Tema DA-10: Implementación en Prolog de la resolución

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo

Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Introducción a la resolución

- **Reducción de consecuencia lógica a inconsistencia de conjunto de cláusulas**
  - **Sea S un conjunto de fórmulas y F una fórmula. Son equivalentes:**
    - $S \models F$
    - $S \cup \{\neg F\}$ **es inconsistente**
    - **Cláusulas**$(S \cup \{\neg F\})$ **es inconsistente**

- **Decisión de la inconsistencia de un conjunto de cláusulas**
  - **El conjunto de cláusulas S es inconsistente syss la cláusula vacía es consecuencia de S.**

- **Reglas de inferencia:**
  - **Reglas habituales:**

Modus Ponens:	$\dfrac{p \rightarrow q, \quad p}{q}$	$\dfrac{\{\neg p, q\}, \quad \{p\}}{\{q\}}$
Modus Tollens:	$\dfrac{p \rightarrow q, \quad \neg q}{\neg p}$	$\dfrac{\{\neg p, q\}, \quad \{\neg q\}}{\{\neg p\}}$
Encadenamiento:	$\dfrac{p \rightarrow q, \quad q \rightarrow r}{p \rightarrow r}$	$\dfrac{\{\neg p, q\}, \quad \{\neg q, r\}}{\{\neg p, r\}}$

  - **Regla de resolución proposicional:**

$$\frac{\{p_1, \ldots, r, \ldots, p_m\}, \quad \{q_1, \ldots, \neg r, \ldots, q_n\}}{\{p_1, \ldots, p_m, q_1, , \ldots, q_n\}}$$

Regla de resolución proposicional

- **Complementarios**
  - El *complementario de un literal* L es
    $\overline{L} = \begin{cases} p, & \text{si } L = \neg p \\ \neg p, & \text{si } L = p \end{cases}$
  - `complementario(+L1,-L2)` se verifica si `L2` es el complementario del literal `L1`.
  - Def. de `complementario`:

```
complementario(-A, A) :- !.
complementario(A, -A).
```

Regla de resolución proposicional

- **Resolventes**
  - **La cláusula C es una *resolvente* de las cláusulas C_1 y C_2 si existe un literal L tal que $L \in C_1$, $\overline{L} \in C_2$ y $C = (C_1 - \{L\}) \cup (C_2 - \{\overline{L}\})$.**
  - **`resolvente(+C1,+C2,-C3)` se verifica si `C3` es una resolvente de las cláusulas `C1` y `C2`.**
  - **Ejemplos:**

```
?- resolvente([p,q],[-q,r],C).
C = [p, r] ;
No
?- resolvente([q, -p],[p, -q],C).
C = [p, -p] ;
C = [q, -q] ;
No
?- resolvente([q, -p], [p, q], C).
C = [q] ;
No
?- resolvente([q, -p], [q, r], C).
No
?- resolvente([p],[-p],C).
C = [] ;
No
```

  - **Def. de `resolvente`:**

```
resolvente(C1,C2,C) :-
   member(L1,C1),
   complementario(L1,L2),
   member(L2,C2),
   delete(C1, L1, C1P),
   delete(C2, L2, C2P),
   append(C1P, C2P, C3),
   sort(C3,C).
```

Demostraciones por resolución

- **Definiciones**
  - **Sea S un conjunto de cláusulas.**
  - **La sucesión $(C_1, \ldots, C_n)$ es una *demostración por resolución* de la cláusula C a partir de S si $C = C_n$ y para todo $i \in \{1, ..., n\}$ se verifica una de las siguientes condiciones:**
    - **$C_i \in S$;**
    - **existen $j, k < i$ tales que C_i es una resolvente de C_j y C_k**
  - **La cláusula C es *demostrable por resolución* a partir de S si existe una demostración por resolución de C a partir de S.**
  - **Una *refutación por resolución* de S es una demostración por resolución de la cláusula vacía a partir de S.**
  - **Se dice que S es *refutable por resolución* si existe una refutación por resolución a partir de S.**

Demostraciones por resolución

- **Definiciones**
  - **Sea S un conjunto de fórmulas.**
  - **Una *demostración por resolución* de F a partir de S es una refutación por resolución de Cláusulas($S \cup \{\neg F\}$).**
  - **La fórmula F es *demostrable por resolución* a partir de S si existe una demostración por resolución de F a partir de S.**
- **Ejemplo:**
  - **Demostración por resolución de $p \wedge q$ a partir de $\{p \vee q, p \leftrightarrow q\}$:**

```
1 {p,q}    Hipótesis
2 {-p,q}   Hipótesis
3 {p,-q}   Hipótesis
4 {-p,-q}  Hipótesis
5 {q}      Resolvente de 1 y 2
7 {-q}     Resolvente de 3 y 4
8 {}       Resolvente de 5 y 7
```

