Lógica proposicional: Sintaxis y semántica

José A. Alonso Jiménez,
José L. Ruiz Reina y
Francisco J. Martín Mateos

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Introducción

- **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**
    * **El animal es una cebra.**

Introducción

- **Solución con OTTER**
  - **Representación en OTTER** (`animales.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).
```

Introducción

- **Solución con OTTER**

```
> otter <animales.in
-------> sos clausifies to:
list(sos).
1 [] -tiene_pelos | es_mamifero.
2 [] -da_leche | es_mamifero.
3 [] -es_mamifero | -tiene_pezuñas | 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_pezuñas.
9 [] tiene_rayas_negras.
10 [] -es_cebra.
end_of_list.
set(binary_res).
   dependent: set(factor).
   dependent: set(unit_deletion).

======= end of input processing =======
```

Introducción

```
=========== start of search ===========

given clause #1: (wt=1) 7 [] tiene_pelos.

given clause #2: (wt=1) 8 [] tiene_pezuñas.

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.
** KEPT (pick-wt=1): 11 [binary,1.1,7.1] es_mamifero.
11 back subsumes 2.
11 back subsumes 1.

given clause #6: (wt=1) 11 [binary,1.1,7.1] es_mamifero.

given clause #7: (wt=3) 3 [] -es_mamifero
                                 | -tiene_pezuñas
                                 | es_ungulado.
** KEPT (pick-wt=1): 12 [binary,3.1,11.1,unit_del,8]
                        es_ungulado.
12 back subsumes 4.
12 back subsumes 3.

given clause #8: (wt=1) 12 [binary,3.1,11.1,unit_del,8]
                           es_ungulado.

given clause #9: (wt=3) 6 [] -es_ungulado
                                 | -tiene_rayas_negras
                                 | es_cebra.
** KEPT (pick-wt=0): 13 [binary,6.1,12.1,unit_del,9,10]
                        $F.
```

Introducción

```
-------- PROOF --------
Length of proof is 2.  Level of proof is 2.

---------------- PROOF ----------------

1 [] -tiene_pelos | es_mamifero.
3 [] -es_mamifero | -tiene_pezuñas | es_ungulado.
6 [] -es_ungulado | -tiene_rayas_negras | es_cebra.
7 [] tiene_pelos.
8 [] tiene_pezuñas.
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,6.1,12.1,unit_del,9,10] $F.

------------ end of proof -------------
```

Introducción

• Solución con Prolog

• Representación en Prolog (animales.pl)

```
es_mamifero :- tiene_pelos.
es_mamifero :- da_leche.
es_ungulado :- es_mamifero, tiene_pezuñas.
es_ungulado :- es_mamifero, rumia.
es_jirafa   :- es_ungulado, tiene_cuello_largo.
es_cebra    :- es_ungulado, tiene_rayas_negras.

tiene_pelos.
tiene_pezuñas.
tiene_rayas_negras.
```

• Solución con Prolog

```
?- [animales].
animales compiled, 0.02 sec, 2,380 bytes.
Yes

?- es_cebra.
Yes

?- es_jirafa.
No
```

Elementos de una lógica

- **Elementos de una lógica:**
  - Sintaxis: ¿qué expresiones son fórmulas?
  - Semántica: ¿qué significa que una fórmula F es consecuencia de un conjunto de fórmulas S?: $S \models F$
  - Cálculo: ¿qué significa que una fórmual F puede deducirse a partir de un conjunto de fórmulas S?: $S \vdash F$
- **Propiedades:**
  - Potencia expresiva
  - Adecuación: $S \vdash F \Longrightarrow S \models F$
  - Completitud: $S \models F \Longrightarrow S \vdash F$
  - Decidibilidad
  - Complejidad

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
  - $\neg F$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$, $(F \leftrightarrow G)$
- **Ejemplos de fórmulas proposicionales:**
  - (rumia $\rightarrow$ es_rumiante)
  - $((p \rightarrow q) \vee (q \rightarrow p))$

Sintaxis de la lógica proposicional

- **Metavariables:**
  - SP: conjunto de símbolos proposicionales
  - Prop: conjunto de fórmulas proposicionales
  - Símbolos proposicionales: $p, p_0, p_1, \dots, q, q_0, q_1, \dots$
  - Fórmulas proposicionales: $F, F_0, F_1, \dots, G, G_0, G_1, \dots$
- **Eliminación de paréntesis:**
  - Eliminación de paréntesis externos:
    rumia $\wedge$ corre $\Longleftrightarrow$ (rumia $\wedge$ corre)
  - Precedencia: $\neg, \wedge, \vee \rightarrow, \leftrightarrow$
    $p \wedge \neg q \vee r \rightarrow s \Longleftrightarrow ((p \wedge \neg q) \vee r) \rightarrow s$
    $p \vee \neg q \wedge r \rightarrow \neg s \vee t \Longleftrightarrow (p \vee (\neg q \wedge r)) \rightarrow (\neg s \vee t)$
  - Asociatividad: $\wedge$ y $\vee$ asocian por la derecha
    $p \wedge q \wedge r \Longleftrightarrow p \wedge (q \wedge r)$
- **Las fórmulas proposicionales como gramática libre de contexto.**

Sintaxis de la lógica proposicional

- **Representación de conectivas:**

