_list.
```

y el de salida es

```
-------> sos clausifies to:
1 [] F(y).
2 [] -F($f1(y)).
-------- PROOF --------
1 [] F(y).
2 [] -F($f1(y)).
3 [binary,2,1] $F.
```

La negación de la fórmula a demostrar dice que para cada elemento y que verifica la fórmula Fy existe un elemento (que depende de y) que no la verifica; dicho elemento está representado en la cláusula 2 por la función de Skolem `$f1(y)`.

Los ejercicios 5, 6 y 7 requieren resolución binaria. Además, en el 6 se necesita factorización.

Ejercicio 8.
$\{\bigwedge xy(\bigvee(Rxu \wedge Ryu) \to Rxy),$
$\bigwedge x \bigvee yRxy\}$
$\vdash \bigwedge xyz(Rxy \wedge Ryz \to Rxz$

Para resolver el ejercicio con resolución binaria, se necesitan 36 pasos de resolución. Una forma de obtener una demostración más corta consiste en utilizar como regla de inferencia la hiper–resolución. El fichero de entrada es

```
set(hyper_res).
formula_list(sos).
all x y ((exists u (R(x,u) & R(y,u)))
 -> R(x,y)).
all x exists y R(x,y).
- (all x y z (R(x,y) & R(y,z) ->
 R(x,z))).
end_of_list.
```

y el fichero de salida es

```
-------> sos clausifies to:
1 [] -R(x,u)| -R(y,u)|R(x,y).
2 [] R(x,$f1(x)).
3 [] R($c3,$c2).
4 [] R($c2,$c1).
5 [] -R($c3,$c1).
-------- PROOF --------
1 [] -R(x,u)| -R(y,u)|R(x,y).
2 [] R(x,$f1(x)).
3 [] R($c3,$c2).
4 [] R($c2,$c1).
5 [] -R($c3,$c1).
8 [hyper,1,2,2] R(x,x).
9 [hyper,8,1,4] R($c1,$c2).
12 [hyper,9,1,3] R($c3,$c1).
13 [binary,12,5] $F.
```

El Ejercicio 9 se resuelve con resolución binaria; el 10, con hiper–resolución; el 11, con hiper–resolución y factorización.

Ejercicio 12.
$\{\bigwedge x((Bx \vee Nx) \wedge \neg(Bx \wedge Nx)),$
$\bigwedge x(fx = a \to Bx),$
$fa = a\}$
$\vdash \neg \bigwedge xNx$

Aunque en este ejercicio interviene el predicado de igualdad, puede resolverse con resolución binaria. El fichero de entrada es

```
set(binary_res).
formula_list(sos).
all x ((B(x) | N(x)) & - (B(x) &
 N(x))).
all x (f(x) = a -> B(x)).
```

```
f(a) = a.
all x N(x).
end_of_list.
```

y el de salida es

```
-------> sos clausifies to:
1 [] B(x)|N(x).
2 [] -B(x)| -N(x).
3 [] f(x)!=a|B(x).
4 [] f(a)=a.
5 [] N(x).
-------- PROOF --------
2 [] -B(x)| -N(x).
3 [] f(x)!=a|B(x).
4 [] f(a)=a.
5 [] N(x).
6 [binary,2,5] -B(x).
7 [binary,3,4] B(a).
8 [binary,7,6] $F.
```

En la cláusula 3 aparece el símbolo `!=` que representa $\neq$.

Ejercicio 13.
$\{-\bigvee x(\bigwedge y(y = a \to Py) \to Hx),$
$\bigvee x - Hx \to -\bigwedge x(Pa \wedge -Hx)\}$
$\vdash \bigwedge xHx$

Para resolverlo basta la resolución binaria como regla de inferencia, pero se necesita introducir el axioma reflexivo ($x = x$). Dicho axioma se usa en los restantes ejercicios salvo 14, 15, 16 y 34. El fichero de entrada es

```
set(binary_res).
list(usable).
x = x.
end_of_list.
formula_list(sos).
- (exists x ((all y (y = a -> P(y)))
 -> H(x))).
(exists x (- H(x))) -> - (all x (P(a)
 & - H(x))).
- (all x H(x)).
end_of_list.
```

El fichero de salida es

