Razonamiento automático (2005–06)

Tema 2: Razonamiento proposicional con OTTER y MACE

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Sintaxis de la lógica proposicional

- El alfabeto proposicional:
  - *símbolos proposicionales*.
  - *conectivas lógicas*:
    $\neg$ (negación),
    $\wedge$ (conjunción),
    $\vee$ (disyunción),
    $\rightarrow$ (condicional),
    $\leftrightarrow$ (equivalencia).
  - símbolos auxiliares: "(" y ")".

- Sintaxis en OTTER/MACE

Usual	$\neg$	$\wedge$	$\vee$	$\rightarrow$	$\leftrightarrow$
OTTER/MACE	-	&	|	->	<->

- Las fórmulas proposicionales:
  - símbolos proposicionales
  - $\neg F$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$, $(F \leftrightarrow G)$.
- Eliminación de paréntesis:
  - Paréntesis externos.
  - Precedencia: $\neg, \wedge, \vee \rightarrow, \leftrightarrow$
  - Asociatividad: $\wedge$ y $\vee$ asocian por la derecha

Semántica: Verdad e interpretación

- Los valores de verdad:
  - verdadero 1 (en MACE , $\mathbf{T}$)
  - falso 0 (en MACE , $\mathbf{F}$)
- Las funciones de verdad:

i	$\neg i$
1	0
0	1

i	j	i $\wedge$ j	i $\vee$ j	i $\rightarrow$ j	i $\leftrightarrow$ j
1	1	1	1	1	1
1	0	0	1	0	0
0	1	0	1	1	0
0	0	0	0	1	1

Modelo de una fórmula

- El significado de una fórmula en una interpretación
  - Fórmula: $F = (p \vee q) \wedge (\neg q \vee r)$
  - Interpretación: $I, I(p) = I(r) = 1, I(q) = 0$

$$
\begin{array}{lllllll}
(p & \vee & q) & \wedge & (\neg q & \vee & r) \\
(1 & \vee & 0) & \wedge & (\neg 0 & \vee & 1) \\
 & 1 & & \wedge & (1 & \vee & 1) \\
 & 1 & & \wedge & & 1 & \\
 & & & 1 & & &
\end{array}
$$

  - F es válida en I
  - I es modelo de F
  - $I \models F$

Modelo de una fórmula

- Significado de una fórmula en una interpretación
  - Fórmula: $F = (p \vee q) \wedge (\neg q \vee r)$
  - Interpretación: J, $J(r) = 1, J(p) = J(q) = 0$

$$
\begin{array}{lllllll}
(p & \vee & q) & \wedge & (\neg q & \vee & r) \\
(0 & \vee & 0) & \wedge & (\neg 0 & \vee & 1) \\
 & 0 & & \wedge & (1 & \vee & 1) \\
 & 0 & & \wedge & & 1 & \\
 & & & 0 & & &
\end{array}
$$

  - F no es válida en J
  - J no es modelo de F
  - $J \not\models F$

Comprobación de modelo con MACE

- Determinar si la la interpretación I definida por $I(p) = I(r) = 1, I(q) = 0$ es un modelo de la fórmula $(p \vee q) \wedge (\neg q \vee r)$
- Formalización en MACE "**modelo_fla_1.in**"

```
1  formula_list(usable).
2   (p | q) & (-q | r).
3  end_of_list.
4
5  list(mace_constraints).
6   assign(p,T).
7   assign(q,F).
8   assign(r,T).
9  end_of_list.
```

- Ejecución en MACE :

```
mace2 -p < modelo_fla_1.in > modelo_fla_1.out
```

Comprobación de modelo con MACE

- Determinar si la la interpretación J definida por $J(r) = 1$, $J(p) = 0$ y $J(q) = 0$ es un modelo de la fórmula $(p \vee q) \wedge (\neg q \vee r)$
- Formalización en MACE — **"modelo_fla_2.in"**

```
1  formula_list(usable).
2   (p | q) & (-q | r).
3  end_of_list.
4
5  list(mace_constraints).
6   assign(p,F).
7   assign(q,F).
8   assign(r,T).
9  end_of_list.
```

