Lógica informática (2012–13)

Tema 6: Algoritmos para SAT. Aplicaciones

José A. Alonso Jiménez
Andrés Cordón Franco
María J. Hidalgo Doblado

Grupo de Lógica Computacional
Departamento de Ciencias de la Computación e I.A.
Universidad de Sevilla

Tema 6: Algoritmos para SAT. Aplicaciones

1. Algoritmos para SAT

2. Aplicaciones

Tema 6: Algoritmos para SAT. Aplicaciones

1. Algoritmos para SAT
   - Equiconsistencia
   - Eliminación de tautologías
   - Eliminación unitaria
   - Eliminación de literales puros
   - Regla de división
   - Algoritmo DPLL

2. Aplicaciones

Equiconsistencia

- Notación:
  - Literales: $L, L', L_1, L_2, \ldots$
  - Cláusulas: $C, C', C_1, C_2, \ldots$
  - Conjuntos de cláusulas: $S, S', S_1, S_2, \ldots$
- Def. S y S' son equiconsistentes (y se representa por $S \approx S'$) si S es consistente syss S' es consistente
- Ejemplos:
  - $\{\{p\}\} \approx \{\{p\}, \{q\}\}$
  - $\{\{p\}\} \not\approx \{\{p\}, \{\neg p\}\}$

Eliminación de tautologías

- Prop.: Sean S un conjunto de cláusulas y $C \in S$ una tautología. Entonces, $S \approx S \setminus \{C\}$.
- Ejemplo: $\{\{p, q\}, \{p, q, \neg p\}\} \approx \{\{p, q\}\}$.

Eliminación unitaria

- Def.: Una cláusula C es unitaria si C tiene sólo un literal.
- Def.: Sean S un conjunto de cláusulas y $\{L\}$ una cláusula unitaria de S. Entonces la eliminación unitaria de L en S es el conjunto obtenido borrando en S todas las cláusulas que contienen L y borrando L^c en todas las cláusulas; es decir,
$$\mathit{elimacionUnitaria}(S, L) = \{C - \{L^c\} : C \in S, L \notin C\}$$
- Ejemplo: $\mathit{elimacionUnitaria}(\{\{p, q\}, \{p, \neg q\}, \{\neg p\}, \{r\}\}, \neg p) = \{\{q\}, \{\neg q\}, \{r\}\}$
- Prop.: Sean S un conjunto de cláusulas y $\{L\}$ una cláusula unitaria de S. Entonces $S \approx \mathit{eliminacionUnitaria}(S, L)$.
- Ejemplos:
$$\begin{aligned} & \{\{p, q, \neg r\}, \{p, \neg q\}, \{\neg p\}, \{r, u\}\} \\ \approx\ & \{\{q, \neg r\}, \{\neg q\}, \{r, u\}\} \\ \approx\ & \{\{\neg r\}, \{r, u\}\} \\ \approx\ & \{\{u\}\} \end{aligned}$$

Eliminación de literales puros

- Def.: L es un literal puro de S si S es un literal de alguna cláusula de S y el complementario de L no pertenece a las cláusulas de S.
- Ejemplos:
  - p es un literal puro de $\{\{p, q\}, \{p, \neg q\}, \{r, q\}, \{r, \neg q\}\}$.
  - q no es un literal puro de $\{\{p, q\}, \{p, \neg q\}, \{r, q\}, \{r, \neg q\}\}$.
- Def.: Sean S un conjunto de cláusulas. Entonces la eliminación del literal puro L de S es el conjunto obtenido borrando en S las cláusulas que tienen L; es decir,

$$elimacionPuro(S, L) = \{C \in S : L \notin C\}$$

- Prop.: Sean S un conjunto de cláusulas y L un literal puro de S. Entonces $S \approx eliminacionPuro(S, L)$.
- Ejemplo:

$$\begin{aligned} & \{\{p, q\}, \{p, \neg q\}, \{r, q\}, \{r, \neg q\}\} \\ \approx\ & \{\{r, q\}, \{r, \neg q\}\} \\ \approx\ & \{\} \end{aligned}$$

