Politecnico di Milano

ECOLE DES MINES DE NANTES

Reso-conto per il progetto

“Le soriti di Lewis Caroll”

Studenti :	Anne-Sophie Bénard (E02351) Gaspard Boin, Pascal Brunot (E02350)	 *Facoltà V* *Laurea d’ingegneria del software.*
E-mail :	pbrunot@eleve.emn.Fr	*Anno 2001 - 2002*

06/04/2003

Sommario

I. Presentazione delle soriti

Sono degli enigmi logici inventati da Lewis Caroll (matematico e scrittore inglese).
Un esempio (semplice!) è :

- "qualcuno che può uccidere un coccodrillo non è disprezzato"
- "la gente illogica è disprezzata"
- "i bambini sono illogici"

Ciascuna delle singole frasi è chiamata “premessa”. Ciascuna premessa deve essere utilizzata una ed una sola volta per arrivare alla conclusione (eccetto se definisce un’equivalenza).

La sua soluzione è che i bambini non possono uccidere i coccodrilli. Esaminiamo il ragionamento :

- visto che i bambini sono genti illogici (informazione della premessa 3), sono disprezzati (premessa 2 + *modus pollens*).
- Il *modus tollens* applicato alla premessa 1 dà : “la gente disprezzata non può uccidere un coccodrillo”.
- Basta allora usare un *sillogismo* per dedurre che i bambini non possono uccidere un coccodrillo.

II. Una procedura risolutiva

Il nostro scopo era di entrare le clausole di una sorite in linguaggio naturale :

```
“i bambini sono illogici”
“la gente illogica è disprezzata”
“nessuno è disprezzato se può uccidere un coccodrillo”
```

e di ottenere la soluzione della sorite anche in linguaggio naturale :

```
”i bambini non possono uccidere un coccodrillo”
```

Per realizzare questo, abbiamo proceduto in due passi :

- 1° passo : ci siamo interessati alla risoluzione automatica delle soriti, partendo da clausole messe sotto forma normale.
  A partire di delle clausole :
  ```
  [non(bambino(X)),illogico(X)]
  [non(coccodrillo(Z)),non(disprezzare(Z))]
  [non(illogico(Y)),disprezzare(Y)]
  ```
  il programma deve ottenere la soluzione :
  ```
  [non(bambino(X)),non(coccodrillo(X))]
  ```

- 2° passo : abbiamo cercato di ottenere queste clausole messe sotto forma normale a partire di frase in linguaggio naturale.
  A partire delle clausole in linguaggio naturale :
  ```
  “i bambini sono illogici”
  “la gente illogica è disprezzata”
  “nessuno è disprezzato se può uccidere un coccodrillo”
  ```
  il programma deve metterle sotto forma normale :
  ```
  [non(bambino(X)),illogico(X)]
  [non(coccodrillo(Z)),non(disprezzare(Z))]
  [non(illogico(Y)),disprezzare(Y)]
  ```

III.Risoluzione automatica

Prendiamo un'altra sorite, composta delle seguente clausole :
- "gli esercizi che vengono risoluti senza brontolare sono quei che capisco"
- "quest'esercizio non è disposto regolarmente"
- "nessun esercizio che è facile da' l'emicrania"
- "gli esercizi che non sono disposti regolarmente non sono tra quei che capisco"
- "gli esercizi sono risoluti senza brontolare tranne se danno l'emicrania"

? Può essere modellizzata dalle formule logiche seguente :

Qualsiasi x,esercizio : non(brontolare(x)) ✍ capito (x)
Esiste x,esercizio talché : non(disposto regolarmente(x))
Qualsiasi x,esercizio : facile(x) ✍ non(emicrania(x))
Qualsiasi x,esercizio : non(disposto regolarmente(x)) ✍ non(capito(x))
Qualsiasi x,esercizio : emicrania(x) ✍ brontolare(x)

? Tutte queste formule possono essere messe sotto forma normale con i quantificatori in testa (è un teorema del calcolo dei predicati) :

? x : brontolare(x) ? capito (x)
? x : non(disposto regolarmente(x))
? x : non(facile(x)) ? non(emicrania(x)
? x : disposto regolarmente(x) ? non(capito(x))
? x : non(emicrania(x)) ? brontolare(x)

? Ripresentiamo ciascuna premessa da una lista $[x_1,x_2,\ldots x_n]$ corrispondente alla formula logica $\bigvee_{i=1}^{n} a_i$. (ai) è una proprietà o la negazione di una proprietà.

? La sorite in generale è logicamente rappresentata da: $\bigwedge_{i=1}^{numero\ Premesse} \left(\bigvee_{j=1}^{n(i)} a_{ij} \right)$

? Nell'insieme delle liste cerchiamo un elemento X, talché apparisca non(X) in un'altra lista. Il sillogismo è la riunione delle due liste meno {X} e meno {non(X)}

E.g.:
Lista 1: [bambino(X),non(illogico(X))] cioè bambino ✍ illogico
Lista 2: [illogico(Z),non(disprezzato(Z))] cioè illogico ✍ disprezzato
SIL : [bambino(X),non(disprezzato(X))] cioè bambino ✍ disprezzato

Si nota l'importanza dell'unificazione Z=X, fatta automaticamente da Prolog, nell'ottenere il risultato.

IV. Traduzione linguaggio naturale

a) Traduzione verso forma normale

Il principio di base è di riconoscere nelle frasi dei pattern. Per esempio in “i bambini sono illogici” dobbiamo riconoscere il pattern “i A sono B” che si modelizza in A✍B, cioè sotto forma normale: non(A),B. Ma prima di riconoscere questi pattern, bisogna semplificare le espressioni a finche il programma non estima che “disprezzato” e “disprezzata” non è la stessa parole.

