Tema 9: Formalización en Prolog de la lógica proposicional

José A. Alonso Jiménez

Jose-Antonio.Alonso@cs.us.es
http://www.cs.us.es/∼jalonso

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Sintaxis de la lógica proposicional

- **Alfabeto proposicional:**
  - **símbolos proposicionales.**
  - **conectivas lógicas: $\neg$ (negación), $\wedge$ (conjunción), $\vee$ (disyunción), $\rightarrow$ (condicional), $\leftrightarrow$ (equivalencia).**
  - **símbolos auxiliares: "(" y ")".**
- **Fórmulas proposicionales:**
  - **símbolos proposicionales**
  - **$\neg F$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$, $(F \leftrightarrow G)$**
- **Eliminación de paréntesis:**
  - **Eliminación de paréntesis externos.**
  - **Precedencia: $\neg, \wedge, \vee \rightarrow, \leftrightarrow$**
  - **Asociatividad: $\wedge$ y $\vee$ asocian por la derecha**

Sintaxis de la lógica proposicional

- **Sintaxis en Prolog**

Usual	$\neg$	$\wedge$	$\vee$	$\longrightarrow$	$\leftrightarrow$
Prolog	-	&	v	=>	<=>

- **Declaración de operadores:**

```
:- op(610, fy,  -).      % negación
:- op(620, xfy, &).      % conjunción
:- op(630, xfy, v).      % disyunción
:- op(640, xfy, =>).     % condicional
:- op(650, xfy, <=>).    % equivalencia
```

Valores de verdad

- **Valores de verdad:**
  - **1: verdadero y 0: falso**
- **Def. de `valor_de_verdad`:**
  - **`valor_de_verdad(?V)` si `V` es un valor de verdad.**

```
valor_de_verdad(0).
valor_de_verdad(1).
```

Funciones de verdad

- **Funciones de verdad:**

i	$\neg i$
1	0
0	1

i	j	i $\wedge$ j	i $\vee$ j	i $\longrightarrow$ 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

  - `función_de_verdad(+Op, +V1, +V2, -V)` **si** `Op(V1,V2)=V`.
    `función_de_verdad(+Op, +V1, -V)` **si** `Op(V1)) = V`

```
función_de_verdad(v,   0, 0, 0) :- !.
función_de_verdad(v,   _, _, 1).
función_de_verdad(&,   1, 1, 1) :- !.
función_de_verdad(&,   _, _, 0).
función_de_verdad(=>,  1, 0, 0) :- !.
función_de_verdad(=>,  _, _, 1).
función_de_verdad(<=>, X, X, 1) :- !.
función_de_verdad(<=>, _, _, 0).

función_de_verdad(-,  1, 0).
función_de_verdad(-,  0, 1).
```

Valor de una fórmula

- **Representación de las interpretaciones**
  - **Listas de pares de variables y valores de verdad**
  - **Ejemplo:** `[(p,1),(r,0),(u,1)]`
- **Def. del valor de una fórmula en una interpretación**
  - `valor(+F, +I, -V)` **se verifica si el valor de la fórmula** `F` **en la interpretación** `I` **es** `V`
  - **Ejemplos:**

```
?- valor((p v q) & (-q v r),[(p,1),(q,0),(r,1)],V).
V = 1
?- valor((p v q) & (-q v r),[(p,0),(q,0),(r,1)],V).
V = 0
```

Valor de una fórmula

- **Def. de `valor`:**

```
valor(F, I, V) :-
   memberchk((F,V), I).
valor(-A, I, V) :-
   valor(A, I, VA),
   función_de_verdad(-, VA, V).
valor(F, I, V) :-
   functor(F,Op,2), arg(1,F,A), arg(2,F,B),   % F =..[Op,A,B],
   valor(A, I, VA),
   valor(B, I, VB),
   función_de_verdad(Op, VA, VB, V).
```

Interpretaciones de una fórmula

- ***I interpretación principal* de *F* syss *I* es una aplicación del conjunto de los símbolos proposicionales de *F* en el conjunto de los valores de verdad.**
- **Cálculo de las interpretaciones principales:**
  - `interpretaciones_fórmula(+F,-L)` **se verifica si** `L` **es el conjunto de las interpretaciones principales de la fórmula** `F`**.**
  - **Ejemplo**

```
?- interpretaciones_fórmula((p v q) & (-q v r),L).
L = [[ (p, 0),  (q, 0),  (r, 0)],
     [ (p, 0),  (q, 0),  (r, 1)],
     [ (p, 0),  (q, 1),  (r, 0)],
     [ (p, 0),  (q, 1),  (r, 1)],
     [ (p, 1),  (q, 0),  (r, 0)],
     [ (p, 1),  (q, 0),  (r, 1)],
     [ (p, 1),  (q, 1),  (r, 0)],
     [ (p, 1),  (q, 1),  (r, 1)]]
```