Regla de división

- Prop.: Sea L un literal del conjunto de cláusulas S. Entonces, son equivalentes
  1. S es consistente,
  2. $(S \cup \{\{L\}\})$ ó $(S \cup \{\{L^c\}\})$ es consistente,
  3. $elimacionUnitaria(S, L)$ ó $elimacionUnitaria(S, L^c)$ es consistente,

El algoritmo DPLL (Davis, Putnam, Logemann y Loveland)

- Entrada: Un conjunto de cláusulas S.
- Salida: "Consistente", si S es consistente e "Inconsistente", en caso contrario.
- Procedimiento $DPLL(S)$
  1. Si $\square \in S$, entonces "Inconsistente".
  2. Si $S = \emptyset$, entonces "Consistente".
  3. Si S tiene alguna cláusula unitaria $\{L\}$, entonces $DPLL(eliminacionUnitaria(S, L)$.
  4. Si S tiene algún literal puro L, entonces $DPLL(eliminacionPuro(S, L))$.
  5. Sea L un literal de S.
     - Si $DPLL(S \cup \{L\})) =$ "Consistente", entonces devolver "Consistente";
     - en caso contrario $DPLL(S \cup \{L^c\})$.

Ejemplo del algoritmo DPLL

$$
\begin{aligned}
S &= \{\{a,b\},\{\neg a,b\},\{a,\neg b\},\{a,\neg d\},\{\neg a,\neg b,\neg c\},\{b,\neg c\},\{c,\neg f\},\{f\}\} \\
&\quad [\text{Unitaria } \{f\}] \\
&\approx \{\{a,b\},\{\neg a,b\},\{a,\neg b\},\{a,\neg d\},\{\neg a,\neg b,\neg c\},\{b,\neg c\},\{c\}\} \\
&\quad [\text{Unitaria } \{c\}] \\
&\approx \{\{a,b\},\{\neg a,b\},\{a,\neg b\},\{a,\neg d\},\{\neg a,\neg b\},\{b\}\} \\
&\quad [\text{Unitaria } \{b\}] \\
&\approx \{\{a\},\{a,\neg d\},\{\neg a\}\} \\
&\quad [\text{Unitaria } \{\neg a\}] \\
&\approx \{\square,\{\neg d\}\}
\end{aligned}
$$

Por tanto, S es inconsistente.

Ejemplo del algoritmo DPLL

$$
\begin{array}{lll}
S & = \{\{p, q\}, \{\neg q\}, \{\neg r\}\} & [\text{Unitaria } \{\neg q\}] \\
& \approx \{\{p\}, \{\neg r\}\} & [\text{Puro } p] \\
& \approx \{\{\neg r\}\} & [\text{Puro } \neg r] \\
& \emptyset &
\end{array}
$$

Por tanto

- S es consistente.
- Un modelo es I con $I(p) = 1$, $I(q) = 0$ e $I(r) = 0$.

Ejemplo del algoritmo DPLL

Sea $S = \{\{\neg q, r\}, \{\neg r, p\}, \{\neg r, q\}, \{\neg p, q, r\}, \{p, q\}, \{\neg p, \neg q\}\}$ No tiene cláusulas unitarias ni literales puros. Elegimos el literal p para dividir los casos.

- Primer caso:
  $S \cup \{p\}$
  $= \{\{\neg q, r\}, \{\neg r, p\}, \{\neg r, q\}, \{\neg p, q, r\}, \{p, q\}, \{\neg p, \neg q\}, \{p\}\}$ [Unit. $\{p\}$]
  $\approx \{\{\neg q, r\}, \{\neg r, q\}, \{q, r\}, \{\neg q\}\}$ [Unit. $\{\neg q\}$]
  $\approx \{\{\neg r\}, \{r\}\}$ [Unit. $\{r\}$]
  $\approx \{\square\}$
- Segundo caso:
  $S \cup \{\neg p\}$
  $= \{\{\neg q, r\}, \{\neg r, p\}, \{\neg r, q\}, \{\neg p, q, r\}, \{p, q\}, \{\neg p, \neg q\}, \{\neg p\}\}$ [Unit. $\{\neg p\}$]
  $\approx \{\{\neg q, r\}, \{\neg r\}, \{\neg r, q\}, \{q\}\}$ [Unit. $\{\neg r\}$]
  $\approx \{\{\neg q\}, \{q\}\}$ [Unit. $\{\neg q\}$]
  $\approx \{\square\}$

Por tanto, S es inconsistente.