- **El cálculo por resolución es adecuado y completo.**

Búsqueda elemental de refutación

- **Reglas para determinar la inconsistencia del conjunto de cláusulas S:**
  - **Si S contiene a la cláusula vacía, entonces S es inconsistente.**
  - **Si S contiene dos cláusulas C_1, C_2 que tienen una resolvente que no pertenece a S, entonces S es inconsistente syss $S \cup \{C\}$ es inconsistente.**
  - **En otro caso, S es consistente.**
- **Dúsqueda elemental de refutación:**
  - **`refutacion(+S,-R)` se verifica si `R` es una refutación por resolución del conjunto de cláusulas `S`.**
  - **Ejemplo:**

```
?- refutacion([[-p,-q],[p,q],[-p,q],[-q,p]],R).
R = [[p,-q],[q,-p],[p,q],[-p,-q],
     [q,-q],[p,-p],[-p],[q],[p],[]]
?- refutacion([[p,q],[-p,q],[-q,p]],R).
No
```

  - **Ejemplo con traza de resolventes:**

```
?- refutacion([[-p,-q],[p,q],[-p,q],[-q,p]],R).
[q, -q] resolvente de [-p, -q] y [p, q]
[p, -p] resolvente de [-p, -q] y [p, q]
[-p] resolvente de [-p, -q] y [q, -p]
[q] resolvente de [-p] y [p, q]
[p] resolvente de [q] y [p, -q]
[] resolvente de [p] y [-p]

R = [[p,-q],[q,-p],[p,q],[-p,-q],
     [q,-q],[p,-p],[-p],[q],[p],[]]
```

Búsqueda elemental de refutación

- **Def. de `refutacion`:**

```
refutacion(S,R) :-
   maplist(sort,S,S1),
   refutacion_aux(S1,R).

refutacion_aux(S,R) :-
   member([],S), !,
   reverse(S,R).
refutacion_aux(S,R) :-
   member(C1,S),
   member(C2,S),
   resolvente(C1,C2,C),
   \+ member(C, S),
   % format('~N~w resolvente de ~w y ~w~n',
   %        [C,C1,C2]),
   refutacion_aux([C|S],R).
```

Búsqueda de refutación de OTTER

- **Ejemplo de búsqueda de refutación con OTTER**
  - **Entrada** `ejemplo.in`

```
list(sos).
-p | -q.
p | q.
-p | q.
-q | p.
end_of_list.

set(binary_res).    % Resolución binaria
set(very_verbose).  % Presentación detallada
```

  - **Ejecución** `otter <ejemplo.in >ejemplo.out`
  - **Fichero de salida** `ejemplo.out`

```
list(sos).
1 [] -p| -q.
2 [] p|q.
3 [] -p|q.
4 [] -q|p.
end_of_list.
```

Búsqueda de refutación de OTTER

```
=========== start of search ===========
given clause #1: (wt=2) 1 [] -p| -q.

given clause #2: (wt=2) 2 [] p|q.
  0 [binary,2.1,1.1] q| -q.
  0 [binary,2.2,1.2] p| -p.

given clause #3: (wt=2) 3 [] -p|q.
  0 [binary,3.1,2.1] q|q.
** KEPT (pick-wt=1): 5 [binary,3.1,2.1,factor_simp] q.
  0 [binary,3.2,1.2] -p| -p.
** KEPT (pick-wt=1): 6 [binary,3.2,1.2,factor_simp] -p.

given clause #4: (wt=1) 5 [...] q.
  0 [binary,5.1,1.2] -p.
  Subsumed by 6.

given clause #5: (wt=1) 6 [...] -p.
  0 [binary,6.1,2.1] q.
  Subsumed by 5.

given clause #6: (wt=2) 4 [] -q|p.
  0 [binary,4.1,5.1] p.
** KEPT (pick-wt=1): 7 [binary,4.1,5.1] p.

----> UNIT CONFLICT at   0.00 sec ----> 8 $F.
```

Búsqueda de refutación de OTTER

```
---------------- PROOF ----------------
1 [] -p| -q.
2 [] p|q.
3 [] -p|q.
4 [] -q|p.
5 [binary,3.1,2.1,factor_simp] q.
6 [binary,3.2,1.2,factor_simp] -p.
7 [binary,4.1,5.1] p.
8 [binary,7.1,6.1] $F.
------------ end of proof -------------
```