- Prima trasformiamo tutte le frasi in liste di parole :
  `“i bambini sono illogici”`
  diventa
  `[i,bambini,sono,illogici]`

- Mettiamo tutte le parole al singolare ; cosi otteniamo :
  `[il, bambino,è,illogico]`

- Poi togliamo le parole non significative (come gli articoli e le preposizioni) :
  `[bambino,è,illogico]`

- Finalmente, riconosciamo il pattern, in questo caso “i A sono B” e mettiamo la clausola sotto forma normale :
  `[non(bambino),illogico]`

b) Traduzione verso linguaggio naturale

- La conclusion può essere da due forme :
  - A ou B dove A e B sono un atomo o una proprietà, eventualmente negati.
  - B(A) dove A è un atomo e B una proprietà eventualmente negata.

- Riconosciamo gli diversi pattern possibile (sono pochi) e per ciascuno pattern abbiamo una catena generica dove viene concatenate i nomi di A o B.

e.g. : [non(bambino),illogico]
pattern : Non(Costante) o Proprietà
traduzione : Costante è Proprietà
risultato : bambino è illogico.

c) Pattern riconosciuti

```
Nessun X che [non] è B [non] è C
Gli X che [non] sono B [non] sono C
I X [non] sono B salvo se sono C
I X [non] sono B almeno che siano C
Tutti i A [non] sono B
I [non] A [non] sono B
Soli i A sono B
Nessun [non] è A quando [non] è C
A [non] è B
```

V. Scelte d'implementazione

? Visto che Prolog è un linguaggio che prova a risolvere le clausole in algoritmo *deep-first*, c'è una **dipendenza** all'ordine dell'insieme di clausole.
Come essere sicuro che una volta arrivato in un punto dove non ci sono più ragionamenti possibili, siamo alla conclusione ?
Utilizziamo la proprietà delle sorite che vuole che ogni clausola viene utilizzata una ed una sola volta. Ogni volta che si utilizza due clausole per ottenere una conclusione, aggiungiamo uno ad un contatore, togliamo le due clausole fonte, e proseguiamo.
Esploriamo tutte le possibilità (si nota che Prolog lo fa da sé: quando abbiamo una soluzione, provochiamo un *fail* finché continua a esplorare l'albero delle possibile soluzioni).
La soluzione con la valutazione maggiore è ritenuta "soluzione" dal nostro algoritmo.

? Si gestisce inoltre tre tipi di atomi : le costanti (cioè un valore che può assumere una variabile libera X), le proprietà (funzioni tipo Illogico(X)) e gli universi (parole che restringono il dominio per quale è valido una proprietà).

1. Se si dice "A è B", viene modellizzato come B(A) dove A è una constante.

2. Se si dice "tutti gli A sono B", viene modellizzato come A(X)=>B(X)

3. Se si dice "tutti gli A che sono B sono C", si definisce l'universo A e la proprietà B(X)=>C(X). Non è gestito il fatto che X deve appartenere a A perché non è utile per risolvere le sorite in esempio e che richiede uno sforzo modellistico grande.
L'universo è utilizzato per presentare il risultato in modo leggibile e acconsentire alla riconoscenza di pattern di funzionare anche con frase complesse.

VI. Bibliografia

Lewis Carroll, *Logique sans peine*, Hermann, 1966.

Linguaggio Prolog :
"*Programmation en logique – FI-3 2002*", Narenda Jussien, Ecole des Mines de Nantes.

Cenni d'algebra logica :
"*Logic algebra*", http://www.geocities.com/Athens/6208/logic1.html

Giochi di Lewis Caroll :

"*Lewis Caroll's game of logic*" : http://www.cut-the-knot.com/LewisCaroll/[1]
"*Lewis Carroll's puzzles*", http://art3idea.ce.psu.edu/boundaries/math/carroll.html

[1] E' presentato un altro modo di risolvere le sorite, contando per ciascuna proprietà le apparizioni a destra o a sinistra.