Tema 6: Algoritmos para SAT. Aplicaciones

1. Algoritmos para SAT

2. Aplicaciones
   - Sobre Prover9 y Mace4
   - El problema de los veraces y los mentirosos
   - El problema de los animales
   - El problema del coloreado del pentágono
   - El problema del palomar
   - El problema de los rectángulos
   - El problema de las 4 reinas
   - El problema de Ramsey

Sobre Prover9 y Mace4

- Prover9 es un demostrador automático para la lógica de primer orden.
- Mace4 un calculador de modelos.
- Prover9 y Mace4 son libres y se encuentran en `http://www.cs.unm.edu/~mccune/mace4`
- Sintaxis (como la de APLI2):

Usual	$\neg$	$\wedge$	$\vee$	$\rightarrow$	$\leftrightarrow$
Prover9/Mace4	-	&	|	->	<->

El problema de mentirosos

- Enunciado: En una isla hay dos tribus, la de los veraces (que siempre dicen la verdad) y la de los mentirosos (que siempre mienten). Un viajero se encuentra con tres isleños A, B y C y cada uno le dice una frase
  - A dice "B y C son veraces syss C es veraz"
  - B dice "Si A y B son veraces, entonces B y C son veraces y A es mentiroso"
  - C dice "B es mentiroso syss A o B es veraz"

  Determinar a qué tribu pertenecen A, B y C.
- Simbolización:
  - `a`, `b` y `c` representan que A, B y C son veraces
  - `-a`, `-b` y `-c` representan que A, B y C son mentirosos

El problema los mentirosos

- Idea: las tribus se determinan a partir de los modelos del conjunto de fórmulas correspondientes a las tres frases.
- Representación en Mace4 (`pb_mentirosos.in`)

```
formulas(assumptions).
 a <-> (b & c <-> c).
 b <-> (a & c -> b & c & -a).
 c <-> (-b <-> a | b).
end_of_list.
```

El problema de los mentirosos

- Cálculo de modelos con Mace4

```
> mace4 -N2 -m9 -p1 <pb_mentirosos.in
 a : 1
 b : 1
 c : 0
```

- Conclusión: A y B son veraces y C es mentiroso.

El problema de los mentirosos

- Representación en Prover9 (`pb_mentirosos_2.in`)

```
formulas(assumptions).
 a <-> (b & c <-> c).
 b <-> (a & c -> b & c & -a).
 c <-> (-b <-> a | b).
end_of_list.

formulas(goals).
 a & b & -c.
end_of_list.
```

El problema de los mentirosos

- Demostración con Prover9:

```
> prover9 <pb_mentirosos_2.in >pb_mentirosos_2.out
 1 a <-> (b & c <-> c).          [assumption]
 2 b <-> (a & c -> b & c & -a). [assumption]
 3 c <-> (-b <-> a | b).         [assumption].
 4 a & b & -c.                   [goal].
 5 -a | b | -c.  [clausify(1)].
 6 a | c.        [clausify(1)].
 9 b | a.        [clausify(2)].
10 b | c.        [clausify(2)].
11 -c | -b.      [clausify(3)].
12 -a | -b | c.  [deny(4)].
13 b | -a.       [10,5]
14 -b | a.       [11,6].
15 a.            [14,9].
16 b.            [13,15].
17 c.            [12,15,16].
18 $F.           [11,17,16].
============================== end of proof ==========================
THEOREM PROVED
```

El problema de los animales

- Enunciado: Disponemos de una base de conocimiento compuesta de reglas sobre clasificación de animales y hechos sobre características de un animal.
  - Regla 1: Si un animal es ungulado y tiene rayas negras, entonces es una cebra.
  - Regla 2: Si un animal rumia y es mamífero, entonces es ungulado.
  - Regla 3: Si un animal es mamífero y tiene pezuñas, entonces es ungulado.
  - Hecho 1: El animal tiene es mamífero.
  - Hecho 2: El animal tiene pezuñas.
  - Hecho 3: El animal tiene rayas negras.