Búsqueda de refutación de OTTER

- **Procedimiento de búsqueda de pruebas**
  - **Mientras el soporte es no vacío y no se ha encontrado una refutación**
    1. **Seleccionar como cláusula actual la cláusula menos pesada del soporte (i.e. con el menor número de átomos).**
    2. **Mover la cláusula actual del soporte a usable.**
    3. **Calcular las resolventes de la cláusula actual con las cláusulas usables.**
    4. **Procesar cada una de las resolventes calculadas anteriormente.**
    5. **Añadir al soporte cada una de las cláusulas procesadas que supere el procesamiento.**

Búsqueda de refutación de OTTER

- **El procesamiento de cada una de resolventes consta de los siguientes pasos (los indicados con * son opcionales):**

  ***1. Escribir la resolvente.**

  ***2. Aplicar a la resolvente eliminación unitaria (i.e. elimina los literales de la resolvente tales que hay una cláusula unitaria complementaria en usable o en soporte).**

  3. Descartar la resolvente y salir si la resolvente es una tautología (i.e `A` y `-A`).

  4. Descartar la resolvente y salir si la resolvente es subsumida por alguna cláusula de usable o del soporte (subsunción hacia adelante).

  5. Añadir la resolvente al soporte.

  ***6. Escribir la resolvente retenida.**

  7. Si la resolvente tiene 0 literales, se ha encontrado una refutación.

  8. Si la resolvente tiene 1 literal, entonces buscar su complementaria (refutación) en usable y soporte.

  ***9. Si se ha encontrado una refutación, escribirla.**

  ***10. Descartar cada cláusula de usable o del soporte subsumida por la resolvente (subsunción hacia atrás).**

  El paso 10 no se da hasta que los pasos 1–9 se han aplicado a todas las resolventes.

Refutación de conjuntos de cláusulas

- **Procedimiento principal**
  - `refutacion(+U,+S)` **se verifica si se existe una refutación con usables** `U` **y soporte** `S` **(además, escribe la búsqueda y la prueba).**
  - **Ejemplos:**

```
?- refutacion([], [[p,q],[-p,q],[-q,p],[-p,-q]]).

Usable:

Soporte:
1 [] [p, q]
2 [] [-p, q]
3 [] [-q, p]
4 [] [-p, -q]

cláusula actual #1: 1 [] [p, q]

cláusula actual #2: 2 [] [-p, q]
  0 [2, 1] [q]
** RETENIDA: 5 [2, 1] [q]
5 subsume a 2
5 subsume a 1

cláusula actual #3: 5 [2, 1] [q]

cláusula actual #4: 3 [] [-q, p]
  0 [3, 5] [p]
** RETENIDA: 6 [3, 5] [p]
6 subsume a 3
```

Refutación de conjuntos de cláusulas

```
cláusula actual #5: 6 [3, 5] [p]

cláusula actual #6: 4 [] [-p, -q]
  0 [4, 6] [-q]
** RETENIDA: 7 [4, 6] [-q]

 ----> CONFLICTO UNITARIO 8 [7,5] []

---------------- DEMOSTRACION ----------------
1 [] [p, q]
2 [] [-p, q]
3 [] [-q, p]
4 [] [-p, -q]
5 [2, 1] [q]
6 [3, 5] [p]
7 [4, 6] [-q]
8 [7, 5] []
----------- fin de la demostración -----------

Yes
```

Refutación de conjuntos de cláusulas

```
?- refutacion([],[[-p,q],[p]]).

Usable:

Soporte:
1 [] [-p, q]
2 [] [p]

cláusula actual #1: 2 [] [p]

cláusula actual #2: 1 [] [-p, q]

  0 [1, 2] [q]
** RETENIDA: 3 [1, 2] [q]
3 subsume a 1

cláusula actual #3: 3 [1, 2] [q]

No
```

- **Def. de** `refutacion/2`**:**

```
refutacion(Usable,Soporte) :-
   refutacion(Usable,Soporte,Ref),
   escribe_prueba(Ref).
```

Refutación de conjuntos de cláusulas

• Búsqueda de la refutación

- `refutacion(+Usable,+Soporte,-Ref)` **se verifica si** `Ref` **es una refutación por resolución del conjunto de cláusulas de** `Usable` **y** `Soporte` **(además, escribe la búsqueda).**

```
refutacion(Usable,Soporte,Ref) :-
   inicia,
   format('~N~nUsable:~n',[]),
   anotado(Usable,Usable1),
   format('~N~nSoporte:~n',[]),
   anotado(Soporte,Soporte1),
   ordenada_por_peso(Soporte1,Soporte2),
   refutacion_anotada(Usable1,Soporte2,[],Ref).
```