- Ejecución en MACE :

```
mace2 -p < modelo_fla_2.in > modelo_fla_2.out
```

Fórmulas satisfacibles e insatisfacibles

- Def.: F es satisfacible syss F tiene modelo
- Def.: F es insatisfacible syss F no tiene modelo
- Ejemplo:
  $(p \rightarrow q) \wedge (q \rightarrow r)$ es satisfacible
  $$I(p) = I(q) = I(r) = 0$$
  $p \wedge \neg p$ es insatisfacible

p	$\neg p$	$p \wedge \neg p$
1	0	0
0	1	0

Satisfacibilidad con MACE

- Determinar si la fórmula $(p \rightarrow q) \wedge (q \rightarrow r)$ es satisfacible
- Formalización en MACE **"satisfacibilidad_1.in"**

```
1  formula_list(sos).
2   (p -> q) & (q -> r).
3  end_of_list.
```

- Ejecución en MACE :

```
mace2 -p < satisfacibilidad_1.in
         > satisfacibilidad_1.out
```

Satisfacibilidad con MACE

- Determinar si la fórmula $p \wedge \neg p$ es satisfacible
- Formalización en MACE **"satisfacibilidad_2.in"**

```
1  formula_list(sos).
2   p & -p.
3  end_of_list.
```

- Ejecución en MACE :

```
mace2 -p < satisfacibilidad_2.in
         > satisfacibilidad_2.out
```

Fórmulas válidas

- Def.: F es válida syss toda interpretación de F es modelo de F
- Fórmula: $F = (p \rightarrow q) \vee (q \rightarrow p)$

p	q	$(p \rightarrow q)$	$(q \rightarrow p)$	$(p \rightarrow q) \vee (q \rightarrow p)$
1	1	1	1	1
1	0	0	1	1
0	1	1	0	1
0	0	1	1	1

- $\models F$
- $\not\models (p \rightarrow q)$

Satisfacibilidad y validez

- Problemas de satisfacibilidad y validez
  - El problema de la satisfacibilidad: Dada F determinar si es satisfacible.
  - El problema de la validez: Dada F determinar si es válida.
- Relaciones entre validez y satisfacibilidad:
  - F es válida $\iff$ $\neg F$ es insatisfacible
  - F es válida $\implies$ F es satisfacible
  - F es satisfacible $\not\iff$ $\neg F$ es insatisfacible
- El problema de la satisfacibilidad es NP–completo

Validez con MACE

- Determinar si la fórmula $(p \rightarrow q) \vee (q \rightarrow p)$ es válida
- Formalización en MACE

"validez_1.in"

```
1  formula_list(usable).
2   -((p -> q) | (q -> p)).
3  end_of_list.
```

- Ejecución en MACE :

```
mace2 -p < validez_1.in > validez_1.out
```

Validez con MACE

- Determinar si la fórmula $p \rightarrow q$ es válida
- Formalización en MACE

"validez_2.in"

```
1  formula_list(usable).
2   -(p -> q).
3  end_of_list.
```

- Ejecución en MACE :

```
mace2 -p < validez_2.in > validez_2.out
```

Modelos de un conjunto de fórmulas

- Def.: I es modelo de S syss para toda $F \in S, I \models F$
- $I \models S$
- Ejemplo:
  $S = \{(p \vee q) \wedge (\neg q \vee r), q \rightarrow r\}$
  $I_1(p) = 1, I_1(q) = 0, I_1(r) = 1 \Longrightarrow I_1 \models S$
  $I_2(p) = 0, I_2(q) = 1, I_2(r) = 0 \Longrightarrow I_2 \not\models S$

Comprobación de modelos con MACE

- Determinar si la interpretación I definida por $I(p) = 1$,$I(q) = 0$, $I(r) = 1$ es un modelo del conjunto de fórmulas $S = \{(p \vee q) \wedge (\neg q \vee r), q \rightarrow r\}$
- Formalización en MACE

"modelo_cjt_1.in"

```
1  formula_list(usable).
2   (p | q) & (-q | r).
3   q -> r.
4  end_of_list.
5
6  list(mace_constraints).
7   assign(p,T).
8   assign(q,F).
9   assign(r,T).
10 end_of_list.
```