  Demostrar a partir de la base de conocimientos que el animal es una cebra.

El problema de los animales

- Representación en Prover9 (`pb_animales.in`)

```
formulas(assumptions).
 es_ungulado & tiene_rayas_negras -> es_cebra.
 rumia & es_mamifero -> es_ungulado.
 es_mamifero & tiene_pezugnas -> es_ungulado.
 es_mamifero.
 tiene_pezugnas.
 tiene_rayas_negras.
end_of_list.

formulas(goals).
 es_cebra.
end_of_list.
```

El problema de los animales

- Demostración con Prover9:

```
> prover9 <pb_animales.in
============================== PROOF =================================
 1 es_ungulado & tiene_rayas_negras -> es_cebra.  [assumption].
 3 es_mamifero & tiene_pezugnas -> es_ungulado.   [assumption].
 4 es_cebra # label(non_clause).                  [goal].
 5 -es_ungulado | -tiene_rayas_negras | es_cebra. [clausify(1)].
 7 -es_mamifero | -tiene_pezugnas | es_ungulado.  [clausify(3)].
 8 es_mamifero.                                   [assumption].
 9 tiene_pezugnas.                                [assumption].
10 tiene_rayas_negras.                            [assumption].
11 -es_cebra.                                     [deny(4)].
12 es_ungulado.                                   [7,8,9].
14 -es_ungulado.                                  [5,10,11].
15 $F.                                            [14,12].
============================== end of proof ==========================
THEOREM PROVED
```

El problema de los animales

- Confirmación con Mace4:

```
> mace4 -N2 <pb_animales.in
formulas(mace4_clauses).
-es_ungulado | -tiene_rayas_negras | es_cebra.
-rumia | -es_mamifero | es_ungulado.
-es_mamifero | -tiene_pezugnas | es_ungulado.
es_mamifero.
tiene_pezugnas.
tiene_rayas_negras.
-es_cebra.
end_of_list.

Exiting with failure.

------ process 5818 exit (exhausted) ------
```

El problema del coloreado del pentágono (con 2 colores)

- Enunciado: Decidir si es posible colorear los vértices de un pentágono de rojo o azul de forma que los vértices adyacentes tengan colores distintos.
- Simbolización:
  - 1, 2, 3, 4, 5 representan los vértices consecutivos del pentágono
  - `ri` $(1 \leq i \leq 5)$ representa que el vértice `i` es rojo
  - `ai` $(1 \leq i \leq 5)$ representa que el vértice `i` es azul

El problema del coloreado del pentágono (con 2 colores)

- Representación en Mace4 (`pb_pentagono_2.in`)

```
formulas(assumptions).
 % 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.
```

El problema del coloreado del pentágono (con 2 colores)

- Cálculo de modelos con Mace4:

```
> mace4 -N2 <pb_pentagono_2.in
Exiting with failure.
------ process 6292 exit (exhausted) ------
```

- Conclusión: Mace4 no ha encontrado ningún modelo. Luego, es imposible colorear los vértices de un pentágono de rojo o azul de forma que los vértices adyacentes tengan colores distintos.

El problema del coloreado del pentágono (con 3 colores)

- Enunciado: Decidir si es posible colorear los vértices de un pentágono de rojo, azul o negro de forma que los vértices adyacentes tengan colores distintos.
- Representación en Mace4 (`pb_pentagono_3.in`)

```
formulas(assumptions).
 % 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).
```

El problema del coloreado del pentágono (con 3 colores)

- Representación en Mace4 (cont.)

```
% 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.
```

El problema del coloreado del pentágono (con 3 colores)

- Cálculo de modelo con Mace4:

```
> mace4 -N2 -p1 <pb_pentagono_3.in
 a1 : 0
 a2 : 0
 a3 : 0
 a4 : 0
 a5 : 1
 n1 : 0
 n2 : 1
 n3 : 0
 n4 : 1
 n5 : 0
 r1 : 1
 r2 : 0
 r3 : 1
 r4 : 0
 r5 : 0
```

- Conclusión: colorear el vértice 1 de rojo, el 2 de negro, el 3 de rojo, el 4 de negro y el 5 de azul.