  - **Def. de** `interpretaciones_fórmula`**:**

```
interpretaciones_fórmula(F,U) :-
   findall(I,interpretación_fórmula(I,F),U).
```

Interpretaciones de una fórmula

- **Interpretación de una fórmula:**
  - `interpretación_fórmula(?I,+F)` **se verifica si** `I` **es una interpretación de la fórmula** `F`**.**
  - **Ejemplo:**

```
?- interpretación_fórmula(I,(p v q) & (-q v r)).
I = [ (p, 0),  (q, 0),  (r, 0)] ;
I = [ (p, 0),  (q, 0),  (r, 1)] ;
I = [ (p, 0),  (q, 1),  (r, 0)] ;
I = [ (p, 0),  (q, 1),  (r, 1)] ;
I = [ (p, 1),  (q, 0),  (r, 0)] ;
I = [ (p, 1),  (q, 0),  (r, 1)] ;
I = [ (p, 1),  (q, 1),  (r, 0)] ;
I = [ (p, 1),  (q, 1),  (r, 1)] ;
No
```

  - **Def. de** `interpretación_fórmula`**:**

```
interpretación_fórmula(I,F) :-
   símbolos_fórmula(F,U),
   interpretación_símbolos(U,I).
```

Interpretaciones de una fórmula

- **Símbolos de una fórmula**
  - `símbolos_fórmula(+F,?U)` **se verifica si** `U` **es el conjunto ordenado de los símbolos proposicionales de la fórmula** `F`**.**
  - **Ejemplo:**

```
?- símbolos_fórmula((p v q) & (-q v r), U).
U = [p, q, r]
```

  - **Def. de** `símbolo_fórmula`

```
símbolos_fórmula(F,U) :-
   símbolos_fórmula_aux(F,U1),
   sort(U1,U).
símbolos_fórmula_aux(F,[F]) :-
   atom(F).
símbolos_fórmula_aux(-F,U) :-
   símbolos_fórmula_aux(F,U).
símbolos_fórmula_aux(F,U) :-
   arg(1,F,A), arg(2,F,B),        % F =..[_Op,A,B],
   símbolos_fórmula_aux(A,UA),
   símbolos_fórmula_aux(B,UB),
   union(UA,UB,U).
```

Interpretaciones de una fórmula

- **Interpretación de una lista de símbolos:**
  - `interpretación_símbolos(+L,-I)` **se verifica si** `I` **es una interpretación de la lista de símbolos proposicionales** `L`**.**
  - **Ejemplo:**

```
?- interpretación_símbolos([p,q,r],I).
I = [ (p, 0),  (q, 0),  (r, 0)] ;
I = [ (p, 0),  (q, 0),  (r, 1)] ;
I = [ (p, 0),  (q, 1),  (r, 0)] ;
I = [ (p, 0),  (q, 1),  (r, 1)] ;
I = [ (p, 1),  (q, 0),  (r, 0)] ;
I = [ (p, 1),  (q, 0),  (r, 1)] ;
I = [ (p, 1),  (q, 1),  (r, 0)] ;
I = [ (p, 1),  (q, 1),  (r, 1)] ;
No
```

  - **Def. de** `interpretación_símbolos`

```
interpretación_símbolos([],[]).
interpretación_símbolos([A|L],[(A,V)|IL]) :-
   valor_de_verdad(V),
   interpretación_símbolos(L,IL).
```

Modelo de una fórmula

- **La interpretación *I* es un *modelo de la fórmula F* si el valor de *F* en *I* es verdadero.**
- **Comprobación de modelo de una fórmula:**
  - `es_modelo_fórmula(+I,+F)` **se verifica si la interpretación** `I` **es un modelo de la fórmula** `F`**.**
  - **Ejemplos:**

```
?- es_modelo_fórmula([(p,1),(q,0),(r,1)], (p v q) & (-q v r)).
Yes
?- es_modelo_fórmula([(p,0),(q,0),(r,1)], (p v q) & (-q v r)).
No
```