VII. Codice sorgente con esempi

Vedere al fine per esempi risoluti con il programma. Il software utilizzato è *Sicstus Prolog* (gratuito per uso academico, scaricabile su www.sicstus.com).

```
% ************************************************************************
% *                            Primo passo :                             *
% *                         RESOLUZIONE SIMBOLICA                        *
% ************************************************************************


% ************************ 1.1 PREDICATI AUSILI ************************
% Predicati ausili di cui ci serviremo spesso.

% append/3 : append(Lista1,Lista2,Risultato) concatena Lista1 con Lista2 e mette il
% risultato in R.
append([],X,X).
append([X|Y],Z,[X|W]):-
    append(Y,Z,W).

% member/2 : member(X,Lista) testa l'appartenenza di X a Lista.
member(X,[X|_]).
member(X,[_|Ys]):-
        member(X,Ys).

% cancella/3 : cancella(E,Lista,Risultato) cancella tutte le occorrenze di E in
% Lista e mette il risultato in R.

% 1° caso: termine la recursione
cancella(_,[],[]).

% 2° caso: se E è il primo elemento della lista esaminata
cancella(E,[E|R1],R2):-
       !,
       cancella(E,R1,R2).

% 3° caso: se E non è il primo elemento della lista esaminata
cancella(E,[X|R1],[X|R2]) :-
        !,
        cancella(E,R1,R2).

% not/1 : not(P) ritorna "yes" si P è false, "no" altrimenti.
not(P):-
       P,
       !,
       fail.
not(_).


% ****************** 1.2 RISOLUZIONE TRA DUE CLAUSOLE ******************

% risoluzione/4 : risoluzione(Clausola1,Clausola2,Risultato,Value) risolve le due
% clausole e mette il risultato in Risultato e il numero di passi necessari per
% ottenere il risultato in Value.
% Esempi di utilizzazione :
% risoluzione([non(bambino(X)),illogico(X)],[non(illogico(Y)),disprezzare(Y)],R,
% Value).
% --> R = [non(bambino(X)),disprezzare(X)],
%     Y = X,
%     Value = 1 ? ;
%     no
% risoluzione([a(X), b(X)], [c(X), d(X)],R,Value).
% --> no
% risoluzione([a(X), b(X)], [non(b(Y)), non(a(Y))],R,Value).
% --> R = [b(X),non(b(X))],
%     Y = X,
%     Value = 1 ? ;
%     R = [a(X),non(a(X))],
%     Y = X,
%     Value = 1 ? ;
%     no
```

```
% 1° caso: se X sta nella la prima lista e non(X) nella seconda :
risoluzione([X|R],L,Risultato,Value) :-
        member(non(X),L),
        cancella(non(X),L,R1),
        Value is 1,
        append(R,R1,Risultato).

% 2° caso: se non(X) sta nella prima lista e X nella seconda :
risoluzione([non(X)|R],L,Risultato,Value) :-
        member(X,L),
        cancella(X,L,R1),
        Value is 1,
        append(R,R1,Risultato).

% 3° caso: se X non si può semplificare  :
risoluzione([X|R],L,[X|Risultato],Value) :-
        risoluzione(R,L,Risultato,Value).


% ****************** 1.3 RISOLUZIONE TRA PIU CLAUSOLE ******************

% rsorite/4 : rsorite(ListaDiClausole,Clausola,Risultato,Value) risolve Clausola
% con l'insieme di clausole fornito da ListaDiClausole e mette il risultato in
% Risultato e il numero di passi che sono stati necessari per ottenere il risultato
% in Value.
% Esempio di utilizzazione :
% rsorite([[non(cocodrillo(X)),non(disprezzare(X))],[non(illogico(Y)),
% disprezzare(Y)]],[non(bambino(Z)),illogico(Z)],R,Value).
% --> R = [non(bambino(X)),non(cocodrillo(X))],
%     Y = X,
%     Z = X,
%     Value = 2 ? ;
%     R = [non(bambino(Y)),disprezzare(Y)],
%     Z = Y,
%     Value = 1 ? ;
%     R = [non(bambino(Z)),illogico(Z)],
%     Value = 0 ? ;
%     no

rsorite(L,T,Risultato,Value) :-
      rsorite2(L,L,T,Risultato,0,Value).

% rsorite2/6 : rsorite2(ListaDiClausole,TutteLeClausole,Clausola,RisultatoCorrente,
% ValueCorrente,ValueFinal).

% 1° caso: termina la recursione
rsorite2([],_,T,T,Value,Value).

% 2° caso: prova di risolvere X a partire della clausola T
rsorite2([X|R],Tutto,T,Risultato,L,Value) :-
        risoluzione(T,X,Risu,I),
        format("********************************** \n Insieme delle clausole
attive : ~p\n  \n Ragionamento : \n ~p \n  + \n ~p \n => \n ~p\n
\n",[Tutto,T,X,Risu]), % stampa del ragionamento
        Value2 is L+I,
        cancella(X,Tutto,NewTutto),
        rsorite2(NewTutto,NewTutto,Risu,Risultato,Value2,NewValue),
        Value is NewValue.

% 3° caso: se X con T non si può risolvere (cioè risoluzione(T,X,Risu,I) a
% ritornato un "no"), lanciamo la recursione sulle altre clausole non esaminate.
rsorite2([X|R],Tutto,T,Risultato,L,Value) :-
        rsorite2(R,Tutto,T,Risultato,L,Value).


% ********************** 1.4 SOLUZIONE OTTIMA **********************

% sorite/1 : sorite(L) trova la soluzione migliore per le clausole della
% nostra base di fatti e mette il risultato in L.
% Esempio di utilizzazione :
% Primo bisogna definire le clausole e inserirle nella nostra base di fatti.
% Lo facciamo via il predica predefinito assert/1.
% clauseLogique([non(bambino(X)),illogico(X)]).
