Programación declarativa (2006–07)

Tema 11: Formalización en Prolog de la lógica proposicional

José A. Alonso Jiménez

Andrés Cordón Franco

Grupo de Lógica Computacional

Dpto. 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
  - $\neg \boldsymbol{F}$, $(\boldsymbol{F} \wedge \boldsymbol{G})$, $(\boldsymbol{F} \vee \boldsymbol{G})$, $(\boldsymbol{F} \rightarrow \boldsymbol{G})$, $(\boldsymbol{F} \leftrightarrow \boldsymbol{G})$.
- 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	$\neg$	$\wedge$	$\vee$	$\rightarrow$	$\leftrightarrow$
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
```

Demostración por tableros semánticos

- Demostración de fórmula válida:
  - Demostración:

    $\neg p \vee \neg q \longrightarrow \neg(p \wedge q)$ es válida

    syss $\{\neg(\neg p \vee \neg q \longrightarrow \neg(p \wedge q))\}$ es inconsistente

    syss $\{\neg p \vee \neg q, \neg\neg(p \wedge q)\}$ es inconsistente

    syss $\{\neg p \vee \neg q, p \wedge q\}$ es inconsistente

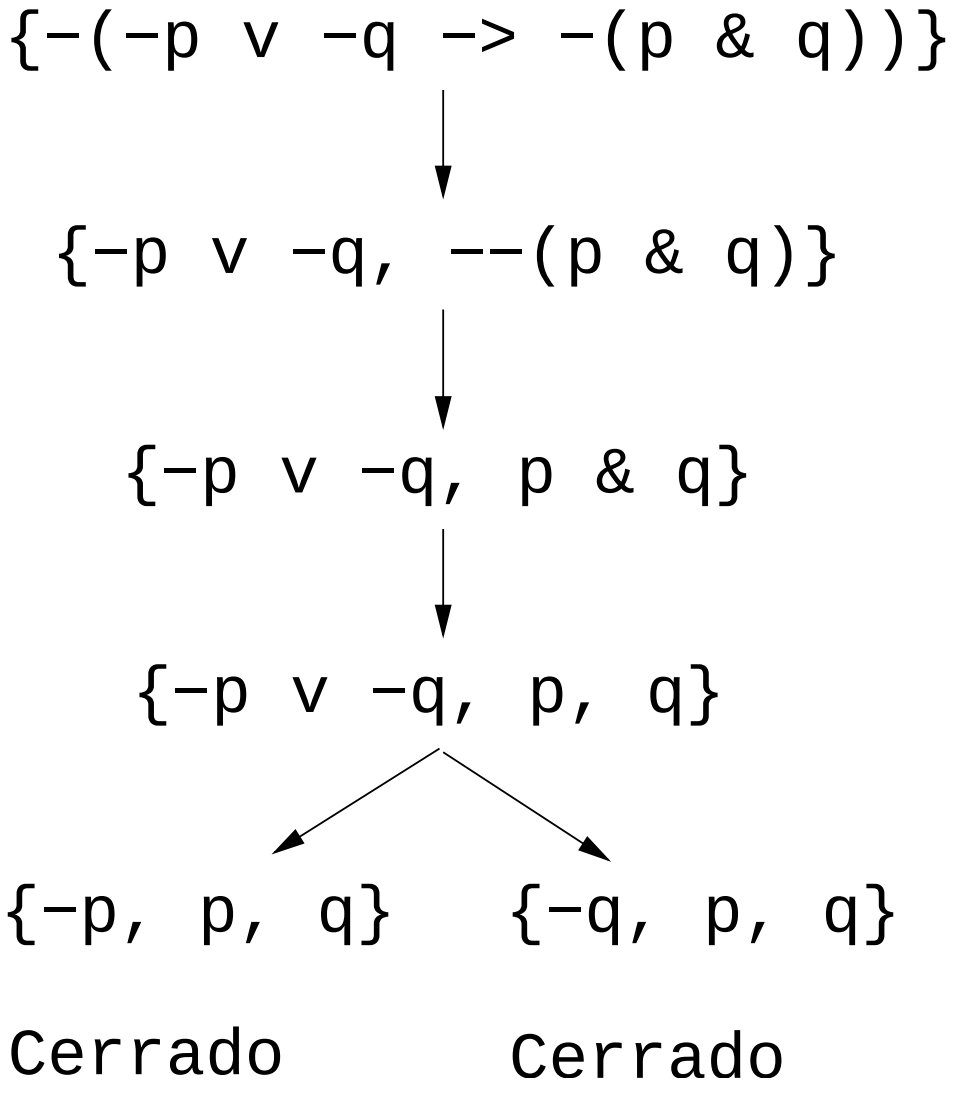