  - **Def. de** `es_modelo_fórmula`**:**

```
es_modelo_fórmula(I,F) :-
   valor(F,I,V),
   V = 1.
```

Cálculo de los modelos de una fórmula

- **Cálculo de los modelos principales de una fórmula:**
  - `modelo_fórmula(?I,+F)` **se verifica si** `I` **es un modelo principal de la fórmula** `F`**.**
  - **Ejemplo:**

```
?- modelo_fórmula(I,(p v q) & (-q v r)).
I = [ (p, 0),  (q, 1),  (r, 1)] ;
I = [ (p, 1),  (q, 0),  (r, 0)] ;
I = [ (p, 1),  (q, 0),  (r, 1)] ;
I = [ (p, 1),  (q, 1),  (r, 1)] ;
No
```

  - **Def. de** `modelo_fórmula`**:**

```
modelo_fórmula(I,F) :-
   interpretación_fórmula(I,F),
   es_modelo_fórmula(I,F).
```

Cálculo de los modelos de una fórmula

- `modelos_fórmula(+F,-L)` **se verifica si** `L` **es el conjunto de los modelos principales de la fórmula** `F`**.**
- **Ejemplo:**

```
?- modelos_fórmula((p v q) & (-q v r),L).
L = [[ (p, 0),  (q, 1),  (r, 1)],
     [ (p, 1),  (q, 0),  (r, 0)],
     [ (p, 1),  (q, 0),  (r, 1)],
     [ (p, 1),  (q, 1),  (r, 1)]]
```

- **Def. de** `modelos_fórmula`

```
modelos_fórmula(F,L) :-
   findall(I,modelo_fórmula(I,F),L).
```

Satisfacibilidad

- **Una fórmula es *satisfacible* si tiene modelo e *insatisfacible* si no lo tiene.**
- **Comprobación de satisfacibilidad:**
  - `es_satisfacible(+F)` **se verifica si la fórmula** `F` **es satisfacible.**
  - **Ejemplos:**

```
?- es_satisfacible((p v q) & (-q v r)).
Yes
?- es_satisfacible((p & q) & (p => r) & (q => -r)).
No
```

  - **Def. de** `es_satisfacible`**:**

```
es_satisfacible(F) :-
   interpretación_fórmula(I,F),
   es_modelo_fórmula(I,F).
```

Contramodelo de una fórmula

- **Un *contramodelo principal* de F es una interpretación principal de F que no es modelo de F.**
- **Cálculo de contramodelos:**
  - `contramodelo_fórmula(?I,+F)` **se verifica si** `I` **es un contramodelo principal de la fórmula** `F`**.**
  - **Ejemplos:**

```
?- contramodelo_fórmula(I, p <=> q).
I = [ (p, 0),  (q, 1)] ;
I = [ (p, 1),  (q, 0)] ;
No
?- contramodelo_fórmula(I, p => p).
No
```

  - **Def. de** `contramodelo_fórmula`**:**

```
contramodelo_fórmula(I,F) :-
   interpretación_fórmula(I,F),
   \+ es_modelo_fórmula(I,F).
```

Validez. Tautologías

- ***F* es *válida* (o *tautología*) si todas las interpretaciones son modelos de *F*.**
- **Comprobación de tautologías:**
  - `es_tautología(+F)` **se verifica si la fórmula** `F` **es una tautología.**
  - **Ejemplos:**

```
?- es_tautología((p => q) v (q => p)).
Yes
?- es_tautología(p => q).
No
```

  - **Def. de** `es_tautología`**:**

```
es_tautología(F) :-
   \+ contramodelo_fórmula(_I,F).
```

  - **Definición alternativa:**

```
es_tautología_alt(F) :-
   \+ es_satisfacible(-F).
```

Interpretaciones de conjuntos

- **Una *interpretación principal* de un conjunto de fórmulas es una aplicación del conjunto de sus símbolos proposicionales en el conjunto de los valores de verdad.**
- **Cálculo de las interpretaciones principales de un conjunto de fórmulas:**
  - **`interpretaciones_conjunto(+S,-L)` se verifica si `L` es el conjunto de las interpretaciones principales del conjunto `S`.**
  - **Ejemplo:**

```
?- interpretaciones_conjunto([p => q, q=> r],U).
U = [[ (p, 0),  (q, 0),  (r, 0)], [ (p, 0),  (q, 0),  (r, 1)],
     [ (p, 0),  (q, 1),  (r, 0)], [ (p, 0),  (q, 1),  (r, 1)],
     [ (p, 1),  (q, 0),  (r, 0)], [ (p, 1),  (q, 0),  (r, 1)],
     [ (p, 1),  (q, 1),  (r, 0)], [ (p, 1),  (q, 1),  (r, 1)]]
```