Comprobación de modelos con MACE

- Determinar si la interpretación I definida por $I(p) = 0$,$I(q) = 1$, $I(r) = 0$ es un modelo del conjunto de fórmulas $S = \{(p \vee q) \wedge (\neg q \vee r), q \rightarrow r\}$
- Formalización en MACE **"modelo_cjt_2.in"**

```
1  formula_list(usable).
2   (p | q) & (-q | r).
3   q -> r.
4  end_of_list.
5
6  list(mace_constraints).
7   assign(p,F).
8   assign(q,T).
9   assign(r,F).
10 end_of_list.
```

Conjuntos consistentes

- Def.: S es consistente syss S tiene modelo
- Def.: S es inconsistente syss S no tiene modelo
- Ejemplo: $\{(p \vee q) \wedge (-q \vee r), p \rightarrow r\}$ es consistente
- Ejemplo: $\{(p \vee q) \wedge (-q \vee r), p \rightarrow r, \neg r\}$ es inconsistente:

	p	q	r	$(p \vee q)$	$(\neg q \vee r)$	$(p \vee q) \wedge (\neg q \vee r)$	$p \rightarrow r$	$\neg r$
I_1	0	0	0	0	1	0	1	1
I_2	0	0	1	0	1	0	1	0
I_3	0	1	0	1	0	0	1	1
I_4	0	1	1	1	1	1	1	0
I_5	1	0	0	1	1	1	0	1
I_6	1	0	1	1	1	1	1	0
I_7	1	1	0	1	0	0	0	1
I_8	1	1	1	1	1	1	1	0

Consistencia con MACE

- Determinar si el conjunto $\{(p \vee q) \wedge (-q \vee r), p \rightarrow r\}$ es consistente
- Formalización en MACE

"consistencia.in"

```
1  formula_list(sos).
2   (p | q) & (-q | r).
3   p -> r.
4  end_of_list.
```

Consistencia con MACE

- Determinar si el conjunto $\{(p \vee q) \wedge (-q \vee r), p \rightarrow r, \neg r\}$ inconsistente
- Formalización en MACE **"inconsistencia.in"**

```
1  formula_list(sos).
2   (p | q) & (-q | r).
3   p -> r.
4   -r.
5  end_of_list.
```

Consecuencia lógica

- F es consecuencia de S syss todos los modelos de S son modelos de F
- Representación: $S \models F$
  - Ejemplo: $\{p \rightarrow q, q \rightarrow r\} \models p \rightarrow r$

	p	q	r	$p \rightarrow q$	$q \rightarrow r$	$p \rightarrow r$
I_1	0	0	0	1	1	1
I_2	0	0	1	1	1	1
I_3	0	1	0	1	0	1
I_4	0	1	1	1	1	1
I_5	1	0	0	0	1	0
I_6	1	0	1	0	1	1
I_7	1	1	0	1	0	0
I_8	1	1	1	1	1	1

- Ejemplo: $\{p\} \not\models p \wedge q$

p	q	$p \wedge q$
1	1	1
1	0	0
0	1	0
0	0	0

Consecuencia, validez y consistencia

- Las siguientes condiciones son equivalentes:
  - $\{F_1, \ldots, F_n\} \models G$
  - $\models F_1 \wedge \ldots \wedge F_n \rightarrow G$
  - $\neg(F_1 \wedge \ldots \wedge F_n \rightarrow G)$ es insatisfacible
  - $\{F_1, \ldots, F_n, \neg G\}$ es inconsistente

Consecuencia lógica con MACE

- Determinar si $\{p \leftrightarrow q, r \leftrightarrow (p \wedge q)\} \not\models p \wedge r$
- Formalización en MACE **"no-consecuencia.in"**

```
1  formula_list(sos).
2   p <->q.
3   r <-> (p & q).
4   -(p & r).
5  end_of_list.
```

Consecuencia lógica con MACE

- Determinar si $\{p \leftrightarrow q, r \leftrightarrow (p \wedge q)\} \models p \leftrightarrow r$
- Formalización en MACE

"consecuencia.in"

```
1  formula_list(sos).
2   p <->q.
3   r <-> (p & q).
4   -(p <-> r).
5   end_of_list.
```