```
1 [] x=x.
-------> sos clausifies to:
2 [] y!=a|P(y).
3 [] -H(x).
4 [] H(x)| -P(a)|H($c1).
5 [] -H($c2).
-------- PROOF --------
1 [] x=x.
2 [] y!=a|P(y).
3 [] -H(x).
4 [] H(x)| -P(a)|H($c1).
5 [] -H($c2).
6 [binary,2,1] P(a).
7 [binary,4,6] H(x)|H($c1).
9 [binary,7,3] H(x).
10 [binary,9,5] $F.
```

El Ejercicio 14 se resuelve con hiper–resolución.

Ejercicio 15.
$\{a = hfcc \vee \bigwedge xRxx,$
$\bigwedge xfcx = c,$
$\neg a = hc\}$
$\vdash \neg Rcc \to Rcc$

En este ejercicio es conveniente usar como regla de inferencia la paramodulación. El fichero de entrada es

```
set(binary_res).
set(para_from).
formula_list(sos).
a=h(f(c,c)) | (all x R(x,x)).
all x (f(c,x)=c).
a!=h(c).
-(-R(c,c) -> R(c,c)).
end_of_list.
```

y el de salida es

```
-------> sos clausifies to:
1 [] a=h(f(c,c))|R(x,x).
2 [] f(c,x)=c.
3 [] a!=h(c).
4 [] -R(c,c).
5 [] -R(c,c).
-------- PROOF --------
1 [] a=h(f(c,c))|R(x,x).
2 [] f(c,x)=c.
3 [] a!=h(c).
5 [] -R(c,c).
10 [para_from,2,3] a!=h(f(c,x)).
28 [binary,1,10] R(x,x).
29 [binary,28,5] $F.
```

Ejercicio 16.
$\vdash \neg(\neg fa = fb \vee \neg Hfa) \to Hfb$

Este ejercicio puede resolverse con resolución binaria, añadiendo el axioma de sustitución correspondiente a H. El fichero de entrada es

```
set(binary_res).
formula_list(usable).
all x y (x = y -> (H(x) -> H(y))).
end_of_list.
formula_list(sos).
-( - (f(a) != f(b) | - H(f(a))) ->
 H(f(b))).
end_of_list.
```

y el de salida es

```
-------> usable clausifies to:
1 [] x!=y| -H(x)|H(y).
-------> sos clausifies to:
2 [] f(a)=f(b).
3 [] H(f(a)).
4 [] -H(f(b)).
-------- PROOF --------
1 [] x!=y| -H(x)|H(y).
2 [] f(a)=f(b).
3 [] H(f(a)).
4 [] -H(f(b)).
5 [binary,3,1] f(a)!=x|H(x).
8 [binary,5,2] H(f(b)).
9 [binary,8,4] $F.
```

El Ejercicio 17 se soluciona utilizando resolución binaria y paramodulación; el 18 y 19, con resolución binaria.

Ejercicio 20.

$\{\bigwedge xyuw(fxu = y \wedge fxw = y \rightarrow u = w,$
$\bigvee z \bigwedge x(fxz = x \wedge \bigvee yfxy = z\}$
$\vdash \bigvee z \bigwedge w(\bigwedge xfxw = x \leftrightarrow w = z$

Para este ejercicio se necesita una nueva regla de inferencia (la UR–resolución) y una nueva estrategia (la demodulación dinámica). Además, hemos añadido un axioma de sustitución para la función f y reducido el conjunto soporte a la negación de la conclusión. El fichero de entrada es

```
set(ur_res).
set(factor).
set(para_from).
set(dynamic_demod).
formula_list(usable).
all x (x=x).
all x y u w (f(x,u)=y & f(x,w)=y ->
 u=w).
exists z all x (f(x,z)=x & (exists y
 (f(x,y)=z))).
end_of_list.
formula_list(sos).
- (exists z all w ((all x (f(x,w)=x))
 <-> w=z)).
end_of_list.
```

y el de salida es