```
(defconstant *simbolo-de-negacion*      '-)
(defconstant *simbolo-de-conjuncion*    '&)
(defconstant *simbolo-de-disyuncion*    '/)
(defconstant *simbolo-de-implicacion*   '->)
(defconstant *simbolo-de-equivalencia*  '<->)
```

- **Construcción de fórmulas:**

```
(defun negacion (F)
  (list *simbolo-de-negacion* F))

(defun conjuncion (F G)
  (list F *simbolo-de-conjuncion* G))

(defun disyuncion (F G)
  (list F *simbolo-de-disyuncion* G))

(defun implicacion (F G)
  (list F *simbolo-de-implicacion* G))

(defun equivalencia (F G)
  (list F *simbolo-de-equivalencia* G))
```

Sintaxis de la lógica proposicional

- **Elementos de las fórmulas:**

```
(defun op (exp)
  (if (= (length exp) 2)
      (first exp)
      (second exp)))

(defun arg1 (exp)
  (if (= (length exp) 2)
      (second exp)
      (first exp)))

(defun arg2 (exp)
  (third exp))
```

- **Clasificación de fórmulas:**

```
(defun tipo (F)
  (if (symbolp F)
      'es-atomica
      (let ((op (op F)))
        (cond ((eql op *simbolo-de-negacion*)
               'es-negacion)
              ((eql op *simbolo-de-conjuncion*)
               'es-conjuncion)
              ((eql op *simbolo-de-disyuncion*)
               'es-disyuncion)
              ((eql op *simbolo-de-implicacion*)
               'es-implicacion)
              ((eql op *simbolo-de-equivalencia*)
               'es-equivalencia)))))
```

Sintaxis de la lógica proposicional

```
(defun es-atomica (F)
  (symbolp F))

(defun es-negacion (F)
  (and (listp F)
       (eql (op F) *simbolo-de-negacion*)))

(defun es-conjuncion (F)
  (and (listp F)
       (eql (op F) *simbolo-de-conjuncion*)))

(defun es-disyuncion (F)
  (and (listp F)
       (eql (op F) *simbolo-de-disyuncion*)))

(defun es-implicacion (F)
  (and (listp F)
       (eql (op F) *simbolo-de-implicacion*)))

(defun es-equivalencia (F)
  (and (listp F)
       (eql (op F) *simbolo-de-equivalencia*)))
```

Símbolos de una fórmula

- **Símbolos proposicionales de una fórmula:**
  - **Ejemplo:** $\mathrm{SP}((\boldsymbol{p} \vee \boldsymbol{q}) \wedge (\neg \boldsymbol{q} \vee \boldsymbol{r})) = \{\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}\}$
  - **Definición:**

$$
\begin{array}{ll}
\mathrm{SP}(\boldsymbol{p}) & = \{\boldsymbol{p}\} \\
\mathrm{SP}(\neg \boldsymbol{F}) & = \mathrm{SP}(\boldsymbol{F}) \\
\mathrm{SP}(\boldsymbol{F} \wedge \boldsymbol{G}) & = \mathrm{SP}(\boldsymbol{F}) \cup \mathrm{SP}(\boldsymbol{G}) \\
\mathrm{SP}(\boldsymbol{F} \vee \boldsymbol{G}) & = \mathrm{SP}(\boldsymbol{F}) \cup \mathrm{SP}(\boldsymbol{G}) \\
\mathrm{SP}(\boldsymbol{F} \rightarrow \boldsymbol{G}) & = \mathrm{SP}(\boldsymbol{F}) \cup \mathrm{SP}(\boldsymbol{G}) \\
\mathrm{SP}(\boldsymbol{F} \leftrightarrow \boldsymbol{G}) & = \mathrm{SP}(\boldsymbol{F}) \cup \mathrm{SP}(\boldsymbol{G})
\end{array}
$$

Símbolos de una fórmula

- **Símbolos proposicionales de una fórmula:**