Problema de los animales con MACE

- Base de conocimiento
  - Base de reglas:
    1. Si el animal tiene pelos es mamífero
    2. Si el animal da leche es mamífero
    3. Si el animal es un mamífero y tiene pezuñas es ungulado
    4. Si el animal es un mamífero y rumia es ungulado
    5. Si el animal es un ungulado y tiene cuello largo es una jirafa
    6. Si el animal es un ungulado y tiene rayas negras es una cebra
  - Base de hechos:
    1. El animal tiene pelos
    2. El animal tiene pezuñas
    3. El animal tiene rayas negras
  - Consecuencia:

    El animal es una cebra

Problema de los animales con MACE

- Formalización en MACE

"animales_1.in"

```
formula_list(sos).
tiene_pelos | da_leche -> es_mamifero.
es_mamifero & (tiene_pezuñas | rumia) -> es_ungulado.
es_ungulado & tiene_cuello_largo -> es_jirafa.
es_ungulado & tiene_rayas_negras -> es_cebra.

tiene_pelos & tiene_pezuñas & tiene_rayas_negras.

-es_cebra.
end_of_list.
```

Problema de los animales con MACE

- Modelo del problema de los animales
- Formalización en MACE

"animales_2.in"

```
formula_list(sos).
tiene_pelos | da_leche -> es_mamifero.
es_mamifero & (tiene_pezuñas | rumia) -> es_ungulado.
es_ungulado & tiene_cuello_largo -> es_jirafa.
es_ungulado & tiene_rayas_negras -> es_cebra.

tiene_pelos & tiene_pezuñas & tiene_rayas_negras.

% -es_cebra.
end_of_list.
```

Problema del coloreado de pentágonos

- Demostrar que es imposible colorear los vértices de un pentágono de rojo o azul de forma que los vértices adyacentes tengan colores distintos
- Representación: Se numeran los vértices consecutivos del pentágono con los números 1, 2, 3, 4 y 5. Se usan los símbolos r_i $(1 \leq i \leq 5)$ para representar que el vértice i es rojo y los símbolos a_j $(1 \leq j \leq 5)$ para representar que el vértice j es azul

Problema del coloreado de pentágonos

- Formalización en MACE

”pentagono_2.in”

```
formula_list(usable).
% El vértice i (1 <= i <= 5) es azul o rojo:
a1 | r1.        a2 | r2.        a3 | r3.
a4 | r4.        a5 | r5.
% Dos vértices adyacentes no pueden ser azules:
-(a1 & a2).     -(a2 & a3).     -(a3 & a4).
-(a4 & a5).     -(a5 & a1).
% Dos vértices adyacentes no pueden ser rojos:
-(r1 & r2).     -(r2 & r3).     -(r3 & r4).
-(r4 & r5).     -(r5 & r1).
end_of_list.
```

Problema del coloreado de pentágonos

- Demostrar que es posible colorear los vértices de un pentágono de rojo, azul o negro de forma que los vértices adyacentes tengan colores distintos
- Representación: Se numeran los vértices consecutivos del pentágono con los números 1, 2, 3, 4 y 5. Se usan los símbolos r_i $(1 \leq i \leq 5)$ para representar que el vértice i es rojo, los símbolos a_j $(1 \leq j \leq 5)$ para representar que el vértice j es azul y los símbolos n_j $(1 \leq j \leq 5)$ para representar que el vértice j es negro

Problema del coloreado de pentágonos

- Formalización en MACE

"pentagono_3.in"

```
formula_list(usable).
% El vértice i (1 <= i <= 5) es azul, rojo o negro:
a1 | r1 | n1.      a2 | r2 | n2.     a3 | r3 | n3.
a4 | r4 | n4.      a5 | r5 | n5.
% Dos vértices adyacentes no pueden ser azules:
-(a1 & a2).        -(a2 & a3).       -(a3 & a4).
-(a4 & a5).        -(a5 & a1).
% Dos vértices adyacentes no pueden ser rojos:
-(r1 & r2).        -(r2 & r3).       -(r3 & r4).
-(r4 & r5).        -(r5 & r1).
% Dos vértices adyacentes no pueden ser negros:
-(n1 & n2).        -(n2 & n3).       -(n3 & n4).
-(n4 & n5).        -(n5 & n1).
end_of_list.
```

Otros problemas

- Problema de las reinas.
- Problema del Sudoku.