El problema del palomar

- Enunciado: Cuatro palomas comparten tres huecos. Decidir si es posible que no haya dos palomas en el mismo hueco.
- Simbolización: `pihj` ($i \in \{1,2,3,4\}$ y $j \in \{1,2,3\}$) representa que la paloma `i` está en el hueco `j`.

El problema del palomar

- Representación en Mace4 (`pb_palomar.in`):

```
formulas(assumptions).
 % La paloma 1 está en algún hueco:
 p1h1 | p1h2 | p1h3.

 % La paloma 2 está en algún hueco:
 p2h1 | p2h2 | p2h3.

 % La paloma 3 está en algún hueco:
 p3h1 | p3h2 | p3h3.

 % La paloma 4 está en algún hueco:
 p4h1 | p4h2 | p4h3.
```

El problema del palomar

- Representación en Mace4 (cont.)

```
 % No hay dos palomas en el hueco 1:
 -p1h1 | -p2h1. -p1h1 | -p3h1. -p1h1 | -p4h1.
 -p2h1 | -p3h1. -p2h1 | -p4h1. -p3h1 | -p4h1.

 % No hay dos palomas en el hueco 2:
 -p1h2 | -p2h2. -p1h2 | -p3h2. -p1h2 | -p4h2.
 -p2h2 | -p3h2. -p2h2 | -p4h2. -p3h2 | -p4h2.

 % No hay dos palomas en el hueco 3:
 -p1h3 | -p2h3. -p1h3 | -p3h3. -p1h3 | -p4h3.
 -p2h3 | -p3h3. -p2h3 | -p4h3. -p3h3 | -p4h3.
end_of_list.
```

El problema del palomar

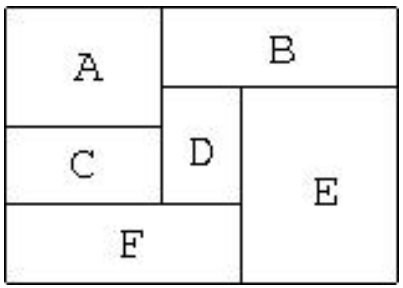
- Cálculo de modelo con Mace4:

```
> mace4 -N2 <pb_palomar.in
Exiting with failure.
------ process 6598 exit (exhausted) ------
```

- Conclusión: Mace4 no ha encontrado ningún modelo. Luego, es imposible que no haya dos palomas en el mismo hueco.

El problema de los rectángulos

- Enunciado: Un rectángulo se divide en seis rectángulos menores como se indica en la figura. Demostrar que si cada una de los rectángulos menores tiene un lado cuya medida es un número entero, entonces la medida de alguno de los lados del rectángulo mayor es un número entero.



- Simbolización:
  - `base`: la base del rectángulo mayor es un número entero
  - `altura`: la altura del rectángulo mayor es un número entero
  - `base_x`: la base del rectángulo X es un número entero
  - `altura_x`: la altura del rectángulo X es un número entero

El problema de los rectángulos

- Representación en Prover9 (`pb_rectangulos.in`)

```
formulas(assumptions).
 base_a | altura_a. base_b | altura_b. base_c | altura_c.
 base_d | altura_d. base_e | altura_e. base_f | altura_f.
 base_a <-> base_c.
 base_a & base_d -> base_f.
 base_d & base_e -> base_b.
 base_a & base_b -> base.
 altura_d & altura_f -> altura_e.
 altura_a & altura_c & altura_f -> altura.
 altura_b & altura_d & altura_f -> altura.
 altura_b & altura_e -> altura.
end_of_list.
```

El problema de los rectángulos

- Representación en Prover9 (cont.)

```
formulas(goals).
 base | altura.
end_of_list.
```

- Demostración con Prover9:

```
> prover9 <pb_rectangulos.in
THEOREM PROVED
```

El problema de las 4 reinas

- Enunciado: Calcular las formas de colocar 4 reinas en un tablero de 4x4 de forma que no haya más de una reina en cada fila, columna o diagonal.
- Representación: `cij` ($1 \leq i, j \leq 4$) indica que hay una reina en la fila `i` columna `j`.