• Iniciación de las variables globales

- `inicia` **asigna valores a las variables globales: a** `n_clausulas_analizadas` **(que cuenta las cláusulas analizadas) le asigna el valor 0 y a** `n_clausulas_retenidas` **(que cuenta las cláusulas retenidas) le asigna el valor 0.**

```
inicia :-
   asigna_global(n_clausulas_retenidas,0),
   asigna_global(n_clausulas_analizadas,0).
```

- `asigna_global(+A,+T)` **asigna al átomo** `A` **(de forma global) el valor del término** `T`**.**

```
asigna_global(A,T) :-
   flag(A,_,T).
```

Refutación de conjuntos de cláusulas

• Anotación de conjuntos de cláusulas

- anotado(+S1,-S2) **se verifica si** S2 **es el conjunto de cláusulas anotadas correspondiente al conjunto de cláusulas** S1 **(además, escribe las cláusulas anotadas). Por ejemplo,**

```
?- asigna_global(n_clausulas_retenidas,0),
   anotado([[-p, q],[q,p]],S).
1 [] [-p, q]
2 [] [q, p]
S = [1*[]*[q, -p], 2*[]*[p, q]]
```

- **Def. de** anotado**:**

```
anotado([],[]).
anotado([C|S1],[N*[]*C|S2]) :-
   incrementa_global(n_clausulas_retenidas,N),
   format('~N~w [] ~w~n',[N,C]),
   anotado(S1,S2).
```

- incrementa_global(+A) **incrementa en 1 el valor grobal del átomo** A**.**

```
incrementa_global(A,N) :-
   valor_global(A,M),
   N is M+1,
   asigna_global(A,N).
```

- valor_global(+A,-T) **se verifica si** T **es el valor global del átomo** A**.**

```
valor_global(A,T) :-
   var(T),
   flag(A,T,T).
```

Refutación de conjuntos de cláusulas

• Ordenación de cláusulas por peso

- `ordenada_por_peso(+S1,-S2)` **se verifica si** `S2` **es el conjunto de las cláusulas anotadas del conjunto** `S1` **ordenadas por peso. Por ejemplo,**

```
?- ordenada_por_peso([3*[]*[p,q],
                      5*[1,2]*[r],
                      7*[4,5]*[]],
                     S).
S = [7*[4, 5]*[], 5*[1, 2]*[r], 3*[]*[p, q]]
```

- **Def. de** `ordenada_por_peso`**:**

```
ordenada_por_peso(S1,S2) :-
   findall(L1-N1*P1*C1,
           (member(N1*P1*C1,S1),
            length(C1,L1)),
           S1a),
   keysort(S1a,S2a),
   findall(CA2,member(_-CA2,S2a),S2).
```

Refutación de conjuntos de cláusulas

- **Refutación de conjuntos de cláusulas anotadas y con soporte ordenada por peso**
  - `refutacion_anotada(+Usable,+Soporte,+Subsumidas,-Ref)` **se verifica si** `Ref` **es una refutación por resolución del conjunto de cláusulas anotadas de** `Usable` **y** `Soporte` **(además, escribe la búsqueda). Por ejemplo,**

```
?- asigna_global(n_clausulas_retenidas,4),
   refutacion_anotada([1*[]*[-r]],
                      [2*[]*[p],
                       3*[]*[-p,q],
                       4*[]*[-q, r]],
                      [],
                      R).

...
R = [1*[]*[-r],
     2*[]*[p],
     3*[]*[-p, q],
     4*[]*[-q, r],
     5*[3, 2]*[q],
     6*[4, 5]*[r],
     7*[6, 1]*[]]
```

Refutación de conjuntos de cláusulas

- **Def. de** `refutacion_anotada`**:**

```
refutacion_anotada(Usable,Soporte,Sub,Ref) :-
   Soporte = [_*_*[]|_],
   findall(C,(member(C,Usable);
              member(C,Soporte);
              member(C,Sub)),
           S),
   prueba(S,Ref), !.

refutacion_anotada(Usable,[C|Soporte],Sub,Ref) :-
   escribe_actual(C),
   resolventes(C,[C|Usable],S1),
   procesa(Usable,Soporte,S1,S2),
   ( memberchk(_*_*[],S2) ->
     append(S2,Soporte,Soporte1),
     refutacion_anotada([C|Usable],Soporte1,Sub,Ref)
   ; % \+ memberchk(_*_*[],S2) ->
     elimina_subsumidas(S2,[C|Usable],Sub,Usable1,Sub1),
     elimina_subsumidas(S2,Soporte,Sub1,Soporte1,Sub2),
     append(Soporte1,S2,Soporte2),
     ordenada_por_peso(Soporte2,Soporte3),
     refutacion_anotada(Usable1,Soporte3,Sub2,Ref)).
```