Razonamiento proposicional con OTTER

- Base de conocimiento
  - Base de reglas:

    R1 Si el animal tiene pelos es mamífero

    R2 Si el animal da leche es mamífero

    R3 Si el animal es un mamífero y tiene pezuñas es ungulado

    R4 Si el animal es un mamífero y rumia es ungulado

    R5 Si el animal es un ungulado y tiene cuello largo es una jirafa

    R6 Si el animal es un ungulado y tiene rayas negras es una cebra
  - Base de hechos:

    H1 El animal tiene pelos

    H2 El animal tiene pezuñas

    H3 El animal tiene rayas negras
  - Consecuencia:

    C El animal es una cebra

Razonamiento proposicional con OTTER

- Formalización en OTTER

"animales_1.in"

```
formula_list(sos).
 tiene_pelos | da_leche -> es_mamifero.
 es_mamifero & (tiene_pezuñas | rumia) -> es_ungulado.
 es_ungulado & tiene_cuello_largo -> es_jirafa.
 es_ungulado & tiene_rayas_negras -> es_cebra.
 tiene_pelos & tiene_pezuñas & tiene_rayas_negras.
 -es_cebra.
end_of_list.

set(binary_res).
set(very_verbose).
```

- Ejecución en OTTER :

```
otter < animales_1.in > animales_1.out
```

Preprocesamiento

”animales_1.out”

```
-------> sos clausifies to:

list(sos).
1 [] -tiene_pelos|es_mamifero.
2 [] -da_leche|es_mamifero.
3 [] -es_mamifero| -tiene_pezugnas|es_ungulado.
4 [] -es_mamifero| -rumia|es_ungulado.
5 [] -es_ungulado| -tiene_cuello_largo|es_jirafa.
6 [] -es_ungulado| -tiene_rayas_negras|es_cebra.
7 [] tiene_pelos.
8 [] tiene_pezugnas.
9 [] tiene_rayas_negras.
10 [] -es_cebra.
end_of_list.
```

Transformación a cláusulas

- Una cláusula es una disyunción de literales.
- Un literal es un átomo o la negación de un átomo.
- Procedimiento de transformación a cláusulas:

1. Eliminar las equivalencias usando la relación

$$A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A) \quad (1)$$

2. Eliminar las implicaciones usando la equivalencia

$$A \rightarrow B \equiv \neg A \vee B \quad (2)$$

3. Interiorizar las negaciones usando las equivalencias

$$\neg(A \wedge B) \equiv \neg A \vee \neg B \quad (3)$$

$$\neg(A \vee B) \equiv \neg A \wedge \neg B \quad (4)$$

$$\neg\neg A \equiv A \quad (5)$$

4. Interiorizar las disyunciones usando las propiedades distributivas

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C) \quad (6)$$

$$(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C) \quad (7)$$

Características activas

- Reglas dependientes:

”animales_1.out”

```
set(binary_res).
   dependent: set(factor).
   dependent: set(unit_deletion).
```

- Resolución binaria: `set(binary_res).`
- Factorización: `set(factor).`
- Eliminación unitaria: `set(unit_deletion).`

Búsqueda de la prueba

”animales_1.out”

```
========== start of search ==========
given clause #1: (wt=1) 7 [] tiene_pelos.
given clause #2: (wt=1) 8 [] tiene_pezugnas.
given clause #3: (wt=1) 9 [] tiene_rayas_negras.
given clause #4: (wt=1) 10 [] -es_cebra.
given clause #5: (wt=2) 1 [] -tiene_pelos|es_mamifero.
  0 [binary,1.1,7.1] es_mamifero.
** KEPT (pick-wt=1): 11 [binary,1.1,7.1] es_mamifero.
11 back subsumes 2.
11 back subsumes 1.
...
given clause #11: (wt=3) 6 [] -es_ungulado| -tiene_rayas_negras|es_cebra.
  0 [binary,6.1,12.1] -tiene_rayas_negras|es_cebra.
** KEPT (pick-wt=0): 14 [binary,6.1,12.1,unit_del,9,10] $F.
--> EMPTY CLAUSE at 0.01 sec --> 14 [binary,6.1,12.1,unit_del,9,10] $F.
```