    syss $\{p, q, \neg p \vee \neg q\}$ es inconsistente

    syss $\{p, q, \neg p\}$ es inconsistente y

    $\{p, q, \neg q\}$ es inconsistente

Demostración por tableros semánticos

- Tablero semántico:



Demostración por tableros semánticos

- Demostración de fórmula no válida:
  - Demostración:

    $\neg p \vee \neg q \longrightarrow \neg(p \wedge r)$ es válida

    syss $\{\neg(\neg p \vee \neg q \longrightarrow \neg(p \wedge r))\}$ es inconsistente

    syss $\{\neg p \vee \neg q, \neg\neg(p \wedge r)\}$ es inconsistente

    syss $\{\neg p \vee \neg q, p \wedge r\}$ es inconsistente

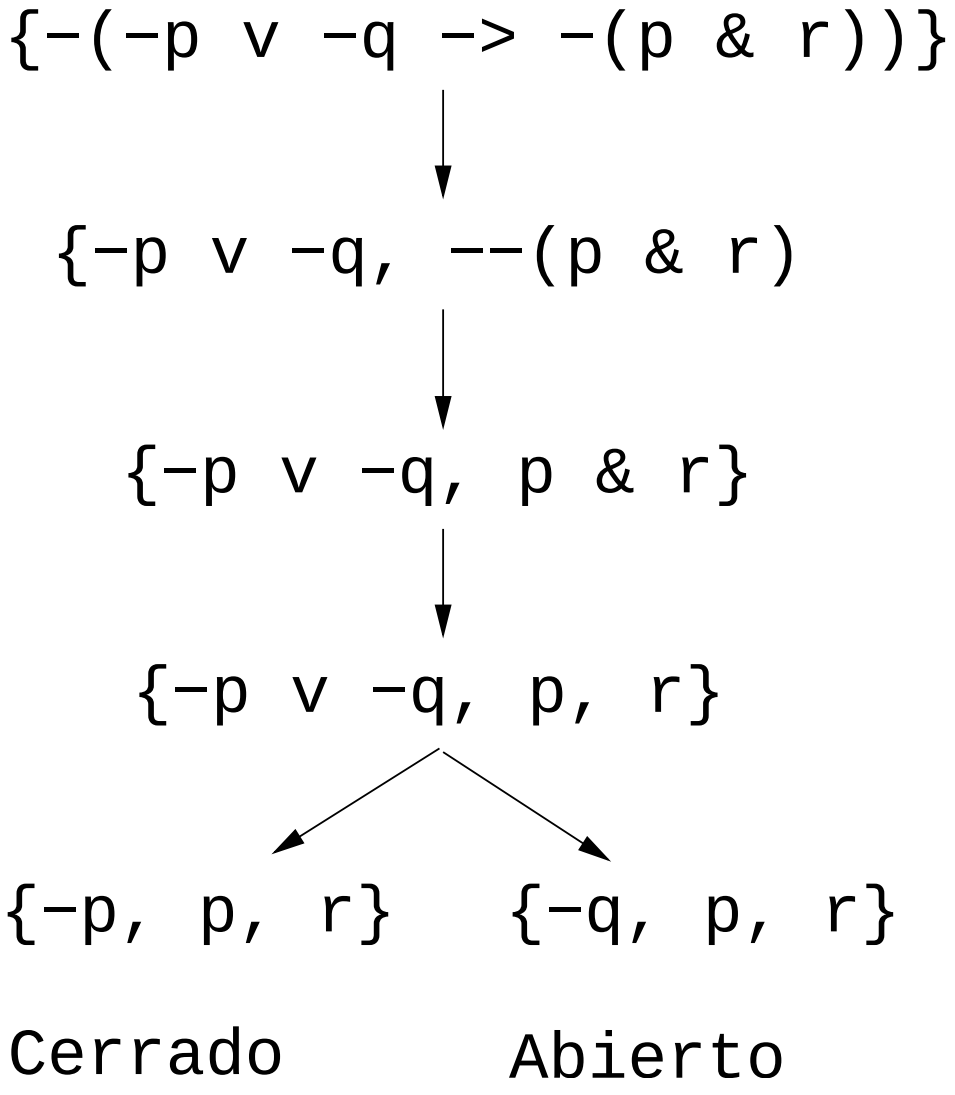
    syss $\{p, q, \neg p \vee \neg r\}$ es inconsistente

    syss $\{p, q, \neg p\}$ es inconsistente y

    $\{p, q, \neg r\}$ es inconsistente

Demostración por tableros semánticos

- Tablero semántico:



Notación uniforme: Literales y doble negación

- Literales
  - `literal(+F)` se verifica si la fórmula `F` es un literal (es decir, un átomo o la negación de un átomo).