El problema de las 4 reinas

- Representación en Mace4 (`pb_4_reinas.in`)

```
formulas(assumptions).
% En cada fila hay una reina:
c11 | c12 | c13 | c14.
c21 | c22 | c23 | c24.
c31 | c32 | c33 | c34.
c41 | c42 | c43 | c44.
```

El problema de las 4 reinas

- Representación en Mace4 (cont.)

```
% Si en una casilla hay reina, entonces no hay más
% reinas en su fila, su columna y su diagonal:
c11 -> (-c12 & -c13 & -c14) & (-c21 & -c31 & -c41) &
       (-c22 & -c33 & -c44).
c12 -> (-c11 & -c13 & -c14) & (-c22 & -c32 & -c42) &
       (-c21 & -c23 & -c34).
c13 -> (-c11 & -c12 & -c14) & (-c23 & -c33 & -c43) &
       (-c31 & -c22 & -c24).
c14 -> (-c11 & -c12 & -c13) & (-c24 & -c34 & -c44) &
       (-c23 & -c32 & -c41).
```

El problema de las 4 reinas

- Representación en Mace4 (cont.)

```
c21 -> (-c22 & -c23 & -c24) & (-c11 & -c31 & -c41) &
       (-c32 & -c43 & -c12).
c22 -> (-c21 & -c23 & -c24) & (-c12 & -c32 & -c42) &
       (-c11 & -c33 & -c44 & -c13 & -c31).
c23 -> (-c21 & -c22 & -c24) & (-c13 & -c33 & -c43) &
       (-c12 & -c34 & -c14 & -c32 & -c41).
c24 -> (-c21 & -c22 & -c23) & (-c14 & -c34 & -c44) &
       (-c13 & -c33 & -c42).
```

El problema de las 4 reinas

- Representación en Mace4 (cont.)

```
c31 -> (-c32 & -c33 & -c34) & (-c11 & -c21 & -c41) &
       (-c42 & -c13 & -c22).
c32 -> (-c31 & -c33 & -c34) & (-c12 & -c22 & -c42) &
       (-c21 & -c43 & -c14 & -c23 & -c41).
c33 -> (-c31 & -c32 & -c34) & (-c13 & -c23 & -c43) &
       (-c11 & -c22 & -c44 & -c24 & -c42).
c34 -> (-c31 & -c32 & -c33) & (-c14 & -c24 & -c44) &
       (-c12 & -c23 & -c43).
```

El problema de las 4 reinas

- Representación en Mace4 (cont.)

```
c41 -> (-c42 & -c43 & -c44) & (-c11 & -c21 & -c31) &
       (-c14 & -c23 & -c32).
c42 -> (-c41 & -c43 & -c44) & (-c12 & -c22 & -c32) &
       (-c31 & -c24 & -c33).
c43 -> (-c41 & -c42 & -c44) & (-c13 & -c23 & -c33) &
       (-c21 & -c32 & -c34).
c44 -> (-c41 & -c42 & -c43) & (-c14 & -c24 & -c34) &
       (-c11 & -c22 & -c33).
end_of_list.
```

El problema de las 4 reinas

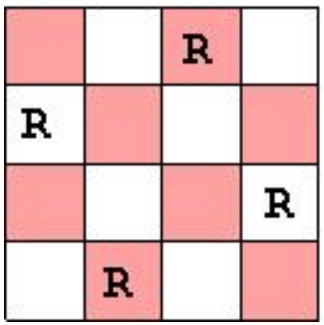
- Búsqueda de modelos con Mace4:

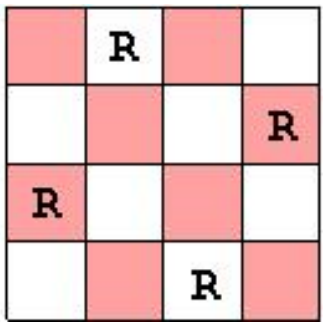
```
> mace4 -N2 -m9 -p1 <pb_4_reinas.in
 c11 : 0  c12 : 0  c13 : 1  c14 : 0
 c21 : 1  c22 : 0  c23 : 0  c24 : 0
 c31 : 0  c32 : 0  c33 : 0  c34 : 1
 c41 : 0  c42 : 1  c43 : 0  c44 : 0

 c11 : 0  c12 : 1  c13 : 0  c14 : 0
 c21 : 0  c22 : 0  c23 : 0  c24 : 1
 c31 : 1  c32 : 0  c33 : 0  c34 : 0
 c41 : 0  c42 : 0  c43 : 1  c44 : 0
```

El problema de las 4 reinas

- Conclusión: Gráficamente los modelos son





El problema de Ramsey

- Enunciado: Probar el caso más simple del teorema de Ramsey: entre seis personas siempre hay (al menos) tres tales que cada una conoce a las otras dos o cada una no conoce a ninguna de las otras dos.
- Simbolización:
  - 1,2,3,4,5,6 representan a las personas
  - `pij` ($1 \leq i < j \leq 6$) indica que las personas `i` y `j` se conocen.

El problema de Ramsey

- Representación en Prover9 (`pb_ramsey.in`)

```
formulas(goals).
 % Hay 3 personas que se conocen entre ellas:
 (p12 & p13 & p23) | (p12 & p14 & p24) |
 (p12 & p15 & p25) | (p12 & p16 & p26) |
 (p13 & p14 & p34) | (p13 & p15 & p35) |
 (p13 & p16 & p36) | (p14 & p15 & p45) |
 (p14 & p16 & p46) | (p15 & p16 & p56) |
 (p23 & p24 & p34) | (p23 & p25 & p35) |
 (p23 & p26 & p36) | (p24 & p25 & p45) |
 (p24 & p26 & p46) | (p25 & p26 & p56) |
 (p34 & p35 & p45) | (p34 & p36 & p46) |
 (p35 & p36 & p56) | (p45 & p46 & p56) |
```

El problema de Ramsey

- Representación en Prover9 (cont.)

```
% Hay 3 personas que que se desconocen:
(-p12 & -p13 & -p23) | (-p12 & -p14 & -p24) |
(-p12 & -p15 & -p25) | (-p12 & -p16 & -p26) |
(-p13 & -p14 & -p34) | (-p13 & -p15 & -p35) |
(-p13 & -p16 & -p36) | (-p14 & -p15 & -p45) |
(-p14 & -p16 & -p46) | (-p15 & -p16 & -p56) |
(-p23 & -p24 & -p34) | (-p23 & -p25 & -p35) |
(-p23 & -p26 & -p36) | (-p24 & -p25 & -p45) |
(-p24 & -p26 & -p46) | (-p25 & -p26 & -p56) |
(-p34 & -p35 & -p45) | (-p34 & -p36 & -p46) |
(-p35 & -p36 & -p56) | (-p45 & -p46 & -p56).
end_of_list.
```

El problema de Ramsey

- Demostración con Prover9:

```
> prover9 <pb_ramsey.in
THEOREM PROVED
```

Bibliografía

- Alonso, J.A. Razonamiento proposicional con Otter y Mace
- Alonso, J.A. y Borrego, J. *Deducción automática (Vol. 1: Construcción lógica de sistemas lógicos)* (Ed. Kronos, 2002)
  - Cap. 3: Elementos de lógica proposicional
- Ben–Ari, M. *Mathematical Logic for Computer Science (third ed.)* (Springer, 2011)
  - Cap. 2: Propositional Calculus: Formulas, Models, Tableaux
- Fitting, M. *First-Order Logic and Automated Theorem Proving (2nd ed.)* (Springer, 1995)
- Nerode,A. y Shore, R.A. *Logic for Applications* (Springer, 1997)