```
;;; (simbolos-proposicionales-formula '((p & q) -> r))  =>  (P Q R)
(defun simbolos-proposicionales-formula (F)
  (case (tipo F)
    (es-atomica  (list F))
    (es-negacion (simbolos-proposicionales-formula (arg1 F)))
    (t           (n-union (simbolos-proposicionales-formula (arg1 F))
                          (simbolos-proposicionales-formula (arg2 F))))))

;;; (n-union '(a b c) '(a c d))  =>  (A B C D)
(defun n-union (conjunto-1 conjunto-2 &key (test #'eql))
  (append conjunto-1
          (set-difference conjunto-2 conjunto-1 :test test)))
```

Semántica: Funciones de verdad

- **Valores de verdad:**
  - **0: falso**
  - **1: verdadero**
- **Funciones de verdad:**
  - $\mathbf{FV}_\neg : \{0,1\} \to \{0,1\}$ t.q. $\mathbf{FV}_\neg(i) = \begin{cases} 1, & \text{si } i = 0; \\ 0, & \text{si } i = 1. \end{cases}$
  - $\mathbf{FV}_\wedge : \{0,1\}^2 \to \{0,1\}$ t.q. $\mathbf{FV}_\wedge(i,j) = \begin{cases} 1, & \text{si } i = j = 1; \\ 0, & \text{en otro caso.} \end{cases}$
  - $\mathbf{FV}_\vee : \{0,1\}^2 \to \{0,1\}$ t.q. $\mathbf{FV}_\vee(i,j) = \begin{cases} 0, & \text{si } i = j = 0; \\ 1, & \text{en otro caso.} \end{cases}$
  - $\mathbf{FV}_\to : \{0,1\}^2 \to \{0,1\}$ t.q. $\mathbf{FV}_\to(i,j) = \begin{cases} 0, & \text{si } i = 1, j = 0 \\ 1, & \text{en otro caso.} \end{cases}$
  - $\mathbf{FV}_\leftrightarrow : \{0,1\}^2 \to \{0,1\}$ t.q. $\mathbf{FV}_\leftrightarrow(i,j) = \begin{cases} 1, & \text{si } i = j; \\ 0, & \text{en otro caso.} \end{cases}$

Semántica: Funciones de verdad

- **Funciones de verdad en Lisp:**

```
(defun funcion-de-verdad-de-negacion (i)
  (if (= i 0) 1 0))

(defun funcion-de-verdad-de-conjuncion (i j)
  (if (= i j 1) 1 0))

(defun funcion-de-verdad-de-disyuncion (i j)
  (if (= i j 0) 0 1))

(defun funcion-de-verdad-de-implicacion (i j)
  (if (and (= i 1) (= j 0)) 0 1))