Refutación de conjuntos de cláusulas

• Construcción de la prueba

- `prueba(+S,-Dem)` **se verifica si** `Dem` **es la prueba contenida en el conjunto de cláusulas anotadas** `S`**. Por ejemplo,**

```
?- prueba([3*[]*[r, -q],
           6*[2, 4]*[p],
           5*[2, 1, 4]*[q],
           4*[]*[-r],
           8*[7, 4]*[],
           7*[3, 5]*[r],
           2*[]*[p, r],
           1*[]*[q, -p]], Dem).

Dem = [1*[]*[q, -p],
       2*[]*[p, r],
       3*[]*[r, -q],
       4*[]*[-r],
       5*[2, 1, 4]*[q],
       7*[3, 5]*[r],
       8*[7, 4]*[]]
```

- **Def. de** `prueba`**:**

```
prueba(S,Dem) :-
   member(N*H*[],S),
   setof(CA,antecesor(S,N*H*[],CA),P1),
   append(P1,[N*H*[]],Dem).
```

Refutación de conjuntos de cláusulas

- **antecesor(+S,+CA1,-CA2) se verifica si CA2 es un antecesor de la cláusula anotada CA1 en el conjunto de cláusulas anotadas S. Por ejemplo,**

```
?- antecesor([3*[]*[r, -q],
              6*[2, 4]*[p],
              5*[2, 1, 4]*[q],
              4*[]*[-r],
              8*[7, 4]*[],
              7*[3, 5]*[r],
              2*[]*[p, r],
              1*[]*[q, -p]],
             8*[7, 4]*[],
             C).
C = 7*[3, 5]*[r] ;
C = 4*[]*[-r] ;
C = 3*[]*[r, -q] ;
C = 5*[2, 1, 4]*[q] ;
C = 2*[]*[p, r] ;
C = 1*[]*[q, -p] ;
C = 4*[]*[-r] ;
No
```

- **Def. de antecesor:**

```
antecesor(S,CA1,CA2) :-
   padre(S,CA1,CA2).
antecesor(S,CA1,CA2) :-
   padre(S,CA1,CA3),
   antecesor(S,CA3,CA2).
```

Refutación de conjuntos de cláusulas

- **padre(+S,+CA1,-CA2) se verifica si CA2 es un padre en S de la cláusula anotada CA1. Por ejemplo,**

```
?- padre([3*[]*[r, -q],
          6*[2, 4]*[p],
          5*[2, 1, 4]*[q],
          4*[]*[-r],
          8*[7, 4]*[],
          7*[3, 5]*[r],
          2*[]*[p, r],
          1*[]*[q, -p]],
         8*[7, 4]*[],
         C).
C = 7*[3, 5]*[r] ;
C = 4*[]*[-r] ;
No
```

- **Def. de padre:**

```
padre(S,CA1,CA2) :-
   CA1 = _*H*_,
   member(N,H),
   CA2 = N*_*_,
   member(CA2,S).
```

Refutación de conjuntos de cláusulas

- **Escritura de la cláusula actual**
  - `escribe_actual(N*P*C)` **incrementa el número de cláusulas analizadas y escribe la cláusula anotada** `N*P*C`**. Por ejemplo,**

```
?- asigna_global(n_clausulas_analizadas,5),
   escribe_actual(2*[]*[p]).

cláusula actual #6: 2 [] [p]
Yes
```

  - **Def. de** `escribe_actual`**:**

```
escribe_actual(N*P*C) :-
   incrementa_global(n_clausulas_analizadas,M),
   format('~N~ncláusula actual #~w: ~w ~w ~w~n',
          [M,N,P,C]).
```

- **Cálculo de las resolventes anotadas**
  - `resolventes(+C,+S1,-S2)` **se verifica si** `S2` **es el conjunto obtenido resolviendo la cláusula anotada** `C` **con las cláusulas anotadas del conjunto** `S1`**. Por ejemplo,**

```
?- resolventes(1*[]*[-p,q],[2*[]*[p,q],
                3*[]*[p,r],4*[]*[-q,s]],
                S).
S = [_*[1,2]*[q], _*[1,3]*[q,r], _*[1,4]*[-p,s]]
```

  - **Def. de** `resolventes`**:**

```
resolventes(C,S1,S2) :-
   findall(C2,(member(C1,S1),resolvente(C,C1,C2)),S2).
```