```
literal(F) :-
    atom(F).
literal(-F) :-
    atom(F).
```

- Dobles negaciones
  - La fórmula F es una *doble negación* si existe una fórmula G tal que F es de la forma $\neg\neg G$.
  - Entonces $\models F \leftrightarrow G$.

Notación uniforme: Fórmulas alfa

- Las fórmulas *alfa*, junto con sus componentes, son las siguientes

$A_1 \wedge A_2$	A_1	A_2
$\neg(A_1 \rightarrow A_2)$	A_1	$\neg A_2$
$\neg(A_1 \vee A_2)$	$\neg A_1$	$\neg A_2$

- `alfa(+A,-A1,-A2)` se verifica si `A` es una fórmula alfa y sus componentes son `A1` y `A2`.

```
alfa(A1 & A2,      A1,  A2).
alfa(-(A1 => A2), A1,  -A2).
alfa(-(A1 v A2),  -A1, -A2).
```

Notación uniforme: Fórmulas beta

- Las fórmulas *beta*, junto con sus componentes, son las siguientes:

$B_1 \vee B_2$	B_1	B_2
$B_1 \rightarrow B_2$	$\neg B_1$	B_2
$\neg(B_1 \wedge B_2)$	$\neg B_1$	$\neg B_2$
$B_1 \leftrightarrow B_2$	$B_1 \wedge B_2$	$\neg B_1 \wedge \neg B_2$
$\neg(B_1 \leftrightarrow B_2)$	$B_1 \wedge \neg B_2$	$\neg B_1 \wedge B_2$

- `beta(+B,-B1,-B2)` se verifica si `B` es una fórmula beta y sus componentes son `B1` y `B2`.

```
beta(B1 v B2,       B1,         B2).
beta(B1 => B2,      –B1,        B2).
beta(–(B1 & B2),    –B1,        –B2).
beta(B1 <=> B2,     B1 & B2,    –B1 & –B2).
beta(–(B1 <=> B2), B1 & –B2, –B1 & B2).
```

Completación de tableros: Procedimiento de completación

Construcción del tablero de un conjunto de fórmulas S.

(I) El árbol cuyo único nodo tiene como etiqueta S es un tablero de S.

(C) Sea $\mathcal{T}$ un tablero de S y S_1 la etiqueta de una hoja de $\mathcal{T}$.

(C.1) Si S_1 es cerrado, entonces el árbol obtenido añadiendo como hijo de S_1 el nodo etiquetado con `cerrado` es un tablero de S.

(C.2) Si S_1 es abierto, entonces el árbol obtenido añadiendo como hijo de S_1 el nodo etiquetado con `abierto` es un tablero de S.

(C.3) Si S_1 contiene una doble negación $\neg\neg F$, entonces el árbol obtenido añadiendo como hijo de S_1 el nodo etiquetado con $(S_1 \setminus \{\neg\neg F\}) \cup \{F\}$ es un tablero de S.

(C.4) Si S_1 contiene una fórmula alfa F de componentes F_1 y F_2, entonces el árbol obtenido añadiendo como hijo de S_1 el nodo etiquetado con $(S_1 \setminus \{F\}) \cup \{F_1, F_2\}$ es un tablero de S.

(C.5) Si S_1 contiene una fórmula beta F de componentes F_1 y F_2, entonces el árbol obtenido añadiendo como hijos de S_1 los nodos etiquetados con $(S_1 \setminus \{F\}) \cup \{F_1\}$ y $(S_1 \setminus \{F\}) \cup \{F_2\}$ es un tablero de S.

Completación de tableros: Procedimiento de completación

- Un conjunto de fórmulas es *cerrado* si contiene una fórmula y su negación,
- Un conjunto de fórmulas es *abierto* si es un conjunto de literales que no contiene una fórmula y su negación.
- Un tablero semántico de S es *completo* si no se le puede aplicar ninguna de las reglas de expansión; es decir, todas sus hojas son abiertas o cerradas.

Completación de tableros: Tablero completo

- `tablero_completo(+S,-Tab)` se verifica si `Tab` es un tablero completo del conjunto `S`. Por ejemplo,