(defun funcion-de-verdad-de-equivalencia (i j)
  (if (= i j) 1 0))
```

Semántica: Interpretación y significado

- **Interpretación:**
  - Definición: I es una interpretación si I es un conjunto de símbolos proposicionales.
  - Significado informal: Los símbolos de I son verdaderos y los restantes falsos.
- **Interpretaciones: Conjunto de todas las interpretaciones.**
- **Significado: Definición:**
  $\mathrm{sig} : \textsc{Prop} \times \textsc{Interpretaciones} \to \{0, 1\}$
  - $\mathrm{sig}(p, I) = \begin{cases} 1, & \text{si } p \text{ pertenece a } I \\ 0, & \text{en caso contrario} \end{cases}$
  - $\mathrm{sig}(\neg F, I) = \mathrm{FV}_{\neg}(\mathrm{sig}(F, I))$
  - $\mathrm{sig}(F \wedge G, I) = \mathrm{FV}_{\wedge}(\mathrm{sig}(F, I), \mathrm{sig}(G, I))$
  - $\mathrm{sig}(F \vee G, I) = \mathrm{FV}_{\vee}(\mathrm{sig}(F, I), \mathrm{sig}(G, I))$
  - $\mathrm{sig}(F \rightarrow G, I) = \mathrm{FV}_{\rightarrow}(\mathrm{sig}(F, I), \mathrm{sig}(G, I))$
  - $\mathrm{sig}(F \leftrightarrow G, I) = \mathrm{FV}_{\leftrightarrow}(\mathrm{sig}(F, I), \mathrm{sig}(G, I))$

Semántica: Interpretación y significado

- **Significado: Ejemplos:** $F = (p \lor q) \land (\neg q \lor r)$
  - **Interpretación 1:** $I = \{p, r\}$

$$
\begin{aligned}
&\mathrm{sig}(F, I) = \\
&= \mathrm{sig}((p \lor q) \land (\neg q \lor r), I) = \\
&= \mathrm{FV}_\land(\mathrm{sig}(p \lor q, I), \mathrm{sig}(\neg q \lor r, I)) = \\
&= \mathrm{FV}_\land(\mathrm{FV}_\lor(\mathrm{sig}(p, I), \mathrm{sig}(q, I)), \mathrm{FV}_\lor(\mathrm{sig}(\neg q, I), \mathrm{sig}(r, I))) = \\
&= \mathrm{FV}_\land(\mathrm{FV}_\lor(1, 0), \mathrm{FV}_\lor(\mathrm{FV}_\neg(\mathrm{sig}(q, I), 1))) = \\
&= \mathrm{FV}_\land(1, \mathrm{FV}_\lor(\mathrm{FV}_\neg(0), 1)) = \\
&= \mathrm{FV}_\land(1, \mathrm{FV}_\lor(1, 1)) = \\
&= \mathrm{FV}_\land(1, 1) = \\
&= 1
\end{aligned}
$$

$$
\begin{array}{ccccccccc}
(p & \lor & q) & \land & (\neg q & \lor & r) \\
(1 & \lor & 0) & \land & (\neg 0 & \lor & 1) \\
 & 1 & & \land & (1 & \lor & 1) \\
 & 1 & & \land & & 1 & \\
 & & & 1 & & &
\end{array}
$$

  - **Interpretación 2:** $J = \{r\}$
    $\mathrm{sig}(F, J) = 0$

$$
\begin{array}{ccccccccc}
(p & \lor & q) & \land & (\neg q & \lor & r) \\
(0 & \lor & 0) & \land & (\neg 0 & \lor & 1) \\
 & 0 & & \land & (1 & \lor & 1) \\
 & 0 & & \land & & 1 & \\
 & & & 0 & & &
\end{array}
$$

Semántica: Interpretación y significado

- **Procedimiento** `significado`:

```
;;; (significado '((p / q) & ((- q) / r)) '(p r))    =>  1
;;; (significado '((p / q) & ((- q) / r)) '(r))      =>  0
(defun significado (F I)
  (case (tipo F)
    (es-atomica      (if (member F I) 1 0))
    (es-negacion     (funcion-de-verdad-de-negacion
                      (significado (arg1 F) I)))
    (es-conjuncion   (funcion-de-verdad-de-conjuncion
                      (significado (arg1 F) I) (significado (arg2 F) I)))
    (es-disyuncion   (funcion-de-verdad-de-disyuncion
                      (significado (arg1 F) I) (significado (arg2 F) I)))
    (es-implicacion  (funcion-de-verdad-de-implicacion
                      (significado (arg1 F) I) (significado (arg2 F) I)))
    (es-equivalencia (funcion-de-verdad-de-equivalencia
                      (significado (arg1 F) I) (significado (arg2 F) I)))
    (t               (error "SIGNIFICADO: La expresion ~s no es una formula"
                            F))))
```

Interpretaciones de una fórmula

- **Definición:**
  $\textbf{Interpretaciones}(\boldsymbol{F}) = \{\boldsymbol{I} : \boldsymbol{I} \subseteq \mathrm{SP}(\boldsymbol{F})\}$

- **Ejemplo:**
  $\textbf{Interpretaciones}((\boldsymbol{p} \vee \boldsymbol{q}) \wedge (\neg \boldsymbol{q} \vee \boldsymbol{r})) =$
  $= \{\{\}, \{\boldsymbol{r}\}, \{\boldsymbol{q}\}, \{\boldsymbol{q}, \boldsymbol{r}\}, \{\boldsymbol{p}\}, \{\boldsymbol{p}, \boldsymbol{r}\}, \{\boldsymbol{p}, \boldsymbol{q}\}, \{\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}\}\}$

- **Número de interpretaciones de una fórmula:**
  $|\textbf{Interpretaciones}(\boldsymbol{F})| = \mathbf{2}^{|\mathrm{SP}(\boldsymbol{F})|}$

- **Procedimiento de cálculo**