Procedimiento de búsqueda de pruebas

- Mientras el soporte es no vacío y no se ha encontrado una refutación
  1. Seleccionar como cláusula actual la cláusula menos pesada del soporte.
  2. Mover la cláusula actual del soporte a usable.
  3. Calcular las resolventes de la cláusula actual con las cláusulas usables.
  4. Procesar cada una de las resolventes calculadas anteriormente.
  5. Añadir al soporte cada una de las cláusulas procesadas que supere el procesamiento.
  *6. Descartar cada cláusula de usable o del soporte subsumida por alguna resolvente (subsunción hacia atrás).

Procesamiento de las resolventes

*1 Escribir la resolvente.

*2 Aplicar a la resolvente eliminación unitaria (elimina los literales de la resolvente tales que hay una cláusula unitaria complementaria en usable o en soporte).

3 Si la resolvente es una tautología se descarta.

4 Si la resolvente es subsumida por alguna cláusula de usable o del soporte (subsunción hacia delante) se descarta.

5 Añadir la resolvente al soporte.

*6 Escribir la resolvente retenida.

7 Si la resolvente tiene 0 literales, se ha encontrado una refutación.

8 Si la resolvente tiene 1 literal, entonces buscar su complementaria (refutación) en usable y soporte.

*9 Escribir la demostración si se ha encontrado una refutación.

El paso 10 no se da hasta que los pasos 1-9 se han aplicado a todas las resolventes.

Cláusulas usadas

```
 1 [] -tiene_pelos | es_mamifero.
 2 [] -da_leche | es_mamifero.
 3 [] -es_mamifero | -tiene_pezugnas |es_ungulado.
 4 [] -es_mamifero | -rumia | es_ungulado.
 5 [] -es_ungulado | -tiene_cuello_largo | es_jirafa.
 6 [] -es_ungulado | -tiene_rayas_negras |es_cebra.
 7 [] tiene_pelos.
 8 [] tiene_pezugnas.
 9 [] tiene_rayas_negras.
10 [] -es_cebra.
11 [binary,1.1,7.1] es_mamifero.
12 [binary,3.1,11.1,unit_del,8] es_ungulado.
13 [binary,5.1,12.1] -tiene_cuello_largo|es_jirafa.
14 [binary,6.1,12.1,unit_del,9,10] $F.
```

Evolución del soporte y usable

N	Soporte	Usable
0	1 2 3 4 5 6 7 8 9 10	
1	1 2 3 4 5 6 8 9 10	7
2	1 2 3 4 5 6 9 10	8 7
3	1 2 3 4 5 6 10	9 8 7
4	1 2 3 4 5 6	10 9 8 7
5	3 4 5 6 11	10 9 8 7
6	3 4 5 6	11 10 9 8 7
7	5 6 12	11 10 9 8 7
8	5 6	12 11 10 9 8 7
9	6 13	12 11 10 9 8 7
10	6	13 12 11 10 9 8 7
11	14	6 13 12 11 10 9 8 7

Demostración

”animales_1.out”

```
Length of proof is 2.  Level of proof is 2.
---------------- PROOF ----------------
1 [] -tiene_pelos|es_mamifero.
3 [] -es_mamifero| -tiene_pezugnas|es_ungulado.
6 [] -es_ungulado| -tiene_rayas_negras|es_cebra.
7 [] tiene_pelos.
8 [] tiene_pezugnas.
9 [] tiene_rayas_negras.
10 [] -es_cebra.
11 [binary,1.1,7.1] es_mamifero.
12 [binary,3.1,11.1,unit_del,8] es_ungulado.
14 [binary,6.1,12.1,unit_del,9,10] $F.
------------ end of proof -------------
```

Estadísticas

”animales_1.out”

```
------------- statistics ------------
clauses given                    11
clauses generated                 5
  binary_res generated            5
  factors generated               0
clauses forward subsumed          1
  (subsumed by sos)               1
unit deletions                    4
clauses kept                      3
empty clauses                     1
clauses back subsumed             5
usable size                       8
----------- times (seconds) -----------
user CPU time          0.00          (0 hr, 0 min, 0 sec)
system CPU time        0.01          (0 hr, 0 min, 0 sec)
```