```
-------> usable clausifies to:
1 [] x=x.
2 [] f(x,u)!=y|f(x,w)!=y|u=w.
3 [] f(x,$c1)=x.
4 [] f(x,$f1(x))=$c1.
-------> sos clausifies to:
5 [] f(x,$f3(z))=x|$f3(z)=z.
6 [] f($f2(z),$f3(z))!=$f2(z)|
  $f3(z)!=z.
-------- PROOF --------
1 [] x=x.
2 [] f(x,u)!=y|f(x,w)!=y|u=w.
3 [] f(x,$c1)=x.
5 [] f(x,$f3(z))=x|$f3(z)=z.
6 [] f($f2(z),$f3(z))!=$f2(z)|
  $f3(z)!=z.
7 [para_from,5,2] f(x,y)!=z|x!=z|
  $f3(u)=y|$f3(u)=u.
8 [factor,7] f(x,y)!=z|x!=z|$f3(y)=y.
16,15 [ur,8,3,1] $f3($c1)=$c1.
24 [ur,15,6,demod,16] f($f2($c1),$c1)
   !=$f2($c1).
25 [binary,24,3] $F.
```

Ejercicio 21.

$\{\bigwedge xyzffxyz = fxfyz,$
$\bigwedge xfxhx = k,$
$\bigwedge xfkx = x\}$
$\vdash \bigwedge xy \bigvee zfxz = y$

Podemos interpretar las hipótesis del ejercicio como axiomas de la teoría de grupos. Representaremos el símbolo de función f por el operador infijo + y el símbolo de función h por el operador prefijo -. Además, reducimos el conjunto soporte a la negación de la conclusión. El fichero de entrada es

```
set(binary_res).
set(para_into).
op(500,xfy,+).
```

```
op(400,xf,'-').
formula_list(usable).
all x (x=x).
all x y z ((x + y) + z=x + (y + z)).
all x (x + (- x) = e).
all x (e + x = x).
end_of_list.
formula_list(sos).
- (all x y ((exists z (x + z = y)))).
end_of_list.
```

y el de salida es

```
-------> usable clausifies to:
1 [] x=x.
2 [] (x+y)+z=x+y+z.
3 [] x+ -x=e.
4 [] e+x=x.
-------> sos clausifies to:
5 [] $c2+z!=$c1.
-------- PROOF --------
2 [] (x+y)+z=x+y+z.
3 [] x+ -x=e.
4 [] e+x=x.
5 [] $c2+z!=$c1.
7 [para_into,5,2] ($c2+x)+y!=$c1.
12 [para_into,7,3] e+x!=$c1.
13 [binary,12,4] $F.
```

Ejercicio 22.
$\{\bigwedge xyzfxfyz = ffxyz,$
$\bigwedge xy \bigvee zfxz = y,$
$\bigwedge xy \bigvee zfzx = y\}$
$\vdash \bigvee z \bigwedge x(fxz = x \wedge \bigvee yfxy = z)$

Para resolver este ejercicio necesitamos utilizar una nueva regla de inferencia, la hiper–resolución negativa, y nuevas estrategias: el uso del conjunto soporte como cola, demodulación hacia atrás y demoduladores explícitos. El fichero de entrada es

```
set(neg_hyper_res).
set(para_from).
set(back_demod).
set(sos_queue).
op(500,xfy,+).
op(400,xf,'-').
formula_list(usable).
all x (x=x).
all x y z (x+(y+z)=(x+y)+z).
end_of_list.
formula_list(sos).
- (exists z (all x (x+z=x &
 (exists y (x+y=z))))).
all x y (exists z (x + z = y)).
all x y (exists z (z + x = y)).
end_of_list.
list(demodulators).
((x + y) + z = x + y + z).
end_of_list.
```

y el de salida es