```
;;; (interpretaciones-formula '((p / q) & ((- q) / r)))
;;; => (NIL (R) (Q) (Q R) (P) (P R) (P Q) (P Q R))
(defun interpretaciones-formula (F)
  (subconjuntos (simbolos-proposicionales-formula F)))

;;; (subconjuntos '(p q r))
;;; => (NIL (R) (Q) (Q R) (P) (P R) (P Q) (P Q R))
(defun subconjuntos (x)
  (if (null x)
      (list ())
      (let ((primero (first x))
            (subconjuntos-del-resto (subconjuntos (rest x))))
        (append subconjuntos-del-resto
                (mapcar #'(lambda (a) (cons primero a))
                        subconjuntos-del-resto)))))
```

Modelos de fórmulas

- **Modelo:**
  - **Def.:** I **modelo de** $F \iff \mathrm{sig}(F, I) = 1$
  - **Representación:** $I \models F$
  - **Ejemplo:** $\{p, r\} \models (p \lor q) \land (\neg q \lor r)$
- **Procedimiento de decisión de modelos:**

```
;;; (es-modelo-formula '(p r) '((p / q) & ((- q) / r)))
;;; => T
;;; (es-modelo-formula '(r)   '((p / q) & ((- q) / r)))
;;; => NIL
(defun es-modelo-formula (I F)
  (= (significado F I) 1))
```

- **Contramodelo:**
  - **Def.:** I **contramodelo de** $F \iff \mathrm{sig}(F, I) = 0.$
  - **Representación:** $I \not\models F$
  - **Ejemplo:** $\{r\} \not\models (p \lor q) \land (\neg q \lor r)$

Modelos de fórmulas

- **Modelos de una fórmula:**
  - **Def.:** Modelos(F) = $\{I \in$ Interpretaciones$(F) : I \models F\}$
  - **Ejemplo:**
    Modelos($(p \vee q) \wedge (\neg q \vee r)$)
    $= \{\{q, r\}, \{p\}, \{p, r\}, \{p, q, r\}\}$

- **Procedimiento de cálculo de modelos:**

```
;;; (modelos-formula '(p -> q))
;;; => (NIL (Q) (P Q))
;;; (modelos-formula '(p & (- p)))
;;; => NIL
;;; (modelos-formula '((p -> q) / (q -> p)))
;;; => (NIL (P) (Q) (Q P))
(defun modelos-formula (F)
  (remove-if-not #'(lambda (I) (es-modelo-formula I F))
                 (interpretaciones-formula F)))
```

- **Tablas de verdad**

Modelos de fórmulas

```
> (trace es-modelo-formula)
> (modelos-formula '((p & q) -> (r <-> (- s))))
(ES-MODELO-FORMULA 'NIL       '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(S)       '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(R)       '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(R S)     '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(Q)       '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(Q S)     '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(Q R)     '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(Q R S)   '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(P)       '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(P S)     '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(P R)     '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(P R S)   '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(P Q)     '((P & Q) -> (R <-> (- S)))) ==> NIL
(ES-MODELO-FORMULA '(P Q S)   '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(P Q R)   '((P & Q) -> (R <-> (- S)))) ==> T
(ES-MODELO-FORMULA '(P Q R S) '((P & Q) -> (R <-> (- S)))) ==> NIL
(NIL (S) (R)(R S) (Q)(Q S)(Q R)(Q R S) (P)(P S)(P R)(P R S)(P Q S)(P Q R))
```

Fórmulas válidas y satisfacibles

- **Fórmula válida (tautología):**
  - Def.:
    F es válida $\iff$ todo interpretación de F es modelo de F
  - Representación: $\models F$
  - Ejemplo: $\models (p \to q) \lor (q \to p)$
- **Tablas de verdad:**

p	q	r	$(p \to q)$	$(q \to r)$	$(p \to q) \lor (q \to r)$
1	1	1	1	1	1
1	1	0	1	0	1
1	0	1	0	1	1
1	0	0	0	1	1
0	1	1	1	1	1
0	1	0	1	0	1
0	0	1	1	1	1
0	0	0	1	1	1

Fórmulas válidas y satisfacibles

- **Procedimiento de decisión de validez**