% clauseLogique([non(coccodrillo(X)),non(disprezzare(X))]).
% clauseLogique([non(illogico(Y)),disprezzare(Y)]).
% assert(clauseLogiquee([non(bambino(X)),illogico(X)])).
```

```
% assert(clauseLogique([non(coccodrillo(X)),non(disprezzare(X))])).
% assert(clauseLogique([non(illogico(Y)),disprezzare(Y)])).
% Poi :
% sorite(L).
% --> L = [non(bambino(_A)),non(coccodrillo(_A))] ? ;
%     no

sorite(Soluzione) :-
       setof([X,V],sorite(X,V),Lista),
       ricostruisce(Lista,Lista2),
       ilpiupertinente(Lista2,Soluzione),
!.

% sorite/1 utilizza il predicato predefinito setof/3 : setof(Template,Goal,Set)
% crea una lista dei termini Template che soddisfano il goal e mette questa lista
% in Set.

% sorite/2 : sorite(Soluzione,Value) risolve l'insieme delle clausole messe dentro
% la nostra base di fatti, mette questa soluzione in Soluzione e la sua valutazione
% in Value.

sorite(Soluzione,Value) :-
        setof(X,clauseLogique(X),S),
% costruisce la lista S di tutte le clausole dentro la nostra base di fatti
        sorite(Soluzione,S,S,Value).

% sorite/4 : sorite(Soluzione,[X|R],S,Value)
sorite(Soluzione,[X|R],S,Value) :-
       cancella(X,S,S2),
       rsorite(S2,X,Soluzione,Value).

% ricostruise/2 : ricostruisce(Lista1,Lista2) ricostruisce la Lista1 [[Ui,Vi],...]
% in tale modo che Ui sia differente da Uj e Vi sia la valutazione più piccola e
% mette il risultato in Lista2.
ricostruisce(Lista,Lista2):-
       ricostruisce(Lista,Lista2,[]).

% ricostruisce/3 : ricostruisce(Lista1,Lista2,Accu) che ricostruisce Lista1 e mette
% il risultato in Lista2, utilizzando l'accumulatore Accu.

% 1°  caso: termina la recursione
ricostruisce([],Lista,Lista) :- !.

% 2° caso: il primo elemento della copia della lista non è ancora stato incontrata
ricostruisce([[A,B]|L],Lista,ListaSoFar):-
       not(member([A,_],ListaSoFar)),
       ListaSoFar2 = [[A,B]|ListaSoFar],
       ricostruisce(L,Lista,ListaSoFar2).

% 3° caso: il primo elemento della coppia è già stato incontrato, con una
% valutazione associata più alta.
ricostruisce([[A,B]|L],Lista,ListaSoFar):-
       member([A,X],ListaSoFar),
       B<X,
       cancella([A,X],ListaSoFar,ListaSoFar2),
       ListaSoFar3 = [[A,B]|ListaSoFar2],
       ricostruisce(L,Lista,ListaSoFar3).

% 4° caso: il primo elemento della coppia è già stato incontrato, con una
% valutazione associata più bassa.
ricostruisce([[A,B]|L],Lista,ListaSoFar):-
       ricostruisce(L,Lista,ListaSoFar).

% ilpiupertinente/2 : ilpiupertinente(Lista,Soluzione) trova nella Lista
% [[Ui,Vi],...] il Ui a cui è associato il Vi il più grande e mette questo Ui
% soluzione in Soluzione.
ilpiupertinente(Lista,Soluzione) :-
       max(Lista,0,Max,[],Soluzione).

% max/5 : max(Lista,MaxSoFar,MaxSoluzione,ItemSoFar,ItemSoluzione)

% 1° caso: termina la recursione
max([], MaxSoFar, MaxSoFar, ItemSoFar, ItemSoFar).

% 2° caso: la valore associata a l'elemento esaminato è più grande che il la più
% grande delle valore già incontrate.
```

```
max([[X,Number]|Rest], MaxSoFar, Max, ItemSoFar, Item) :-
    Number > MaxSoFar,
    max(Rest, Number, Max, X, Item).

% 3 caso: la valore associata a l'elemento esaminato è più piccola che il la più
% grande delle valore già incontrate.
max([[X,Number]|Rest], MaxSoFar, Max, ItemSoFar, Item) :-
    Number =< MaxSoFar,
    max(Rest, MaxSoFar, Max, ItemSoFar, Item).



% ************************************************************************
% *                            Secondo passo :                           *
% *                      ELABORAZIONE DELLE ESPRESSIONI                  *
% *                           IN LINGUAGGIO NATURALE                     *
% ************************************************************************


% ************** 2.1 STRUMENTI DI MANIPULAZIONE DELLE CATENE ***************

% traduce/2 : traduce(Catena,Tagliata) restituisce in Tagliata la lista degli atomi
% componendo Catena.
% Esempi di utilizzazione :
% traduce("i bambini sono illogici",L).
% --> L = [i,bambini,sono,illogici]
% traduce("nessun esercizio che è facile da l'emicrania",L).
% --> L = [nessun,esercizio,che,è,facile,da,'l\'emicrania']

traduce(Catena,Tagliata):-
      name(X,Catena),
      name(X,ASCII),
      taglia(ASCII,Tagliata).

% traduce/2 utilizza il predicato predefinito name/2 che permette di trasformare
% una lista di caratteri nella lista di codice ASCII corrispondenti.
% Il primo name/2, cioè name(X,Catena), permette di trasformare le apostrofi che
% Catena può contenere nel simbolo  \'.
% Il secondo name/2, cioè name (X,ASCII), trasforma X, cioè la Catena in cui sono
% state gestite le apostrofi via il primo name/2, nella lista corrispondente di
% codice ASCII.

% taglia/2 : taglia(Xs,Ys) trasforma una lista Xs di codici ASCII nella lista
% corrispondente di parole e la restituisce in Ys.
taglia(Xs,Ys):-
      !,
      taglia(Xs,[],[],Ys).