  - **Def. de `interpretaciones_conjunto`:**

```
interpretaciones_conjunto(S,U) :-
   findall(I,interpretación_conjunto(I,S),U).
```

Interpretaciones de conjuntos

- `interpretación_conjunto(?I,+S)` **se verifica si** `I` **es una interpretación del conjunto de fórmulas** `S`**.**
- **Ejemplo:**

```
?- interpretación_conjunto(I,[p => q, q=> r]).
I = [ (p, 0),  (q, 0),  (r, 0)] ;
I = [ (p, 0),  (q, 0),  (r, 1)] ;
I = [ (p, 0),  (q, 1),  (r, 0)] ;
I = [ (p, 0),  (q, 1),  (r, 1)] ;
I = [ (p, 1),  (q, 0),  (r, 0)] ;
I = [ (p, 1),  (q, 0),  (r, 1)] ;
I = [ (p, 1),  (q, 1),  (r, 0)] ;
I = [ (p, 1),  (q, 1),  (r, 1)] ;
No
```

- **Def. de** `interpretación_conjunto`**:**

```
interpretación_conjunto(I,S) :-
   símbolos_conjunto(S,U),
   interpretación_símbolos(U,I).
```

Interpretaciones de conjuntos

- **Cálculo de los símbolos de un conjunto de fórmulas:**
  - `símbolos_conjunto(+S,?U)` **se verifica si** `U` **es el conjunto ordenado de los símbolos proposicionales del conjunto de fórmulas** `S`**.**
  - **Ejemplo:**

```
?- símbolos_conjunto([p => q, q=> r],U).
U = [p, q, r]
```

  - **Def. de** `símbolos_conjunto`**:**

```
símbolos_conjunto(S,U) :-
   símbolos_conjunto_aux(S,U1),
   sort(U1,U).

símbolos_conjunto_aux([],[]).
símbolos_conjunto_aux([F|S],U) :-
   símbolos_fórmula(F,U1),
   símbolos_conjunto_aux(S,U2),
   union(U1,U2,U).
```

Modelo de un conjunto de fórmulas

- **La interpretación *I* es un *modelo del conjunto de fórmulas S* si *I* es modelo de todas las fórmulas de *S*.**
- **Comprobación de modelo de un conjunto de fórmulas**
  - `es_modelo_conjunto(+I,+S)` **se verifica si la interpretación** `I` **es un modelo del conjunto de fórmulas** `S`**.**
  - **Ejemplos:**

```
?- es_modelo_conjunto([(p,1),(q,0),(r,1)], [(p v q) & (-q v r),q => r]).
Yes
?- es_modelo_conjunto([(p,0),(q,1),(r,0)], [(p v q) & (-q v r),q => r]).
No
```

  - **Def. de** `es_modelo_conjunto`**:**

```
es_modelo_conjunto(_I,[]).
es_modelo_conjunto(I,[F|S]) :-
   es_modelo_fórmula(I,F),
   es_modelo_conjunto(I,S).
```

Cálculo de modelos de conjuntos

- **Cálculo de los modelos principales de conjuntos de fórmulas**
  - `modelo_conjunto(?I,+S)` **se verifica si** `I` **es un modelo principal del conjunto de fórmulas** `S`**.**
  - **Ejemplo:**

```
?- modelo_conjunto(I,[(p v q) & (-q v r),p => r]).
I = [ (p, 0),  (q, 1),  (r, 1)] ;
I = [ (p, 1),  (q, 0),  (r, 1)] ;
I = [ (p, 1),  (q, 1),  (r, 1)] ;
No
```

  - **Def. de** `modelo_conjunto`**:**

```
modelo_conjunto(I,S) :-
   interpretación_conjunto(I,S),
   es_modelo_conjunto(I,S).
```