```
;;; (es-valida '(p -> p))                    =>  T
;;; (es-valida '((p -> q) / ( q -> p)))  =>  T
;;; (es-valida '(p -> q))                    =>  NIL
(defun es-valida (F)
  (every #'(lambda (I) (es-modelo-formula I F))
         (interpretaciones-formula F)))
```

- **Ejemplos de cálculo**

```
> (trace es-modelo-formula)
(ES-MODELO-FORMULA)

> (es-valida '(((p -> q) & p) -> q))
(ES-MODELO-FORMULA 'NIL '(((P -> Q) & P) -> Q))
ES-MODELO-FORMULA ==> T
(ES-MODELO-FORMULA '(Q) '(((P -> Q) & P) -> Q))
ES-MODELO-FORMULA ==> T
(ES-MODELO-FORMULA '(P) '(((P -> Q) & P) -> Q))
ES-MODELO-FORMULA ==> T
(ES-MODELO-FORMULA '(P Q) '(((P -> Q) & P) -> Q))
ES-MODELO-FORMULA ==> T
T

> (es-valida '((p -> q) & (p & (- q))))
(ES-MODELO-FORMULA 'NIL '((P -> Q) & (P & (- Q))))
ES-MODELO-FORMULA ==> NIL
NIL
```

Fórmulas válidas y satisfacibles

- **Fórmula satisfacible:**
  - Def.: F es satisfacible $\Longleftrightarrow$ F tiene modelo
  - Def.: F es insatisfacible $\Longleftrightarrow$ F no tiene modelo
  - Ejemplo:
    $(p \to q) \land (q \to r)$ es satisfacible
    $p \land \neg p$ es insatisfacible
- **Los problemas de satisfacibilidad y validez:**
  - Problema de la satisfacibilidad: Dada F determinar si es satisfacible.
  - Problema de la validez: Dada F determinar si es válida
- **Relaciones entre validez y satisfacibilidad:**
  - F es válida $\Longleftrightarrow$ $\neg F$ es insatisfacible
  - F es válida $\Longrightarrow$ F es satisfacible
  - F es satisfacible $\not\Longrightarrow$ $\neg F$ es insatisfacible
- **El problema de la satisfacibilidad es NP–completo**

Fórmulas válidas y satisfacibles

- **Procedimiento de decisión de insatisfacibilidad:**

```
;;; (es-insatisfacible '(p & (- p)))
;;; =>  T
;;; (es-insatisfacible '((p -> q) & (q -> r)))
;;; =>  NIL
(defun es-insatisfacible (F)
  (every #'(lambda (I) (not (es-modelo-formula I F)))
         (interpretaciones-formula F)))
```

- **Ejemplos de cálculo**

```
> (trace es-modelo-formula)
(ES-MODELO-FORMULA)

> (es-insatisfacible '(((p -> q) & p) -> q))
(ES-MODELO-FORMULA 'NIL '(((P -> Q) & P) -> Q))
ES-MODELO-FORMULA ==> T
NIL

> (es-insatisfacible '((p -> q) & (p & (- q))))
(ES-MODELO-FORMULA 'NIL '((P -> Q) & (P & (- Q))))
ES-MODELO-FORMULA ==> NIL
(ES-MODELO-FORMULA '(Q) '((P -> Q) & (P & (- Q))))
ES-MODELO-FORMULA ==> NIL
(ES-MODELO-FORMULA '(P) '((P -> Q) & (P & (- Q))))
ES-MODELO-FORMULA ==> NIL
(ES-MODELO-FORMULA '(P Q) '((P -> Q) & (P & (- Q))))
ES-MODELO-FORMULA ==> NIL
T
```

Símbolos de un conjunto de fórmulas

- **Símbolos proposicionales de un conjunto de fórmulas:**
  - **Def.:** $\mathrm{SP}(\boldsymbol{S}) = \bigcup\{\mathrm{SP}(\boldsymbol{F}) : \boldsymbol{F} \in \boldsymbol{S}\}$
  - **Ejemplo:** $\mathrm{SP}(\{(\boldsymbol{p} \vee \boldsymbol{q}) \wedge (\neg \boldsymbol{q} \vee \boldsymbol{r}), \boldsymbol{p} \rightarrow \boldsymbol{a}\}) = \{\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{a}\}$

- **Símbolos proposicionales de un conjunto de fórmulas:**