```
-------> usable clausifies to:
1 [] x=x.
2 [] x+y+z= (x+y)+z.
-------> sos clausifies to:
3 [] $f1(z)+z!=$f1(z)|$f1(z)+y!=z.
4 [] x+$f2(x,y)=y.
5 [] $f3(x,y)+x=y.
list(demodulators).
6 [] (x+y)+z=x+y+z.
end_of_list.
-------- PROOF --------
2 [] x+y+z= (x+y)+z.
3 [] $f1(z)+z!=$f1(z)|$f1(z)+y!=z.
4 [] x+$f2(x,y)=y.
5 [] $f3(x,y)+x=y.
6 [] (x+y)+z=x+y+z.
7 [neg_hyper,4,3] $f1(x)+x!=$f1(x).
11 [para_from,4,2,demod,6] x+z+$f2(z,
   y)=x+y.
12 [para_from,5,2] $f3(x,y)+x+z=y+z.
55 [para_from,11,5] $f3(x,y)+z+$f2(z,
   x)=y.
81 [para_from,12,7] $f3(x,$f1(y))+x+y
   !=$f1(y).
82 [binary,81,55] $F.
```

Ejercicio 23.
$\{\bigwedge xyzfxfyz = ffxyz,$
$\bigwedge xy \bigvee zfxz = y,$
$\bigwedge xy \bigvee zfzx = y\}$
$\vdash \bigwedge xyuw(fxu = y \wedge fxw = y \rightarrow u = w)$

Para resolver el ejercicio hemos usado como demodulador que `e+x=x` y particularizado una de las hipótesis. El fichero de entrada es

```
set(para_from).
set(back_demod).
set(sos_queue).
op(500,xfy,+).
op(400,xf,'-').
formula_list(usable).
```

```
all x (x=x).
all x y z (x+(y+z)=(x+y)+z).
all x y (exists z (x + z = y)).
all x y (exists z (z + x = y)).
end_of_list.
formula_list(sos).
- (all x y u w (x+u=y & x+w=y ->
 u=w)).
all x (exists z (z + x = e)).
end_of_list.
list(demodulators).
x + y + z = (x + y) + z.
e + x = x.
end_of_list.
```

y el de salida es

```
-------> usable clausifies to:
1 [] x=x.
2 [] x+y+z= (x+y)+z.
3 [] x+$f1(x,y)=y.
4 [] $f2(x,y)+x=y.
-------> sos clausifies to:
5 [] $c4+$c2=$c3.
6 [] $c4+$c1=$c3.
7 [] $c2!=$c1.
8 [] $f3(x)+x=e.
9 [] axioma.
list(demodulators).
10 [] x+y+z= (x+y)+z.
11 [] e+x=x.
end_of_list.
-------- PROOF --------
2 [] x+y+z= (x+y)+z.
5 [] $c4+$c2=$c3.
6 [] $c4+$c1=$c3.
7 [] $c2!=$c1.
8 [] $f3(x)+x=e.
10 [] x+y+z= (x+y)+z.
11 [] e+x=x.
13 [para_from,5,2] (x+$c4)+$c2=x+$c3.
15 [para_from,6,2] (x+$c4)+$c1=x+$c3.
17 [para_from,8,2,demod,10,11] ($f3(x
   )+x)+y=y.
54,53 [para_from,17,15] $f3($c4)+$c3=
      $c1.
55 [para_from,17,13,demod,54] $c2=$c1
57 [binary,55,7] $F.
```

La demostración del Lema que justifica el uso del demodulador `e+x=x` se obtiene con las mismas reglas.

El Ejercicio 24 se soluciona utilizando UR–resolución, factorización, paramodulación y demodulación hacia atrás.

Ejercicio 25.
$\vdash \bigwedge x(Px \leftrightarrow x = a) \rightarrow \iota x Px = a$

Este es el primer ejercicio con descriptores. Para trabajar con el descriptor correspondiente al símbolo de predicado P hemos representado el término $\iota x Px$ por `el(x,P(x))` y hemos añadido los axiomas

$$(\bigvee y)(\bigwedge x)[Px \leftrightarrow x = y] \rightarrow (\bigwedge x)P(\iota x Px)$$
$$\neg(\bigvee y)(\bigwedge x)[Px \leftrightarrow x = y] \rightarrow (\bigwedge x)[\iota x Px = el\_total]$$

El fichero de entrada es

```
set(binary_res).
set(factor).
set(back_demod).
formula_list(usable).
all x (x = x).
(exists y all x (P(x)<->x=y)) ->
 (all x P(el(x,P(x)))).
- (exists y all x (P(x)<->x=y)) ->
 (all x (el(x,P(x))=el_total)).
end_of_list.
formula_list(sos).
- ((all x (P(x) <-> x = a)) ->  el(x,
 P(x)) = a).
end_of_list.
```

y el de salida es