```
?- tablero_completo([- (-p v -q => - (p & r))],T).
T = t([- (-p v -q => - (p & r))],
      t([-p v -q, - -(p & r)],
        t([p & r, -p v -q],
          t([p, r, -p v -q],
            t([-p, p, r], cerrado, vacio),
            t([-q, p, r], abierto, vacio)),
          vacio),
        vacio),
      vacio)
Yes
```

- Representación: `t(S,Izq,Dcha)` representa el tablero de raíz `S`, rama izquierda `Izq` y rama derecha `Dcha`.

Completación de tableros

- Def. de `tablero_completo`:

```
tablero_completo(S,Tab) :-
    completación(t(S,_Izq,_Dcha),Tab).
```

- `completación(+Tab1,-Tab2)` se verifica si `Tab2` es una completación (i.e. un tablero completo) del tablero `Tab1`.

```
completación(t(Flas,Izq1,Dcha1),t(Flas,Izq3,Dcha3)) :-
    paso(t(Flas,Izq1,Dcha1),t(Flas,Izq2,Dcha2)), !,
    completación(Izq2,Izq3),
    completación(Dcha2,Dcha3).
completación(Tab,Tab).
```

Completación de tableros: Paso

- `paso(+Tab1,-Tab2)` se verifica si `Tab2` es un tablero obtenido aplicando una regla de completación al tablero `Tab1`.

```
paso(t(S1,_,_),t(S1, cerrado, vacio)) :-        % C1
    cerrada(S1), !.
paso(t(S1,_,_),t(S1, abierto, vacio)) :-        % C2
    lista_de_literales(S1), !.
paso(t(S1,_,_),t(S1, Izq, vacio)) :-            % C3
    regla_doble_negacion(S1, S2), !,
    Izq = t(S2, _, _).
paso(t(S1,_,_),t(S1, Izq, vacio)) :-            % C4
    regla_alfa(S1, S2), !,
    Izq = t(S2, _, _).
paso(t(S1,_,_),t(S1, Izq, Dcha)) :-             % C5
    regla_beta(S1, S2, S3),
    Izq  = t(S2, _, _),
    Dcha = t(S3, _, _).
```

Completación de tableros: Cerradas y lista de literales

- `cerrada(+S)` se verifica si `S` es una lista cerrada (i.e. que contiene una fórmula y su negación).

```
cerrada(S) :-
    member(-F,S),
    member(F,S).
```

- `lista_de_literales(+S)` se verifica si `S` es una lista de literales.

```
lista_de_literales([]).
lista_de_literales([F|S]) :-
    literal(F),
    lista_de_literales(S).
```

Completación de tableros: Doble negación

- `regla_doble_negacion(+S1,-S2)` que se verifica si `S1` es una lista de fórmulas que contiene una doble negación `--F` y `S2` es la lista de fórmulas obtenidas sustituyendo en `S1` la fórmula `--F` por `F`. Por ejmplo,

```
?– regla_doble_negacion([– –(q v r), p => r],L).
L = [q v r, p => r]
?– regla_doble_negacion([p v (q v r), p => r],L).
No
```

- Def. de `regla_doble_negacion`:

```
regla_doble_negacion(S1, [F|S2]) :–
    member(– –F, S1), !,
    delete(S1, – –F, S2).
```

Completación de tableros: Regla alfa

- `regla_alfa(+S1,-S2)` que se verifica si `S1` es una lista de fórmulas que contiene una fórmula alfa `A` y `S2` es la lista de fórmulas obtenidas sustituyendo en `S1` la fórmula `A` por sus componentes. Por ejemplo,

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

- Def. de `regla_alfa(+S1,-S2)`:

```
regla_alfa(S1, [A1,A2|S2]) :-
    member(A, S1),
    alfa(A, A1, A2), !,
    delete(S1, A, S2).
```

Completación de tableros: Regla beta

- `regla_beta(+S1,-S2,-S3)` que se verifica si `S1` es una lista de fórmulas que contiene al menos una fórmula beta `B` y `S2` es la lista de fórmulas obtenidas sustituyendo en `S1` la fórmula `B` por una de sus componentes y `S3`, por la otra. Por ejemplo,

```
?— regla_beta([p & (q v r), p => r],L1,L2).
L1 = [—p, p & (q v r)]
L2 = [r, p & (q v r)]
?— regla_beta([p & (q v r), —(p => r)],L1,L2).
No
```

- Def. de `regla_beta`:

```
regla_beta(S1, [B1|RS1], [B2|RS1]) :—
    member(B, S1),
    beta(B, B1, B2),
    delete(S1, B, RS1).
```

Tableros cerrados

- `es_cerrado(+Tab)` se verifica si `Tab` es un tablero cerrado (es decir, si todas sus hojas están etiquetadas con `cerrado` o `vacio`).

```
es_cerrado(t(_,Izq,Dcha)) :-
    es_cerrado(Izq),
    es_cerrado(Dcha).
es_cerrado(cerrado).
es_cerrado(vacio).
```

Prueba por tableros

- `prueba(+F,-Tab)` se verifica si `Tab` es una prueba de la fórmula `F` (es decir, un tablero completo cerrado de $\{\neg F\}$). Por ejemplo,

```
?- prueba(-p v -q => -(p & q),T).
T = t([- (-p v-q => - (p & q))],
      t([-p v-q, - -(p & q)],
        t([p & q, -p v -q],
          t([p, q, -p v -q],
            t([-p, p, q], cerrado, vacio),
            t([-q, p, q], cerrado, vacio)),
          vacio),
        vacio),
      vacio)
?- prueba(-p v -q => -p,T).
No
```

Teorema por tableros

- Prueba mediante tableros
  - Def. de `prueba`:

```
prueba(F,Tab) :-
    tablero_completo([-F],Tab), !,
    es_cerrado(Tab).
```

- Teorema mediante tableros
  - `es_teorema(+F)` se verifica si la fórmula `F` es teorema mediante tableros semánticos (es decir, F tiene una prueba mediante tableros). Por ejemplo,

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

  - Def. de `es_teorema`:

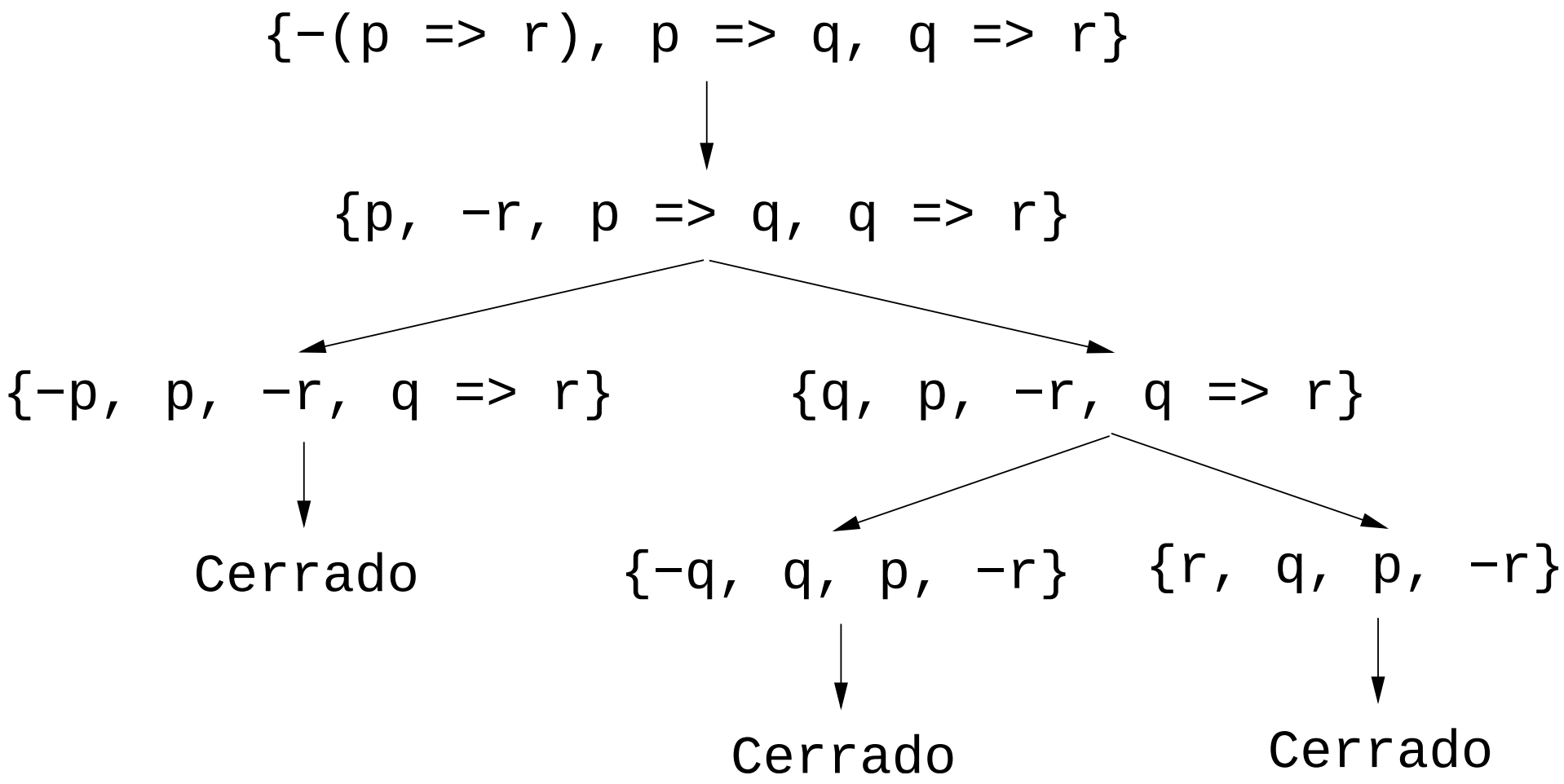
```
es_teorema(F) :-
    prueba(F,_Tab).
```

Deducción por tableros

- `prueba_deducible_tab(+S,+F,-Tab)` se verifica si `Tab` es una prueba por tableros semánticos de que la fórmula `F` es deducible del conjunto de fórmulas `S` (es decir, `Tab` es un tablero completo cerrado de $S \cup \{\neg F\}$). Por ejemplo,

```
?– prueba_deducible_tab([p => q, q => r], p => r,T).
T = t([– (p => r), p => q, q => r],
      t([p, –r, p => q, q => r],
        t([–p, p, –r, q => r], cerrado, vacio),
        t([q, p, –r, q => r],
          t([–q, q, p, –r], cerrado, vacio),
          t([r, q, p, –r], cerrado, vacio))),
      vacio)
Yes
?– prueba_deducible_tab([p => q, q => r], p <=> r,T).
No
```

Deducción por tableros



- Def. de `prueba_deducible_tab`:

```
prueba_deducible_tab(S,F,Tab) :-
    tablero_completo([-F|S],Tab), !,
    es_cerrado(Tab).
```

Deducibilidad por tableros

- `es_deducible_tab(+S,+F)` se verifica si la fórmula `F` es deducible (mediante tableros) del conjunto de fórmulas `S`.