```
;;; (setf S '(((p & q) -> r) (p -> s)))
;;; (simbolos-proposicionales-conjunto S) =>  (Q R P S)
(defun simbolos-proposicionales-conjunto (S)
  (union-general
    (mapcar #'simbolos-proposicionales-formula S)))

;;; (union-general ())                       =>  NIL
;;; (union-general '((a)))                   =>  (A)
;;; (union-general '((a) (a b) (b c)))       =>  (A B C)
(defun union-general (x)
  (if (null x)
      x
      (reduce #'n-union x)))
```

Interpretaciones de un conjunto de fórmulas

- **Interpretaciones de un conjunto de fórmulas:**
  - **Def.: Interpretaciones**$(S) = \{I : I \subseteq \mathrm{SP}(S)\}$
  - **Ejemplo:**
    Interpretaciones$(\{p \to q, q \to r\}) =$
    $= \{\{\}, \{r\}, \{q\}, \{q, r\}, \{p\}, \{p, r\}, \{p, q\}, \{p, q, r\}\}$

- **Interpretaciones de un conjunto de fórmulas:**

```
;;; (interpretaciones-conjunto '((p -> q) (q -> r)))
;;; => (NIL (R) (Q) (Q R) (P) (P R) (P Q) (P Q R))
(defun interpretaciones-conjunto (S)
  (subconjuntos (simbolos-proposicionales-conjunto S)))
```

Modelos de conjuntos de fórmulas

- **Modelo de un conjunto de fórmulas:**
  - Def.: I modelo de S $\Longleftrightarrow$ para toda F de S, $I \models F$
  - Representación: $I \models S$
  - Ejemplo: $\{p, r\} \models \{(p \vee q) \wedge (\neg q \vee r), q \rightarrow r\}$

- **Procedimiento de decisión de modelo de un conjunto de fórmulas:**

```
(es-modelo-conjunto '(p r)
                    '(((p / q) & ((- q) / r)) (q -> r)))
=>  T
(es-modelo-conjunto '(p r)
                    '(((p / q) & ((- q) / r)) (r -> q)))
=>  NIL
(defun es-modelo-conjunto (I S)
  (every #'(lambda (F) (es-modelo-formula I F))
         S))
```

- **Contramodelo de un conjunto de fórmulas:**
  - Def.: I contramodelo de S $\Longleftrightarrow$ para alguna F de S, $I \not\models F$
  - Representación: $I \not\models S$
  - Ejemplo: $\{p, r\} \not\models \{(p \vee q) \wedge (\neg q \vee r), r \rightarrow q\}$

Modelos de conjuntos de fórmulas

* **Modelos de un conjunto de fórmulas**
  * **Def.:** Modelos$(S) = \{I \in \text{Interpretaciones}(S) : I \models S\}$
  * **Ejemplo:** Modelos$(\{(p \vee q) \wedge (\neg q \vee r), p \rightarrow r\}) = \{\{q, r\}, \{p, q, r\}\}$
* **Procedimiento de cálculo de modelos de un conjunto de fórmulas:**

```
;;; (modelos-conjunto '(((p / q) & ((- q) / r)) (q -> r)))
;;; => ((Q R) (P) (P R) (P Q R))
;;; (modelos-conjunto '(((p / q) & ((- q) / r)) (r -> q)))
;;; => ((R Q) (P) (P R Q))
(defun modelos-conjunto (S)
  (remove-if-not #'(lambda (I) (es-modelo-conjunto I S))
                 (interpretaciones-conjunto S)))
```

Conjuntos consistentes e inconsistentes de fórmulas

- **Conjunto consistente de fórmulas:**
  - **Def.: S es consistente S tiene modelo**
  - **Def.: S es inconsistente S no tiene modelo**
  - **Ejemplo: $\{(p \vee q) \wedge (-q \vee r), p \to r\}$ es consistente**
- **Procedimiento de decisión de conjunto consistente:**

```
;;; (es-consistente '((p / q) & ((- q) / r) (p \lif r)))        =>  T
;;; (es-consistente '((p / q) & ((- q) / r) (p \lif r) (- r)))  =>  NIL
(defun es-consistente (S)
  (some #'(lambda (I) (es-modelo-conjunto I S))
        (interpretaciones-conjunto S)))

