Lógica informática (2018–19)

Tema 14: Formalización en Prolog de la lógica proposicional

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 14: Formalización en Prolog de la lógica proposicional

Tema 14: Formalización en Prolog de la lógica proposicional

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
```

Tema 14: Formalización en Prolog de la lógica proposicional

Tema 14: Formalización en Prolog de la lógica proposicional

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

- `función_de_verdad(+Op, +V1, +V2, -V)` se verifica si el valor de verdad de la conectiva binaria `Op` aplicada a los valores de verdad `V1` y `V2` es `V`.
- `función_de_verdad(+Op, +V1, -V)` se verifica si el valor de verdad de la conectiva unaria `Op` aplicada al valor de verdad `V1` es `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`. 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. 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 principales de una fórmula

- *I* es una 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`. 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)]]
```

```
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)] ;
I = [ (p, 1),  (q, 0),  (r, 1)] ;
I = [ (p, 1),  (q, 1),  (r, 0)] ;
I = [ (p, 1),  (q, 1),  (r, 1)] ;
No
```

```
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]
```

```
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,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
```

```
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`. Por ejemplo,

```
?- 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
```

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

Cálculo de los modelos principales 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
```

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

Cálculo de los modelos principales 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)]]
```

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

Comprobación de satisfacibilidad

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

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

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

Tema 14: Formalización en Prolog de la lógica proposicional

Cálculo de contramodelos de una fórmula

- `contramodelo_fórmula(?I,+F)` se verifica si I es un contramodelo principal de la fórmula `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
```

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

Comprobación de tautologías

- es_tautología(+F) se verifica si la fórmula F es una tautología. Por ejemplo,

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

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

- Definición alternativa:

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

Tema 14: Formalización en Prolog de la lógica proposicional

Cálculo de las interpretaciones principales de conjuntos

- `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)]]
```

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

Cálculo de las interpretaciones principales 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=> 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
```

```
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 S. Por ejemplo,

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

```
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. Por ejemplo,

```
?- 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
```

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

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`. 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
```

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

Cálculo de los modelos principales 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)]]
```

```
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 e `inconsistente(+S)`, si es inconsistente. Por ejemplo,

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

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

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

Tema 14: Formalización en Prolog de la lógica proposicional

Comprobación de consecuencia lógica

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

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

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

Comprobación de consecuencia lógica

- `contramodelo_consecuencia(+S,+F,?I)` se verifica si `I` es una interpretación principal de `S` ∪ `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
```

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

Comprobación de consecuencia lógica

- Definición alternativa de `es_consecuencia`

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

Bibliografía

- 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 (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)