% taglia/4 : taglia(ListaASCII,Parole,Accu,Risu) utilizza due accumulatori : Parole
% stocca la parole in corso fino all'incontro di uno spazio mentre Accu stocca le
% prole individuate fin'adesso.

% 1° caso: termine la recursione
taglia([],Parole,Accu,Risu):-
      name(X,Parole),!,
      append(Accu,[X],Risu).

% 2° caso: se si incontra uno spazio (codice ASCII = 32)
taglia([32|L],Parole,Accu,Risu):-
      name(X,Parole),
      append(Accu,[X],Accu2),!,
      taglia(L,[],Accu2,Risu).

% 3° caso: altrimenti
taglia([X|L],Parole,Accu,Risu):-
      append(Parole,[X],ParoleCorrente),!,
      taglia(L,ParoleCorrente,Accu,Risu).

% singolare/2 : singolare(Plurale,Singolare) mette la parole Plurale al singolare e
% mette il risultato in Singolare.
% Esempi di utilizzazione :
% singolare(case,S).
% --> S = casa
% singolare(in,S).
% --> S = in
% singolare(tuoi,S).
```

```
% --> S = tuoo

% 1° caso: eccezioni
% Masculini che finiscono con la "a" e che fanno il plurale con la "i" :
singolare(poeti,poeta):-!.
singolare(logicisti,logicista):-!.

% particolarità NESSUN
singolare(nessuna,nessun).
singolare(nessuni,nessun).
singolare(nessuno,nessun).
singolare(nessune,nessun).

%particolarità TUTTO
singolare(tutte,tutto).

% Plurali irregolari :
singolare(uomini,uomo):-!.
singolare(buoi,bue):-!.
singolare(mogli,moglie):-!.
% Nomi invariabili :
singolare(Parole,Parole):-
      member(Parole,[caffe,citta,in,re,ipotesi,specie,moto,foto,cinema,auto,meno,se
nza]),
      !.
% Eccezioni:pronomi, avverbi, alcuni adiettivi in -e al singolare,verbi,nomi propri
:
singolare(esercizi,esercizio).

singolare(i,il).
singolare(le,la).
singolare(gli,il).
singolare(lei,lei).
singolare(vostri,vostro).
singolare(dei,il).
singolare(sole,solo).
singolare(delle,la).

singolare(di,di).
singolare(tranne,tranne).
singolare(se,se).
singolare(che,che).
singolare(mente,mente).

singolare(possibile,possibile).
singolare(turbolenti,turbolente).
singolare(turbolente,turbolente).
singolare(eccitabili,eccitabile).
singolare(eccitabile,eccitabile).
singolare(capace,capace).
singolare(forti,forte).
singolare(forte,forte).
singolare(occhi,occhio).
singolare(verdi,verde).
singolare(capaci,capace).
singolare(mortali,mortale).
singolare(socrate,socrate).
singolare(illogiche,illogico).
singolare(disprezzate,disprezzato).
singolare(facile,facile).

singolare(uccidere,uccidere).
singolare(brontolare,brontolare).
singolare(giocare,giocare).
singolare(sono,è).
singolare(vengono,viene).
singolare(siano,è).

% 2° caso: "i" finale (codice ASCII = 105). Può trattarsi del plurale di un
% singolare in "o" oppure del plurale di un singolare in "e" ; decidiamo
% arbitrariamente che si tratta del plurale di un singolare in "o". (codice ASCII =
% 111)
singolare(ParoleI,RisuO):-
      name(ParoleI,ParoleCodiceI),
      append(X,[105],ParoleCodiceI),
      append(X,[111],RisuCodiceO),
```

```
	name(RisuO,RisuCodiceO),
	!.
% 3° caso: "e" finale (codice ASCII = 101). Può trattarsi del plurale di un
% singolare in "a" oppure di une nome già al singolare ; decidiamo arbitrariamente
% che si tratta del plurale di un singolare in "a".
singolare(ParoleE,RisuA):-
	name(ParoleE,ParoleCodiceE),
	append(X,[101],ParoleCodiceE),
	append(X,[97],ParoleCodiceA),
	name(RisuA,ParoleCodiceA),
	!.

% 4° caso: altrimenti, si tratta di un singolare, lo lasciamo in quello stato
singolare(ParoleSing,ParoleSing):-
	!.

% tuttiSingolari/2 : tuttiSingolari(ListaSingolari,Risu) mette al singolare tutte
% le parole contenute nella lista ListaSingolare e mette la lista risultato in
% Risu.
% Esempio di utilizzazione:
% tuttiSingolari([vostri,gatti,sono,neri],S).
% --> S = [vostro,gatto,sono,nero]

tuttiSingolari([],[]).
tuttiSingolari([X|Y],[X2|Y2]):-
	singolare(X,X2),
	tuttiSingolari(Y,Y2).

% eliminaNS/2 ; eliminaNS(Lista,Risu) elimina delle parole contenute in Lista
% quelle non significative e mette il risultato in Risu.
% Esempio di utilizazzione:
% eliminaNS([il,gatto,della,vicina],L).
% --> L = [gatto,vicina]

% 1° caso: termine la recursione
eliminaNS([],[]).

% 2° caso: il primo elemento della lista è una parole significativa.
eliminaNS([X|Y],[X|Y2]):-
	significativo(X),
	eliminaNS(Y,Y2),
	!.

% 3° caso: il primo elemento della lista non è una parole significativa.
eliminaNS([_|Y],Y2):-
	eliminaNS(Y,Y2).

% significativo/1 : significativo(Parole) ritorna "yes" se la parole è
% significativa, "no" altrimenti.
significativo(Parole):-
		not(member(Parole,[loro,un,uno,una,dei,di,della,dello,del,
				delle,degli,le,i,la,il,lo,mio,mia,miei,
				mie,tuo,tua,tuoi,tue,suo,sua,suoi,sue,gli
				])).

% unSoloAtomo/2 : unSoloAtomo(ListaParole,Risu) costruisce a partire delle parole
% contenute in ListaParole un solo Atomo dove due parole sono separate da uno
% spazio.
% Esempio di utilizzazione:
% unSoloAtomo([il,gatto,della,vicina],A).
% --> A = 'il gatto della vicina'