Cálculo de modelos de conjuntos

- `modelos_conjunto(+S,-L)` **se verifica si** `L` **es el conjunto de los modelos principales del conjunto de fórmulas** `S`**.**
- **Ejemplo:**

```
?- modelos_conjunto([(p v q) & (-q v r),p => r],L).
L = [[ (p, 0),  (q, 1),  (r, 1)],
     [ (p, 1),  (q, 0),  (r, 1)],
     [ (p, 1),  (q, 1),  (r, 1)]]
```

- **Def. de** `modelos_conjunto`**:**

```
modelos_conjunto(S,L) :-
   findall(I,modelo_conjunto(I,S),L).
```

Consistencia de un conjunto

- **Un conjunto de fórmulas es *consistente* si tiene modelo e *inconsistente* en caso contrario.**
- **Comprobación de consistencia:**
  - `consistente(+S)` **se verifica si el conjunto de fórmulas** `S` **es consistente e** `inconsistente(+S)`**, si es inconsistente.**
  - **Ejemplos:**

```
?- consistente([(p v q) & (-q v r),p => r]).
Yes
?- consistente([(p v q) & (-q v r),p => r, -r]).
No
```

  - **Def. de** `consistente` **e** `inconsistente`**:**

```
consistente(S) :-
   modelo_conjunto(_I,S), !.

inconsistente(S) :-
   \+ modelo_conjunto(_I,S).
```

Consecuencia lógica

- **La fórmula F es *consecuencia lógica* del conjunto de fórmulas S ($S \models F$) si todos los modelos de S son modelos de F.**
- **($S \models F$) syss todas las interpretaciones principales de $S \cup \{F\}$ que son modelos de S también son modelos de F.**
- **Comprobación de consecuencia lógica:**
  - `es_consecuencia(+S,+F)` **se verifica si la fórmula** `F` **es consecuencia del conjunto de fórmulas** `S`**.**
  - **Ejemplos:**

```
?- es_consecuencia([p => q, q => r], p => r).
Yes
?- es_consecuencia([p], p & q).
No
```

  - **Def. de** `es_consecuencia`**:**

```
es_consecuencia(S,F) :-
   \+ contramodelo_consecuencia(S,F,_I).
```

Consecuencia lógica

- `contramodelo_consecuencia(+S,+F,?I)` **se verifica si** `I` **es una interpretación principal de** $\mathtt{S} \cup \{\mathtt{F}\}$ **que es modelo del conjunto de fórmulas** `S` **pero no es modelo de la fórmula** `F`**.**
- **Ejemplos:**

```
?- contramodelo_consecuencia([p], p & q, I).
I = [ (p, 1),  (q, 0)] ;
No
?- contramodelo_consecuencia([p => q, q=> r], p => r, I).
No
```

- **Def. de** `contramodelo_consecuencia`**:**

```
contramodelo_consecuencia(S,F,I) :-
   interpretación_conjunto(I,[F|S]),
   es_modelo_conjunto(I,S),
   \+ es_modelo_fórmula(I,F).
```

- **Definición alternativa de** `es_consecuencia`**:**

```
es_consecuencia_alt(S,F) :-
   inconsistente([-F|S]).
```

Ejemplo: veraces y mentirosos

- **El problema de los veraces y los 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.
  - **Representación:**
    `a`, `b` **y** `c` **representan que A, B y C son veraces**
    `-a`, `-b` **y** `-c` **representan que A, B y C son mentirosos**

Ejemplo: veraces y mentirosos

- **Idea: las tribus se determinan a partir de los modelos del conjunto de fórmulas correspondientes a las tres frases.**

```
?- modelos_conjunto([a <=> (b & c <=> c),
                     b <=> (a & c => b & c & -a),
                     c <=> (-b <=> a v b)],
                    L).
L = [[ (a, 1), (b, 1), (c, 0)]]
```

- **Solución: A y B son veraces y C es mentiroso.**

Ejemplo: El problema de los animales

- **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.

Ejemplo: El problema de los animales

- **Solución:**

```
?- es_consecuencia(
     [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).
Yes
```

Ejemplo: Problema de los trabajos

- **El problema de los trabajos**
  - **Enunciado: Juan, Sergio y Carlos trabajan de programador, ingeniero y administrador (aunque no necesariamente en este orden). Juan le debe 1000 euros al programador. La esposa del administrador le ha prohibido a su marido pedir dinero prestado (y éste le obedece). Sergio está soltero. Determinar el trabajo de cada uno.**
  - **Representación:**
    `cp` **(Carlos es programador)**
    `ci` **(Carlos es ingeniero)**
    `ca` **(Carlos es administrador)**
    `jp` **(Juan es programador)**
    `ji` **(Juan es ingeniero)**
    `ja` **(Juan es administrador)**
    `sp` **(Sergio es programador)**
    `si` **(Sergio es ingeniero)**
    `sa` **(Sergio es administrador).**