```
-------> usable clausifies to:
1 [] x=x.
2 [] P($f1(y))|$f1(y)=y|P(el(x,P(x)))
3 [] -P($f1(y))|$f1(y)!=y|P(el(x,P(x)
     ).
     ).
4 [] -P(x)|x=$c1|el(x1,P(x1))=el_tota
     l.
5 [] P(x)|x!=$c1|el(x1,P(x1))=el_tota
     l.
-------> sos clausifies to:
6 [] -P(x)|x=a.
7 [] P(x)|x!=a.
8 [] el(x,P(x))!=a.
-------- PROOF --------
2 [] P($f1(y))|$f1(y)=y|P(el(x,P(x)))
3 [] -P($f1(y))|$f1(y)!=y|P(el(x,P(x)
```

```
     )).
 6 [] -P(x)|x=a.
 7 [] P(x)|x!=a.
 8 [] el(x,P(x))!=a.
 21 [binary,8,6] -P(el(x,P(x))).
 23 [binary,21,3] -P($f1(x))|$f1(x)!=x
 24 [binary,21,2] P($f1(x))|$f1(x)=x.
 25 [binary,23,7] $f1(x)!=x|$f1(x)!=a.
 27 [binary,23,6] -P($f1(a)).
 29 [factor,25] $f1(a)!=a.
 32 [binary,24,27] $f1(a)=a.
 34 [binary,32,29] $F.
```

Los Ejercicios 26–29, 31–33 y 35 se resuelven de forma análoga. El 34 se soluciona con resolución binaria, factorización y demodulación.

Ejercicio 30.
$\bigwedge x(Px \leftrightarrow x = \iota x Px) \leftrightarrow \bigvee y \bigwedge x(Px \leftrightarrow x = y)$

Para la demostración del bicondicional hemos introducido dos lemas correspondientes a cada una de las implicaciones. El fichero de entrada es

```
set(binary_res).
set(factor).
set(back_demod).
formula_list(usable).
all x (x = x).
(exists y all x (P(x)<->x=y)) ->
 (all x P(el(x,P(x)))).
- (exists y all x (P(x)<->x=y)) ->
 (all x (el(x,P(x))=el_total)).
(all z
 ((all x (P(x)<->x=el(z,P(z)))) ->
  (exists y (all x (P(x)<->x=y)))))
 <-> L1.
(all z
 ((exists y (all x (P(x)<->x=y))) ->
  (all x (P(x) <-> x = el(z,P(z))))))
 <-> L2.
end_of_list.
formula_list(sos).
- (L1 & L2).
end_of_list.
```

y el de salida es