Redistribución del soporte

”animales_2.in”

```
formula_list(usable).
 tiene_pelos | da_leche -> es_mamifero.
 es_mamifero & (tiene_pezuñas | rumia) -> es_ungulado.
 es_ungulado & tiene_cuello_largo -> es_jirafa.
 es_ungulado & tiene_rayas_negras -> es_cebra.

 tiene_pelos & tiene_pezuñas & tiene_rayas_negras.
end_of_list.

formula_list(sos).
 -es_cebra.
end_of_list.

set(binary_res).
set(very_verbose).
```

Nueva demostración

”animales_2.out”

```
Length of proof is 3.  Level of proof is 3.
---------------- PROOF ----------------
1 [] -tiene_pelos|es_mamifero.
3 [] -es_mamifero| -tiene_pezugnas|es_ungulado.
6 [] -es_ungulado| -tiene_rayas_negras|es_cebra.
7 [] tiene_pelos.
8 [] tiene_pezugnas.
9 [] tiene_rayas_negras.
10 [] -es_cebra.
11 [binary,10.1,6.3,unit_del,9] -es_ungulado.
13 [binary,11.1,3.3,unit_del,8] -es_mamifero.
15 [binary,13.1,1.2] -tiene_pelos.
16 [binary,15.1,7.1] $F.
```

Eliminación de tautologías y factorización

”tautologias_factorizacion.in”

```
list(sos).
-p | -q.
p | q.
p | -q.
-p | q.
end_of_list.

set(binary_res).
set(very_verbose).
```

Búsqueda de la prueba

”tautologias_factorizacion.out”

```
given clause #1: (wt=2) 1 [] -p| -q.
given clause #2: (wt=2) 2 [] p|q.
  0 [binary,2.1,1.1] q| -q.
  0 [binary,2.2,1.2] p| -p.
given clause #3: (wt=2) 3 [] p| -q.
  0 [binary,3.1,1.1] -q| -q.
** KEPT (pick-wt=1): 5 [binary,3.1,1.1,factor_simp] -q.
  0 [binary,3.2,2.2] p|p.
** KEPT (pick-wt=1): 6 [binary,3.2,2.2,factor_simp] p.
5 back subsumes 3.
5 back subsumes 1.
6 back subsumes 2.
given clause #4: (wt=1) 5 [binary,3.1,1.1,factor_simp] -q.
given clause #5: (wt=1) 6 [binary,3.2,2.2,factor_simp] p.
given clause #6: (wt=2) 4 [] -p|q.
  0 [binary,4.1,6.1] q.
** KEPT (pick-wt=1): 7 [binary,4.1,6.1] q.
----> UNIT CONFLICT at   0.00 sec ----> 8 [binary,7.1,5.1] $F.
```

Demostración

”tautologias_factorizacion.out”

```
---------------- PROOF ----------------
1 [] -p| -q.
2 [] p|q.
3 [] p| -q.
4 [] -p|q.
5 [binary,3.1,1.1,factor_simp] -q.
6 [binary,3.2,2.2,factor_simp] p.
7 [binary,4.1,6.1] q.
8 [binary,7.1,5.1] $F.
------------ end of proof -------------
```

Bibliografía

- J.A. Alonso y J. Borrego
  Deducción automática (Vol. 1: Construcción lógica de sistemas lógicos) (Ed. Kronos, 2002)
- Bundy, A. *The computer modelling of mathematical reasoning*. (Academic Press, 1983)
  - Cap. 2 “Arguments about propositions”
  - Cap. 3: “The internal structure of propositions”
- Chang, C.L. y Lee, R.C.T *Symbolic logic and mechanical theorem proving*. (Academic Press, 1973)
  - Cap. 2 “The propositional logic”
- Genesereth, M.R. *Computational logic*
  - Cap. 2 “Propositional logic”
  - Cap. 3 “Truth table method”
- Nilsson, N.J. *Inteligencia artificial (Una nueva síntesis)*. (McGraw–Hill, 2000)
  - Cap. 13 “El cálculo proposicional”
- McCune, W. OTTER *3.3 Reference Manual* (2003)