Ejemplo: Problema de los trabajos

- **Solución:**

```
modelos_conjunto(
 [jp v ji v ja,  % Juan es programador, ingeniero o administrador
  sp v si v sa,  % Sergio es programador, ingeniero o administrador
  cp v ci v ca,  % Carlos es programador, ingeniero o administrador
  % No hay más de un programador:
  (jp & -sp & -cp) v (-jp & sp & -cp) v (-jp & -sp & cp),
  % No hay más de un ingeniero:
  (ji & -si & -ci) v (-ji & si & -ci) v (-ji & -si & ci),
  % No hay más de un administrador:
  (ja & -sa & -ca) v (-ja & sa & -ca) v (-ja & -sa & ca),
  % Juan le debe 1000 pesetas al programador [Luego, Juan no es el programador]:
  -jp,
  % La esposa del administrador le ha prohibido a su marido pedir dinero
  % prestado (y éste le obedece) [Luego, Juan no es el administrador]:
  -ja,
  % Sergio está soltero [Luego, no es el adminitrador]:
  -sa],
 L).
L = [[(ca,1),(ci,0),(cp,0), (ja,0),(ji,1),(jp,0), (sa,0),(si,0),(sp,1)]].
```

- **Conclusión: Carlos es administrador, Juan es ingeniero y Sergio es programador.**

Ejemplo: El problema de los cuadrados

- **El problema de los cuadrados**
  - **Enunciado: Existe nueve símbolos proposicionales que se pueden ordenar en un cuadrado. Se sabe que existe alguna letra tal que para todos los números las fórmulas son verdaderas (es decir, existe una fila de fórmulas verdaderas). El objetivo de este ejercicio demostrar que para cada número existe una letra cuya fórmula es verdadera (es decir, en cada columna existe una fórmula verdadera).**

```
a1  a2  a3
b1  b2  b3
c1  c2  c3
```

  - **Solución:**

```
?- es_tautología((a1 & a2 & a3) v
                 (b1 & b2 & b3) v
                 (c1 & c2 & c3)
                 =>
                 (a1 v b1 v c1) v
                 (a2 v b2 v c2) v
                 (a3 v b3 v c3)).
Yes
```

Ejemplo: Problema del coloreado

- **El problema del coloreado del pentágono (con dos colores)**
  - **Enunciado: Demostrar que es imposible colorear los vértices de un pentágono de rojo o azul de forma que los vértices adyacentes tengan colores distintos.**
  - **Representación:**
    - **1, 2, 3, 4, 5 representan los vértices consecutivos del pentágono**
    - **`ri` ($1 \leq$ `i` ≤ 5) representa que el vértice `i` es rojo**
    - **`ai` ($1 \leq$ `i` ≤ 5) representa que el vértice `i` es azul**

Ejemplo: Problema del coloreado

- **Solución:**

```
?- inconsistente(
    [% El vértice i (1 <= i <= 5) es azul o rojo:
     a1 v r1, a2 v r2, a3 v r3, a4 v r4, a5 v r5,

     % Un vértice no puede tener dos colores:
     a1 => -r1, r1 => -a1, a2 => -r2, r2 => -a2, a3 => -r3,
     r3 => -a3, a4 => -r4, r4 => -a4, a5 => -r5, r5 => -a5,

     % 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)]).
Yes
```

Ejemplo: Problema del coloreado

- **El problema del coloreado del pentágono (con tres colores)**
  - **Enunciado: Demostrar que es posible colorear los vértices de un pentágono de rojo, azul o negro de forma que los vértices adyacentes tengan colores distintos.**
  - **Solución:**

```
?- modelo_conjunto(I,
    [% El vértice i (1 <= i <= 5) azul, rojo o negro:
     a1 v r1 v n1, a2 v r2 v n2, a3 v r3 v n3, a4 v r4 v n4, a5 v r5 v n5,

     % Un vértice no puede tener dos colores:
     a1 => -r1 & -n1, r1 => -a1 & -n1, n1 => -a1 & -r1,
     a2 => -r2 & -n2, r2 => -a2 & -n2, n2 => -a2 & -r2,
     a3 => -r3 & -n3, r3 => -a3 & -n3, n3 => -a3 & -r3,
     a4 => -r4 & -n4, r4 => -a4 & -n4, n4 => -a4 & -r4,
     a5 => -r5 & -n5, r5 => -a5 & -n5, n5 => -a5 & -r5,
```