;;; (es-inconsistente '((p / q) & ((- q) / r) (p -> r)))        =>  NIL
;;; (es-inconsistente '((p / q) & ((- q) / r) (p -> r) (- r)))  =>  T
(defun es-inconsistente (S)
  (every #'(lambda (I) (not (es-modelo-conjunto I S)))
         (interpretaciones-conjunto S)))
```

Consecuencia lógica

- **Consecuencia lógica:**
  - Def.: F es consecuencia de $S \iff$ los modelos de S son modelos de F
  - Representación: $S \models F$
  - Ejemplo: $\{p \rightarrow q, q \rightarrow r\} \models p \rightarrow r$

- **Procedimiento de decisión de consecuencia lógica:**

```
;;; (es-consecuencia '((p -> q) (q -> r)) '(p -> r))
;;; =>  T
;;; (es-consecuencia '(p) '(p & q))
;;; =>  NIL
(defun es-consecuencia (S F)
  (every #'(lambda (I) (if (es-modelo-conjunto I S)
                           (es-modelo-formula I F)
                           t))
         (interpretaciones-conjunto (cons F S))))
```

Consecuencia lógica

- **Ejemplos de cálculo:**

```
> (trace es-modelo-formula)
> (es-consecuencia '((p -> q) (- q)) '(- p))
(ES-MODELO-FORMULA 'NIL '(P -> Q))
ES-MODELO-FORMULA ==> T
(ES-MODELO-FORMULA 'NIL '(- Q))
ES-MODELO-FORMULA ==> T
(ES-MODELO-FORMULA 'NIL '(- P))
ES-MODELO-FORMULA ==> T
(ES-MODELO-FORMULA '(Q) '(P -> Q))
ES-MODELO-FORMULA ==> T
(ES-MODELO-FORMULA '(Q) '(- Q))
ES-MODELO-FORMULA ==> NIL
(ES-MODELO-FORMULA '(P) '(P -> Q))
ES-MODELO-FORMULA ==> NIL
(ES-MODELO-FORMULA '(P Q) '(P -> Q))
ES-MODELO-FORMULA ==> T
(ES-MODELO-FORMULA '(P Q) '(- Q))
ES-MODELO-FORMULA ==> NIL
T

> (es-consecuencia '(p) '(p & q))
(ES-MODELO-FORMULA 'NIL 'P)
ES-MODELO-FORMULA ==> NIL
(ES-MODELO-FORMULA '(P) 'P)
ES-MODELO-FORMULA ==> T
(ES-MODELO-FORMULA '(P) '(P & Q))
ES-MODELO-FORMULA ==> NIL
NIL
```

Relación entre consecuencia lógica, validez y consistencia

- **Relación entre consecuencia lógica, validez y consistencia: Las siguientes condiciones son equivalentes:**
  - $\{F_1, \dots, F_n\} \models G$
  - $\models F_1 \wedge \dots \wedge F_n \to G$
  - $\{F_1, \dots, F_n, \neg G\}$ **es inconsistente**

Problema de la B.C. de animales

```
(defun animales ()
  (es-consecuencia
   '(((tiene_pelos / da_leche) -> es_mamifero)
     ((es_mamifero & (tiene_pezugnas / rumia)) -> es_ungulado)
     ((es_ungulado & tiene_cuello_largo) -> es_jirafa)
     ((es_ungulado & tiene_rayas_negras) -> es_cebra)
     (tiene_pelos & (tiene_pezugnas & tiene_rayas_negras)))
   'es_cebra))

> (time (animales))
Real time: 19.922935 sec.
Run time: 19.87 sec.
Space: 194900 Bytes
GC: 1, GC time: 0.11 sec.
T
```

Referencias

- **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. y Nilsson, N.J. *Logical Foundations of Artificial Intelligence* (Morgan Kaufmann, 1987)**
  - **Cap. 2 "Propositional Logic"**
- **Lucas, P. y Gaag, L.v.d. *Principles of Expert Systems* (Addison–Wesley, 1991).**
  - **Cap. 2 "Logic and resolution"**
- **Russell, S. y Norvig, P. *Inteligencia artificial (un enfoque moderno)* (Prentice–Hall, 1996)**
  - **Cap. 6 "Agentes que razonan lógicamente"**
- **Thayse, A. y otros *Aproche logique de l'Intelligence Artificielle. (Vol 1: de la logique classique à la programmation logique).* (Dunod, 1988)**
  - **Cap. 1.1 "Calcul des propositions"**