Refutación de conjuntos de cláusulas

- `resolvente(+C1,+C2,-C3)` **se verifica si** `C3` **es una resolvente de las cláusulas anotadas** `C1` **y** `C2`**.**

```
resolvente(N1*_*C1,N2*_*C2,_*[N1,N2]*C) :-
   member(L1,C1),
   complementario(L1,L2),
   member(L2,C2),
   delete(C1, L1, C1P),
   delete(C2, L2, C2P),
   append(C1P, C2P, C3),
   list_to_set(C3,C).
```

- `complementario(+L1,-L2)` **se verifica si** `L2` **es el complementario del literal** `L1`**.**

```
complementario(-A, A) :- !.
complementario(A, -A).
```

Procesamiento de las resolventes

• Procesamiento de las resolventes

- `procesa(+Usable,+Soporte,+Resolventes,-Retenidas)` **se verifica si** `Retenidas` **son las cláusulas de** `Resolventes` **retenidas al procesarlas con** `Usables` **y** `Soporte` **(además escribe mensajes del procesamiento).**

- **Ejemplo 1**

```
?- asigna_global(n_clausulas_retenidas,7),
   procesa([1*[]*[q]],
           [3*[]*[p]],
           [N1*[]*[q,r],
            N2*[]*[p,r],
            N3*[]*[s,-s],
            N4*[]*[s]],
           R).
  0 [] [q, r]
  Subsumida por 1.
  0 [] [p, r]
  Subsumida por 3.
  0 [] [s, -s]
  0 [] [s]
** RETENIDA: 8 [] [s]

R = [8*[]*[s]]
```

Procesamiento de las resolventes

- **Ejemplo 2**

```
?- asigna_global(n_clausulas_retenidas,7),
   procesa([1*[]*[q]], [], [N*[]*[-q]], R).
  0 [] [-q]
** RETENIDA: 8 [] [-q]

 ----> CONFLICTO UNITARIO 9 [8,1] []

R = [9*[8, 1]*[], 8*[]*[-q]]
```

- **Ejemplo 3**

```
?- asigna_global(n_clausulas_retenidas,7),
   procesa([1*[]*[q]], [], [N*[]*[-q,s]], R).
  0 [] [-q, s]
** RETENIDA: 8 [1] [s]

R = [8*[1]*[s]]
```

- **Ejemplo 4**

```
?- asigna_global(n_clausulas_retenidas,7),
   procesa([1*[]*[q]], [], [N*[]*[]], R).
  0 [] []
** RETENIDA: 8 [] []

 -----> CLAUSULA VACIA: 8 [] []

R = [8*[]*[]]
```

Procesamiento de las resolventes

- **Ejemplo 5**

```
?- asigna_global(n_clausulas_retenidas,7),
   procesa([1*[]*[q]],
           [2*[]*[s]],
           [N1*[]*[-q,r],
            N2*[]*[q,s],
            N3*[]*[-r,-s,t]],
           R).
  0 [] [-q, r]
** RETENIDA: 8 [1] [r]
  0 [] [q, s]
  Subsumida por 1.
  0 [] [-r, -s, t]
** RETENIDA: 9 [8, 2] [t]

R = [8*[1]*[r], 9*[8, 2]*[t]]
```

Procesamiento de las resolventes

- **Def. de** `procesa`**:**

```
procesa(_,_,[],[]).
procesa(Usable,Soporte,[C|S1],S2) :-
   escribe_resolvente(C),
   ( (subsumida(C,Usable) ;
      subsumida(C,Soporte);
      es_clausula_tautologica(C)) ->
     procesa(Usable,Soporte,S1,S2)
   ; % C es retenida ->
     numera(C,C1),
     ( conflicto_unitario(C1,Usable,Soporte,C2) ->
       S2 = [C2,C1]
     ; % C1 no es unitaria con complementaria en
       % Usable o Soporte ->
       eliminacion_unitaria(Usable,Soporte,C1,C2),
       escribe_retenida(C2),
       ( C2 = N*H*[] ->
         format('~N~n -----> CLAUSULA VACIA: ~w ~w []',
                [N,H]),
         S2 = [C2]
       ; % C2 no es la cláusula vacía ->
         procesa(Usable,[C2|Soporte],S1,S3),
         S2 = [C2|S3]))).
```

Procesamiento de las resolventes

- **Escritura de la resolvente procesada**
  - `escribe_resolvente(+C)` **escribe la resolvente procesada** `C`**. Por ejemplo,**

```
?- escribe_resolvente(N*[1,3]*[p,-q]).
  0 [1, 3] [p, -q]
N = _G334
Yes
```