Ejemplo: Problema del coloreado

```
      % 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)]).
```

```
I = [ (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.**

Ejemplo: Problema del palomar

- **El problema del palomar**
  - **Enunciado: Cuatro palomas comparten tres huecos. Demostrar que dos palomas tienen que estar en la misma hueco.**
  - **Representación:** `pihj` **(**`i`**= 1, 2, 3, 4 y** `j`**= 1, 2, 3) representa que la paloma** `i` **está en la hueco** `j`**.**

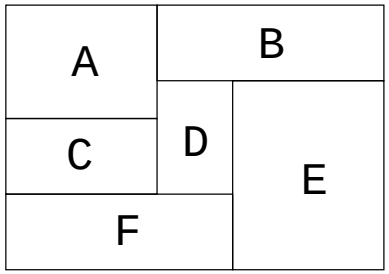
Ejemplo: Problema del palomar

- **Solución:**

```
?- inconsistente([
    % La paloma i está en alguna hueco:
    p1h1 v p1h2 v p1h3, p2h1 v p2h2 v p2h3,
    p3h1 v p3h2 v p3h3, p4h1 v p4h2 v p4h3,

    % No hay dos palomas en la hueco 1:
    -p1h1 v -p2h1, -p1h1 v -p3h1, -p1h1 v -p4h1,
    -p2h1 v -p3h1, -p2h1 v -p4h1, -p3h1 v -p4h1,

    % No hay dos palomas en la hueco 2:
    -p1h2 v -p2h2, -p1h2 v -p3h2, -p1h2 v -p4h2,
    -p2h2 v -p3h2, -p2h2 v -p4h2, -p3h2 v -p4h2,

    % No hay dos palomas en la hueco 3:
    -p1h3 v -p2h3, -p1h3 v -p3h3, -p1h3 v -p4h3,
    -p2h3 v -p3h3, -p2h3 v -p4h3, -p3h3 v -p4h3]).
Yes
```

Ejemplo: Problema de los rectángulos

- **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.**



  - **Representación**
    - `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**

- **Solución:**

```
?- es_consecuencia(
     [base_a v altura_a, base_b v altura_b,
      base_c v altura_c, base_d v altura_d,
      base_e v altura_e, base_f v altura_f,
      base_a <=> base_c,
      base_a & base_d => base_f,
      base_f & base_a => base_d,
      base_f & base_d => base_a,
      base_d & base_e => base_b,
      base_b & base_d => base_e,
      base_b & base_e => base_d,
      base_a & base_b => base,
      base   & base_a => base_b,
      base   & base_b => base_a,
      base_a & base_d & base_e => base,
      base   & base_a & base_d => base_e,
      base   & base_a & base_e => base_d,
      base   & base_d & base_e => base_a,
      base_f & base_e => base,
      base   & base_f => base_e,
      base   & base_e => base_f,
```

Ejemplo: Problema de los rectángulos

```
        altura_d & altura_f => altura_e,
        altura_e & altura_d => altura_f,
        altura_e & altura_f => altura_d,
        altura_a & altura_c & altura_f => altura,
        altura   & altura_a & altura_c => altura_f,
        altura   & altura_a & altura_f => altura_c,
        altura   & altura_c & altura_f => altura_a,
        altura_b & altura_d & altura_f => altura,
        altura   & altura_b & altura_d => altura_f,
        altura   & altura_b & altura_f => altura_d,
        altura   & altura_d & altura_f => altura_b,
        altura_b & altura_e => altura,
        altura   & altura_b => altura_e,
        altura   & altura_e => altura_b],
      base v altura).
Yes
```

Ejemplo: Problema de las 4 reinas

- **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` ≤ 4) indica que hay una reina en la fila `i` columna `j`.**

Ejemplo: Problema de las 4 reinas

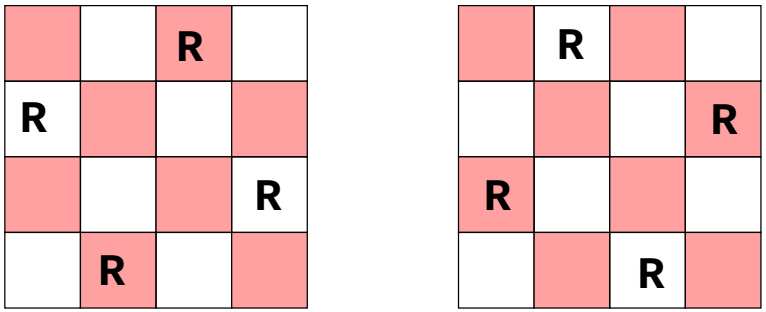
- **Solución:**