```
-------> usable clausifies to:
1 [] x=x.
2 [] P($f1(y))|$f1(y)=y|P(el(x,P(x)))
3 [] -P($f1(y))|$f1(y)!=y|P(el(x,P(x)
     )).
4 [] -P(x)|x=$c1|el(x1,P(x1))=el_tota
5 [] P(x)|x!=$c1|el(x1,P(x1))=el_tota
6 [] -P(x)|x=el($c2,P($c2))|L1.
7 [] P(x)|x!=el($c2,P($c2))|L1.
8 [] P($f2(y))|$f2(y)=y|L1.
9 [] -P($f2(y))|$f2(y)!=y|L1.
10 [] P($f3(z))|$f3(z)=el(z,P(z))| -P
   (x2)|x2=$f4(z)| -L1.
11 [] P($f3(z))|$f3(z)=el(z,P(z))|P(x
   2)|x2!=$f4(z)| -L1.
12 [] -P($f3(z))|$f3(z)!=el(z,P(z))|
   -P(x2)|x2=$f4(z)| -L1.
13 [] -P($f3(z))|$f3(z)!=el(z,P(z))|P
   (x2)|x2!=$f4(z)| -L1.
14 [] -P(x)|x=$c3|L2.
15 [] P(x)|x!=$c3|L2.
16 [] P($c4)|$c4=el($c5,P($c5))|L2.
17 [] -P($c4)|$c4!=el($c5,P($c5))|L2.
18 [] P($f5(z,y))|$f5(z,y)=y| -P(x3)|
   x3=el(z,P(z))| -L2.
19 [] P($f5(z,y))|$f5(z,y)=y|P(x3)|x3
   !=el(z,P(z))| -L2.
20 [] -P($f5(z,y))|$f5(z,y)!=y| -P(x3
   )|x3=el(z,P(z))| -L2.
21 [] -P($f5(z,y))|$f5(z,y)!=y|P(x3)|
   x3!=el(z,P(z))| -L2.
end_of_list.
-------> sos clausifies to:
list(sos).
22 [] -L1| -L2.
-------- PROOF --------
2 [] P($f1(y))|$f1(y)=y|P(el(x,P(x)))
3 [] -P($f1(y))|$f1(y)!=y|P(el(x,P(x)
  )).
6 [] -P(x)|x=el($c2,P($c2))|L1.
7 [] P(x)|x!=el($c2,P($c2))|L1.
8 [] P($f2(y))|$f2(y)=y|L1.
9 [] -P($f2(y))|$f2(y)!=y|L1.
14 [] -P(x)|x=$c3|L2.
15 [] P(x)|x!=$c3|L2.
16 [] P($c4)|$c4=el($c5,P($c5))|L2.
17 [] -P($c4)|$c4!=el($c5,P($c5))|L2.
22 [] -L1| -L2.
23 [binary,22,9] -L2| -P($f2(x))|$f2(
   x)!=x.
24 [binary,22,8] -L2|P($f2(x))|$f2(x)
   =x.
82 [binary,23,6] -L2| -P($f2(el($c2,P
   ($c2))))|L1.
93 [binary,24,7] -L2|P($f2(el($c2,P($
```

```
   c2))))|L1.
112 [binary,82,22] -L2| -P($f2(el($c2
    ,P($c2)))).
127 [binary,93,112] -L2|L1.
140 [binary,127,22] -L2.
141 [binary,140,17] -P($c4)|el($c5,P(
    $c5))!=$c4.
142 [binary,140,16] P($c4)|el($c5,P($
    c5))=$c4.
143 [binary,140,15] P(x)|x!=$c3.
144 [binary,140,14] -P(x)|x=$c3.
151 [binary,143,2] P($f1($c3))|P(el(x
    ,P(x))).
176 [binary,144,3] -P($f1($c3))|P(el(
    x,P(x))).
193 [binary,141,143] el($c5,P($c5))!=
    $c4|$c4!=$c3.
197 [binary,142,144] el($c5,P($c5))=$
    c4|$c4=$c3.
208 [binary,176,151] P(el(x,P(x)))|P(
    el(y,P(y))).
213 [factor,208] P(el(x,P(x))).
215,214 [binary,213,144] el(x,P(x))=$
        c3.
222 [back_demod,197,demod,215] $c4=$c
    3.
226 [back_demod,193,demod,215] $c4!=$
    c3.
227 [binary,226,222] $F.
```

3 Conclusiones

Con este trabajo hemos comprobado que la resolución automática de los ejercicios de Mosterín requiere la utilización de muchas de las reglas de inferencia y estrategias del Razonamiento Automático. Además, hemos observado que OTTER no está desarrollado suficientemente para la utilización de los descriptores. Aunque hemos resuelto los ejercicios con descriptores, pensamos que sería conveniente estudiarlo más detenidamente e integrarlo en OTTER. Otra dificultad ha sido resolver los ejercicios relativos a la teoría de grupos, principalmente por los axiomas propuestos. Finalmente, destacamos la necesidad presentada en el Ejercicio 30 de descomponer el teorema en dos lemas y la posible conveniencia de extender OTTER para considerar razonamiento hacia atrás.

Referencias

1. McCune, W.W., *OTTER 2.0 User's Guide* . Tech. Report ANL–90/9, Argonne National Laboratory, Argonne, Ill., March 1990.

2. McCune, W.W., *What's New in Otter 2.2* . Tech. Report ANL/MCS–TM–153, Argonne National Laboratory, Argonne, Ill., July 1991.

3. Mosterín, J., *Lógica de primer orden* . Ariel, 1970.

4. Wos, L., *Automated Reasoning: 33 Basic Research Problems* , Prentice–Hall, 1987.