unSoloAtomo(Parole,Atomo):-
	raggruppa(Parole,ASCII),
	append(Lista,[32],ASCII),
	name(Atomo,Lista).

% raggruppa/2 : raggruppa(Parole,ASCII)
raggruppa([],[]).
raggruppa([X|R],Accu):-
	name(X,ASCII),
	raggruppa(R,Accu2),
	append(ASCII,[32],AccuCorrente),
	append(AccuCorrente,Accu2,Accu).

% moltoVicini/2 : moltovicini(Espressione1,Espressione2) ritorna "yes" se
```

```
% Espressione1 e Espressione2 sono abbastanza similari, "no" altrimenti.
% Esempi di utilizzazione:
% moltoVicini([le,persone,illogiche],[illogiche]).
% --> yes
% moltoVicini([i,bambini],[le,persone,illogiche]).
% --> no

% 1° caso: se Espressione1 è più corta da Espressione2
moltoVicini(L1,L2):-
       length(L1,T1),
       length(L2,T2),
       T1=<T2,
       comune(L1,L2,Nb),
       Nb>T1/2.

% 2° caso: se Espressione1 è più lunga da Espressione2
moltoVicini(L1,L2):-
       length(L1,T1),
       length(L2,T2),
       T2<T1,
       comune(L2,L1,Nb),
       Nb>T2/2.

% moltoVicini/2 utilizza il predica predefinito length/2 : length(Lista,Lunghezza)
% identifica Lunghezza con la lunghezza di Lista, una lista di lunghezza
% determinata.

% comune/3 : comune(Espressione1,Espressione2,Nb) valuta il numero di caratteri
% che le due espressioni hanno in comune e mette il risultato in Nb.

% 1° caso: termine la recursione
comune([],_,0).

% 2° caso: se il primo elemento dell'espressione considerata appare anche
% nell'altra espressione.
comune([X|Xs],Ys,Nb):-
       member(X,Ys),
       !,
       comune(Xs,Ys,Nb1),
       Nb is Nb1 + 1.

% 3° caso: se il primo elemento dell'espressione considerata non appare nell'altra
% espressione.
comune([_|Xs],Ys,Nb):-
       !,
       comune(Xs,Ys,Nb).


% ************** 2.2 STRUMENTI DI MANIPULAZIONE DEI CONCETTI ***************

% La nostra base di fatti si compone da costante (termini non quantificati
universalmente nelle premesse), proprietà (es. rosso(Pippo))
% clauseLogique (costruite a partire delle proprietà e delle costanti) e di
univers.

% init/0 : init inizializza il sistema di risoluzione cancellando tutto quello che
% si trovava nella nostra basa di fatti.
init :-
       retractall(constante(_,_)),
       retractall(proprieta(_,_)),
       retractall(clauseLogique(_)),
       retractall(univers(_,_)).

% retractall(Head) cancella della basa di fatti tutte le clausole con una testa
% pari a Head.

% proprietaAssociata/3 : proprietaAssociata(Proprieta,Risu,Variabile) aggiunge la
proprietà Proprieta alla nostra base di fatti, la lega a Variabile e visualizza il
risultato in Risu
% Esempio di utilizzazione:
% proprietaAssociata([disposto,regolarmente],Risu,'questo esercizio').
% --> Nuova proprietà : disposto regolarmente
%     Risu = 'disposto regolarmente'('questo esercizio')

proprietaAssociata(Parole,Risu,Variabile) :-
        eliminaNS(Parole,ParoleSign),
```

```
        unSoloAtomo(ParoleSign,ParoleAtomo),
        aggiungeProprieta(ParoleAtomo,Parole,Proprieta),
        Risu=..[Proprieta,Variabile].

% Term=..List : List è una lista con una testa pari a l'atomo corrispondendo al
principale operatore di Term e con la coda pari a la lista degli argomenti di Term.

% aggiungeProprieta/3 : aggiungeProprieta(Parole,Atomo,Proprietà) aggiunge la
% proprietà(Parole,Atomo) alla nostra base di fatti se non si trova già.
% Esempio di utilizzazione:
% aggiungeProprieta('può uccidere un coccodrillo,[può,uccidere,un,coccodrillo],P).
% --> Nuova proprietà : può uccidere un coccodrillo
%     P = 'può uccidere un coccodrillo'
% aggiungeProprieta('può massacrare un coccodrillo,[può,massacrare,un,coccodrillo],
% P).
% --> Proprietà : può uccidere un coccodrillo
%     P = 'può uccidere un coccodrillo'

% 1° caso: se la proprietà si trova già nella nostra base di fatti
aggiungeProprieta(P,A,P) :-
       proprieta(_,P), !,
       format("Proprietà : ~a\n",[P]).

% 2° caso: se si trova nella nostra base di fatti una proprietà "vicina"
aggiungeProprieta(P,A,Q) :-
       proprieta(L,Q),
       moltoVicini(L,A), !,
       format("Proprietà : ~a\n",[Q]).

% 3° caso: se questa proprietà non si trova nella nostra base di fatti
aggiungeProprieta(P,A,P) :-
       format("Nuova proprietà : ~a\n",[P]),
       assert(proprieta(A,P)).

% costanteAssociata/2 : costanteAssociata(Costante,Risu) aggiunge Costante alla
% nostra base di fatti e visualizza il risultato in Risu.
% Esempio di utilizzazione:
% costanteAssociata([questo,esercizio],Risu).
% --> Nuova costante : questo esercizio
%     Risu = 'questo esercizio'

costanteAssociata(A,C) :-
       eliminaNS(A,Cte),
       unSoloAtomo(Cte,As),
       aggiungeCostante(As,A,C).

% universoAssociato/2 : De même pour les univers
universoAssociato(A,C) :-
       eliminaNS(A,Cte),
       unSoloAtomo(Cte,As),
       aggiungeUniverso(As,A,C).

% aggiungeCostante/3 : aggiungeCostante(Parole,Atomo,Proprietà) aggiunge la
constante(Parole,Atomo) alla nostra base di fatti se non si trova già