```
es_deducible_tab(S,F) :-
    prueba_deducible_tab(S,F,_Tab).
```

Valores de verdad

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

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

Funciones de verdad (negación)

- Función de verdad de la negación:

i	$\neg i$
1	0
0	1

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

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

Funciones de verdad (conectivas binarias)

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

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

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`. Por ejemplo,

```
?– 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. del valor de una fórmula en una interpretación
  - 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) :-
    F =..[Op, A, B],
    valor(A, I, VA),
    valor(B, I, VB),
    función_de_verdad(Op, VA, VB, V).
```

Interpretaciones de una fórmula

- `interpretaciones_fórmula(+F,-L)` se verifica si `L` es el conjunto de las interpretaciones principales de la fórmula `F`. Por 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).
```

Interpretación de una fórmula

- `interpretación_fórmula(?I,+F)` se verifica si `I` es una interpretación de la fórmula `F`. Por 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)]
Yes
```

- Def. de `interpretación_fórmula`:

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

Símbolos de una fórmula

- `símbolos_fórmula(+F,?U)` se verifica si `U` es el conjunto ordenado de los símbolos proposicionales de la fórmula `F`. Por ejemplo,

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

- Def. de `símbolos_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) :–
    F =.. [_Op,A,B],
    símbolos_fórmula_aux(A,UA),
    símbolos_fórmula_aux(B,UB),
    union(UA,UB,U).
```

Interpretación de una lista de símbolos

- `interpretación_símbolos(+L,-I)` se verifica si `I` es una interpretación de la lista de símbolos proposicionales `L`. Por ejemplo,

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

- Def. de `interpretación_símbolos`

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

Comprobación de modelo de una fórmula

- `es_modelo_fórmula(+I,+F)` se verifica si la interpretación `I` es un modelo de la fórmula `F` (es decir, si el valor de F en I es verdadero). Por ejemplo,

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

- Def. de `es_modelo_fórmula`:

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

Cálculo de los modelos de una fórmula

- `modelo_fórmula(?I,+F)` se verifica si `I` es un modelo principal de la fórmula `F`. Por 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`. Por 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

- `es_satisfacible(+F)` se verifica si la fórmula `F` es satisfacible (es decir, si tiene modelos). Por ejemplo,

```
?- 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

- `contramodelo_fórmula(?I,+F)` se verifica si `I` es un contramodelo principal de la fórmula `F` (es decir, si `I` es una interpretación principal de F que no es modelo de F). Por ejemplo,

```
?– 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),
    not(es_modelo_fórmula(I,F)).
```

Validez. Tautologías

- `es_tautología(+F)` se verifica si la fórmula `F` es una tautología (es decir, si todas las interpretaciones son modelos de F). Por ejemplo,

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

- Def. de `es_tautología`:

```
es_tautología(F) :–
    not(contramodelo_fórmula(_I,F)).
```

- Definición alternativa:

```
es_tautología_alt(F) :–
    not(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`. Por 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).
```

Interpretación de conjuntos

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

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

- Def. de `interpretación_conjunto`:

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

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`. Por 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).
```

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` (es decir, si `I` es modelo de todas las fórmulas de `S`). Por ejemplo,

```
?– es_modelo_conjunto([(p,1),(q,0),(r,1)], [(p v q) & (–q
Yes
?– es_modelo_conjunto([(p,0),(q,1),(r,0)], [(p v q) & (–q
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 modelo de conjuntos de fórmulas

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

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

- `consistente(+S)` se verifica si el conjunto de fórmulas `S` es consistente (es decir, tiene modelos). Por ejemplo,

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

- `inconsistente(+S)` se verifica si el conjunto de fórmulas `S` es inconsistente (es decir, no tiene modelos).
- Def. de `consistente` e `inconsistente`:

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

inconsistente(S) :–
    not(modelo_conjunto(_I,S)).
```

Consecuencia lógica

- `es_consecuencia(+S,+F)` se verifica si la fórmula `F` es consecuencia del conjunto de fórmulas `S` (es decir, si todos los modelos de `S` son modelos de `F`). Por ejemplo,

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

- Def. de `es_consecuencia`:

```
es_consecuencia(S,F) :–
    not(contramodelo_consecuencia(S,F,_I)).
```

Consecuencia lógica

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

```
?– 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),
    not(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.

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
  - Cap. 4.1: Tableros semánticos
- 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)