```
?- modelos_conjunto([
   % En cada fila hay una reina:
   c11 v c12 v c13 v c14, c21 v c22 v c23 v c24,
   c31 v c32 v c33 v c34, c41 v c42 v c43 v c44,
   % Si en una casilla hay reina, entonces no hay más reinas en sus líneas:
   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),
   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),
   c31 => (-c32 & -c33 & -c34) & (-c11 & -c21 & -c41) & -c42 & (-c13 & -c22),
   c32 => (-c31 & -c33 & -c34) & (-c12 & -c22 & -c42) & (-c21 & -c43)
          & (-c14  & -c23 & -c41),
```

Ejemplo: Problema de las 4 reinas

```
    c33 => (-c31 & -c32 & -c34) & (-c13 & -c23 & -c43) & (-c11 & -c22 & -c44)
           & (-c24 & -c42),
    c34 => (-c31 & -c32 & -c33) & (-c14 & -c24 & -c44) & (-c12 & -c23 & -c43),
    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)],
  L),

L = [[(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)]]
```

- **Conclusión: Gráficamente los modelos son**



Ejemplo: Problema de Ramsey

- **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.**
  - **Representación:**
    - **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.**
  - **Solución:**

```
?- es_tautología(
% Hay 3 personas que se conocen entre ellas:
(p12 & p13 & p23) v (p12 & p14 & p24) v (p12 & p15 & p25) v (p12 & p16 & p26) v
(p13 & p14 & p34) v (p13 & p15 & p35) v (p13 & p16 & p36) v (p14 & p15 & p45) v
(p14 & p16 & p46) v (p15 & p16 & p56) v (p23 & p24 & p34) v (p23 & p25 & p35) v
(p23 & p26 & p36) v (p24 & p25 & p45) v (p24 & p26 & p46) v (p25 & p26 & p56) v
(p34 & p35 & p45) v (p34 & p36 & p46) v (p35 & p36 & p56) v (p45 & p46 & p56) v
```

Ejemplo: Problema de Ramsey

```
% Hay 3 personas tales que cada una desconoce a las otras dos:
(-p12 & -p13 & -p23) v (-p12 & -p14 & -p24) v
(-p12 & -p15 & -p25) v (-p12 & -p16 & -p26) v
(-p13 & -p14 & -p34) v (-p13 & -p15 & -p35) v
(-p13 & -p16 & -p36) v (-p14 & -p15 & -p45) v
(-p14 & -p16 & -p46) v (-p15 & -p16 & -p56) v
(-p23 & -p24 & -p34) v (-p23 & -p25 & -p35) v
(-p23 & -p26 & -p36) v (-p24 & -p25 & -p45) v
(-p24 & -p26 & -p46) v (-p25 & -p26 & -p56) v
(-p34 & -p35 & -p45) v (-p34 & -p36 & -p46) v
(-p35 & -p36 & -p56) v (-p45 & -p46 & -p56)).
Yes
```

Ejemplos: Comparación

- Comparación de la resolución de los problemas:

Problema	Símbolos	Inferencias	Tiempo
mentirosos	3	646	0.00
animales	6	4,160	0.00
trabajos	9	71,044	0.07
cuadrados	9	56,074	0.06
pentágono_3	15	117,716	0.13
palomar	12	484,223	0.50
rectángulos	14	1,026,502	1.08
4 reinas	16	15,901,695	19.90
Ramsey	15	29,525,686	44.27

Bibliografía

- **Alonso, J.A. y Borrego, J. *Deducción automática (Vol. 1: Construcción lógica de sistemas lógicos)* (Ed. Kronos, 2002)**
  `www.cs.us.es/~jalonso/libros/da1-02.pdf`
  - **Cap. 3: Elementos de lógica proposicional**
- **Ben–Ari, M. *Mathematical Logic for Computer Science (2nd ed.)* (Springer, 2001)**
  - **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)**