% 1° caso: se la costante si trova già nella nostra base di fatti
aggiungeCostante(P,A,P) :-
       constante(_,P), !,
       format("Costante : ~a\n",[P]).

% 2° caso: se si trova nella nostra base di fatti una costante "vicina"
aggiungeCostante(P,A,Q) :-
        constante(L,Q),
        moltoVicini(L,A), !,
       format("Costante : ~a\n",[Q]).

% 3° caso: se questa costante non si trova nella nostra base di fatti
aggiungeCostante(P,A,P) :-
       format("Nuova costante : ~a\n",[P]),
       assert(constante(A,P)).

% predicat aggiungeUnivers/3
aggiungeUniverso(P,A,P) :-
       univers(_,P), !,
       format("Universo : ~a\n",[P]).
aggiungeUniverso(P,A,Q) :-
```

```
        univers(L,Q),
        moltoVicini(L,A), !,
      format("Universo : ~a\n",[Q]).
aggiungeUniverso(P,A,P) :-
      format("Nuovo universo : ~a\n",[P]),
      assert(univers(A,P)).

% ********************* 2.3 RICONOSCENZA DI PATTERN *********************

% filtraggio/2 : filtraggio(Frase, Pattern) prova di identificare la Frase con il
% Pattern. Se non riesce, ritorna "no", si riesce ritorna le identificazione che ha
% fatto.
% Esempio di utilizzazione:
% filtraggio([bambino,sono,illogico],[A,sono,B]).
% --> A = [bambino],
%     B = [illogico]
% filtraggio([piccolo,bambino,sono,illogico],[A,sono,B]).
% --> A = [piccolo,bambino],
%     B = [illogico]

% 1° caso: termine la recursione
filtraggio([],[]).

% 2° caso: una variabile sostituisce un vuoto
filtraggio([],[V]).

% 3° caso: una variabile sostituisce un elemento
filtraggio([X|Xs],[V|Ys]) :-
      V=[X],
      filtraggio(Xs,Ys),
      !.

% 4° caso: una variabile sostituisce più elementi
filtraggio([X|Xs],[V|Ys]) :-
      filtraggio(Xs,[V1|Ys]),
      append([X],V1,V),
      !.

% 5° caso: parola della frase pari a parola del pattern
filtraggio([X|Xs],[X|Ys]) :-
      filtraggio(Xs,Ys).

% pattern/1 : pattern(Frase) prova di riconoscere nella Frase un pattern (modello)
% conosciuto. Quando riconosciuto, inserisce nella nostra base di fatti la clausola
% corrispondente. Se non riconosce nessun pattern, ritorna "no".
% Semplifichiamo i pattern, togliendo i "le","i","gli" e "tutti","tutte" che non
% apportano informazioni.
% Bisogna ordinare i predicati nel senso giusto per vedere per primo se non ci sono
% parole del tipo "soli" oppure "non" dentro la frase.

pattern(Xs):- filtraggio(Xs,[nessun,U,che,non,è,A,è,B]),
              universoAssociato(U,U2),
              append([non|A],[è],X),
              append(X,[non|B],X2),
              pattern(X2).

pattern(Xs):- filtraggio(Xs,[nessun,U,è,A,se,non,è,B]),
              universoAssociato(U,U2),
              append([tutto],A,X3),
              append(X3,[è],X),
              append(X,B,X2),
              pattern(X2).

pattern(Xs):- filtraggio(Xs,[nessun,U,che,è,A,è,B]),
              universoAssociato(U,U2),
              append([nessun|A],[è],X),
              append(X,B,X2),
              pattern(X2).


pattern(Xs):- filtraggio(Xs,[U,che,non,è,A,non,è,tra,B]),
              universoAssociato(U,U2),
              append([solo|B],[è],X),
              append(X,A,X2),
              pattern(X2).
```

```
pattern(Xs):- filtraggio(Xs,[U,che,non,è,A,non,è,B]),
              universoAssociato(U,U2),
              append([non|A],[è],X),
              append(X,[non|B],X2),
              pattern(X2).

pattern(Xs):- filtraggio(Xs,[U,che,è,A,è,B]),
              universoAssociato(U,U2),
              append([tutto],A,X4),
              append(X4,[è],X),
              append(X,B,X2),
              pattern(X2).

pattern(Xs):- filtraggio(Xs,[U,non,è,A,almeno,che,è,B]),
              universoAssociato(U,U2),
              append(B,[è],X),
              append(X,A,X2),
              pattern(X2).

pattern(Xs):- filtraggio(Xs,[U,non,è,A,se,è,B]),
              universoAssociato(U,U2),
              append(B,[non|è],X),
              append(X,A,X2),
              pattern(X2).

% les .... sont ... sauf s'ils sont ...
pattern(Xs):- filtraggio(Xs,[U,è,A,salvo,se,è,B]),
              universoAssociato(U,U2),
              append([non|A],[è],X),
              append(X,B,X2),
              pattern(X2).



% Gestion du sauf
pattern(Xs):- filtraggio(Xs,[U,è,A,salvo,se,è,B]),
              universoAssociato(U,U2),
              append([tutto],B,X3),
              append(X3,[non,è],X),
              append(X,A,X2),
              pattern(X2).


% (aggiunto per gestire il pattern "i ... che non sono ... non sono ..."
pattern(Xs):- filtraggio(Xs,[non,A,è,non,B]),
              proprietaAssociata(A,Ta,X),
              proprietaAssociata(B,Tb,X),
              assert(clauseLogique([Ta,non(Tb)])),
              format('Modelizzazione : ~p', [[Ta,non(Tb)]]).

% 1° caso: "I non A sono B" --> A,B
pattern(Xs):-
       filtraggio(Xs,[non,A,è,B]),
       proprietaAssociata(A,Ta,X),
       proprietaAssociata(B,Tb,X),