  - **Def. de** `escribe_resolvente`**:**

```
escribe_resolvente(C) :-
   C = _*P*C1,
   format('~N  0 ~w ~w~n',[P,C1]).
```

Procesamiento de las resolventes

- **Eliminación de cláusulas subsumidas**
  - `subsumida(+C,+S)` **se verifica si existe una cláusula del conjunto** `S` **que subsume a la cláusula** `C` **(en cuyo caso escribe un mensaje indicativo). Por ejemplo,**

```
?- subsumida(_N*[3, 2]*[r, p], [2*[]*[-q, p], 1*[]*[p]]).
  Subsumida por 1.
Yes
?- subsumida(_N*[3, 2]*[r, s], [2*[]*[-q, p], 1*[]*[p]]).
No
```

  - **Def. de** `subsumida`**:**

```
subsumida(C,S) :-
   member(D,S),
   subsume(D,C),
   D = N*_*_,
   format('~N  Subsumida por ~w.~n',[N]).
```

  - `subsume(+C,+D)` **se verifica si la cláusula anotada** `C` **subsume a la** `D`**.**

```
subsume(_*_*C1,_*_*C2) :-
   subset(C1,C2).
```

- **Eliminación de cláusulas tautológicas**
  - `es_clausula_tautologica(+C)` **se verifica si la cláusula anotada C es una tautología (i.e. contiene un literal y su complementario).**

```
es_clausula_tautologica(_*_*C) :-
   member(-A,C),
   memberchk(A,C).
```

Procesamiento de las resolventes

- **Numeración de cláusulas retenidas**
  - `numera(C1,C2)` **se verifica si incrementa el número de cláusulas retenidas y** `C2` **se obtiene numerando la cláusula retenida** `C1`**. Por ejemplo,**

```
?- asigna_global(n_clausulas_retenidas,7),
   numera(_*[2, 1]*[q], C).
C = 8*[2, 1]*[q]
```

  - **Def. de** `numera`**:**

```
numera(_*H*C,N*H*C) :-
   incrementa_global(n_clausulas_retenidas,N).
```

Procesamiento de las resolventes

• Procesamiento de cláusulas unitarias

- `conflicto_unitario(+C1,+U,+S,-C2)` **se verifica si** `C1` **es una cláusula unitaria cuya complementaria** `C` **pertenece a** `U` **ó** `S` **y** `C2` **es la resolvente de** `C1` **y** `C` **(además, escribe un mensaje indicativo). Por ejemplo,**

```
?- conflicto_unitario(7*[4,6]*[-q],
                      [6*[3,5]*[p], 5*[2,1]*[q]],
                      [],
                      C).
** RETENIDA: 7 [4, 6] [-q]
 ----> CONFLICTO UNITARIO 8 [7,5] []
C = 8*[7, 5]*[]
Yes
```

- **Def. de** `conflicto_unitario`**:**

```
conflicto_unitario(N1*H1*[L1],Usa,Sop,M*[N1,N2]*[]) :-
   complementario(L1,L2),
   ( memberchk(N2*_H*[L2],Usa)
   ; memberchk(N2*_H*[L2],Sop) ),
   escribe_retenida(N1*H1*[L1]),
   M is N1+1,
   format('~N~n ----> CONFLICTO UNITARIO ~w [~w,~w] []~n',
          [M,N1,N2]).
```

• Escritura de cláusulas retenidas

- `escribe_retenida(+C)` **escribe la cláusula** `C`**.**

```
escribe_retenida(N*H*C) :-
   format('~N** RETENIDA: ~w ~w ~w',[N,H,C]).
```

Procesamiento de las resolventes

- **Eliminación unitaria**
  - `eliminacion_unitaria(+Usable,+Soporte,+C1,-C2)` **se verifica si** `C2` **es el resultado de eliminar los literales de la cláusula anotada** `C1` **tales que la cláusula unitaria complementaria está en usable o el soporte y actualizar la historia de** `C1` **con los números de las cláusulas unitarias usadas en la eliminación. Por ejemplo,**

```
?- eliminacion_unitaria([1*[]*[q,-p],
                         2*[]*[p]],
                        [4*[]*[-s]],
                         _N*[3, 1]*[-p,r,s],
                        C).
C = _G621*[3, 1, 2, 4]*[r]
?- eliminacion_unitaria([1*[]*[q,-p],
                         2*[]*[t]],
                        [4*[]*[-s,q]],
                         _N*[3, 1]*[-p,r,s],
                        C).
C = _G1188*[3, 1]*[-p, r, s]
```

  - **Def. de** `eliminacion_unitaria`**:**

```
eliminacion_unitaria(_U,_S,N*H*[],N*H*[]).
eliminacion_unitaria(U,S,N*H*[L|C],N*H1*C1) :-
   complementario(L,L1),
   (member(M*_*[L1],U) ; member(M*_*[L1],S)), !,
   append(H,[M],H2),
   eliminacion_unitaria(U,S,N*H2*C,N*H1*C1).
eliminacion_unitaria(U,S,N*H*[L|C],N*H1*[L|C1]) :-
   eliminacion_unitaria(U,S,N*H*C,N*H1*C1).
```

Subsunción hacia atrás

• Subsunción hacia atrás

- `elimina_subsumidas(+S,+S1,+T1,-S2,-T2)` **se verifica si** `S2` **es el conjunto de cláusulas de** `S1` **no subsumidas por cláusulas de** `S` **y** `T2` **es la unión de** `T1` **y las cláusulas de** `S1` **subsumidas por cláusulas de** `S` **(además, escribe el mensaje correspondiente).**

- **Ejemplo 1**

```
?- elimina_subsumidas([5*[2,1]*[q]],
                      [2*[]*[-p,q], 1*[]*[p,q]],
                      [],
                      C,
                      T).
5 subsume a 2
5 subsume a 1
C = []
T = [2*[]*[-p, q], 1*[]*[p, q]]
```

- **Ejemplo 2**

```
?- elimina_subsumidas([6*[3,5]*[p]],
                      [3*[]*[-q,p], 5*[2,1]*[q]],
                      [2*[]*[-p,q], 1*[]*[p,q]],
                      C,
                      T).
6 subsume a 3

C = [5*[2, 1]*[q]]
T = [3*[]*[-q, p], 2*[]*[-p, q], 1*[]*[p, q]]

Yes
```

Subsunción hacia atrás

- **Ejemplo 3**

```
?- elimina_subsumidas([5*[2,1]*[q,r]],
                      [2*[]*[-p,q], 1*[]*[p,q]],
                      [],
                      C,
                      T).

C = [2*[]*[-p, q], 1*[]*[p, q]]
T = []
```

- **Def. de** `elimina_subsumidas`**:**

```
elimina_subsumidas(_,[],T,[],T).
elimina_subsumidas(S,[N1*H1*C1|S1],T1,S2,[N1*H1*C1|T2]) :-
   member(N*_*C,S),
   subsume(N*_*C,N1*_*C1),!,
   format('~N~w subsume a ~w~n',[N,N1]),
   elimina_subsumidas(S,S1,T1,S2,T2).
elimina_subsumidas(S,[C1|S1],T1,[C1|S2],T2) :-
   elimina_subsumidas(S,S1,T1,S2,T2).
```

Escritura de la prueba

• Escritura de la prueba

- `escribe_prueba(+P)` **escribe la prueba** `P`**. Por ejemplo,**

```
?- escribe_prueba([1*[]*[p, q],
                   2*[]*[-p, q],
                   3*[]*[-q, p],
                   4*[]*[-p, -q],
                   5*[2, 1]*[q],
                   6*[3, 5]*[p],
                   7*[4, 6]*[-q],
                   8*[7, 5]*[]]).

---------------- DEMOSTRACION ----------------
1 [] [p, q]
2 [] [-p, q]
3 [] [-q, p]
4 [] [-p, -q]
5 [2, 1] [q]
6 [3, 5] [p]
7 [4, 6] [-q]
8 [7, 5] []
----------- fin de la demostración -----------

Yes
```

- **Def. de** `escribe_prueba`**:**

```
escribe_prueba(Dem) :-
   format('~N~n------- DEMOSTRACION -------~n',[]),
   checklist(escribe_linea,Dem),
   format('~N--- fin de la demostración ---~n',[]).

escribe_linea(N*H*C) :-
   format('~N~w ~w ~w',[N,H,C]).
```

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. 4.3: Resolución
- Ben–Ari, M. *Mathematical Logic for Computer Science (2nd ed.)* (Springer, 2001)
- Chang, C.–L. y Lee, R.C.–T. *Symbolic Logic and Mechanical Theorem Proving* (Academic Press, 1973)
- Fitting, M. *First-Order Logic and Automated Theorem Proving (2nd ed.)* (Springer, 1995)
- Nerode,A. y Shore, R.A. *Logic for Applications* (Springer, 1997)
- Gallier, J.H. *Logic for Computer Science (Foundations of Automatic Theorem Proving)* (Harper & Row, 1986)
- Schöning, U. *Logic for Computer Scientists* (Birkäuser, 1989)