       assert(clauseLogique([Ta,Tb])),
       format('Modelizzazione 1: ~p\n', [[Ta,Tb]]).

% 2° caso: "Soli i A sono B" -->  non A, B
pattern(Xs):-
       filtraggio(Xs,[solo,A,è,B]),
       proprietaAssociata(A,Ta,X),
       proprietaAssociata(B,Tb,X),
       assert(clauseLogique([non(Tb),Ta])),
%      assert(clauseLogique([non(Ta),Tb])),

%      format('Modelisazione 2: ~p\n', [[non(Ta),Tb]]),
       format('Modelisazione 2: ~p\n', [[non(Tb),Ta]]).

% 3° caso: "I A non sono B" --> non A, non B
pattern(Xs):-
       filtraggio(Xs,[tutto,A,non,è,B]),
       proprietaAssociata(A,Ta,X),
       proprietaAssociata(B,Tb,X),
       assert(clauseLogique([non(Ta),non(Tb)])),
```

```
	format('Modelizzazione 3: ~p\n', [[non(Ta),non(Tb)]]).


% 6° caso: "nessun è A quando è B" --> non B, non A
pattern(Xs):-
	filtraggio(Xs,[nessun,è,A,quando,è,B]),
	proprietaAssociata(A,Ta,X),
	proprietaAssociata(B,Tb,X),
	assert(clauseLogique([non(Tb),non(Ta)])),
	format('Modelizzazione 6: ~p\n', [[non(Tb),non(Ta)]]).


% 5° caso: "nessun A è B" --> non A, non B
pattern(Xs):-
	filtraggio(Xs,[nessun,A,è,B]),
	proprietaAssociata(A,Ta,X),
	proprietaAssociata(B,Tb,X),
	assert(clauseLogique([non(Ta),non(Tb)])),
	format('Modelizzazione 5: ~p\n', [[non(Ta),non(Tb)]]).



% 7° caso: "A non è B" --> non B(A)
pattern(Xs):-
	filtraggio(Xs,[A,non,è,B]),
	costanteAssociata(A,Ta),
	proprietaAssociata(B,Tb,Ta),
	assert(clauseLogique([non(Tb)])),
	format('Modelizzazione 7: ~p\n', [non(Tb)]).

% 4° caso: "Tutti i A sono B" --> non A, B
pattern(Xs):-
	filtraggio(Xs,[tutto,A,è,B]),
	proprietaAssociata(A,Ta,X),
	proprietaAssociata(B,Tb,X),
	assert(clauseLogique([non(Ta),Tb])),
	format('Modelisazione 4: ~p\n', [[non(Ta),Tb]]).

% 8° caso: "A è B" --> B(A) : aggiungiamo una costante e la sua proprietà associata
pattern(Xs):-
	filtraggio(Xs,[A,è,B]),
	costanteAssociata(A,Ta),
	proprietaAssociata(B,Tb,Ta),
	assert(clauseLogique([Tb])),
	format('Modelisazione 8: ~p\n', [Tb]).

% ******************** 2.4 RISOLUZIONE DEL PROBLEMA *********************

% Premesse/1 : premesse(Frase) effettua tutte le operazioni necessarie alla
% creazione delle clausole e alla loro inserzione dentro la nostra base di fatti,
% cioè traduzione, messa delle parole al singolare, eliminazione delle parole non
% significative e riconoscenza di un pattern.
premesse(X) :-
	traduce(X,T),
	tuttiSingolari(T,S),
	eliminaNS(S,NS),
	pattern(NS).

% StampaSoluzione/1 : stampaSoluzione(X) è la risoluzione propriamente detta
stampaSoluzione(X):-
	sorite(L),
	espressione(L,Leggibile),
	unSoloAtomo(Leggibile,X).


% espressione/2 : espressione(Simbolico,Leggibile) prova di riconoscere alcune
% espressioni e mette il suo risultato in Leggibile.

espressione([non(Y)], [il,U,:,Costante,non,è,Proprieta]) :-
	univers(_,U),
	Y =..[Proprieta,Costante],
	 !.
espressione([non(Y)], [Costante,non,è,Proprieta]) :-
	Y =..[Proprieta,Costante],
	 !.
```

```
espressione([X], [Costante,è,Proprieta]) :-
       X =.. [Proprieta,Costante],
       !.


espressione([non(A),non(B)], [il, U,Prop1, non, è, Prop2]) :-
       univers(_,U),
        A =.. [Prop1,Val],
        B =.. [Prop2,Val],
        !.

espressione([non(A),non(B)], [il, Prop1, non, è, Prop2]) :-
        A =.. [Prop1,Val],
        B =.. [Prop2,Val],
        !.

espressione([non(A),B], [il, U, Prop1, è, Prop2]) :-
       univers(_,U),
        A =.. [Prop1,Val],
        B =.. [Prop2,Val],
        !.

espressione([non(A),B], [il, Prop1, è, Prop2]) :-
        A =.. [Prop1,Val],
        B =.. [Prop2,Val],
        !.

espressione([A,non(B)], [il, U, Prop1, è, Prop2]) :-
       univers(_,U),
        A =.. [Prop1,Val],
        B =.. [Prop2,Val],
        !.

espressione([A,non(B)], [il, Prop2, è, Prop1]) :-
        A =.. [Prop1,Val],
        B =.. [Prop2,Val],
        !.
         !.
% se non riconosciamo nessune di queste forme, lasciamo l'espressione cosi com'è.
espressione(X,X).



% ************************************************************************
% *               ESEMPI DI RISOLUZIONI CON QUESTO PROGRAMMA             *
% ************************************************************************

classique(X) :-
             init,
             premesse("socrate è un uomo"),
             premesse("tutti gli uomini sono mortali"),
             stampaSoluzione(X).
% X="socrate è mortale"

bambini(X) :-
       init,
       premesse("i bambini sono illogici"),
       premesse("nessuno è disprezzato quando è capace di uccidere un coccodrillo"),
       premesse("tutte le persone illogiche sono disprezzate"),
       stampaSoluzione(X).
% X="bambino non è capace uccidere coccodrillo"

regali(X) :-
       init,
       premesse("sole le pentole appartenendomi sono in ferro bianco"),
       premesse("i regali che vengono da lei sono utile"),
       premesse("nessuna pentola appartenendomi è utile"),
       stampaSoluzione(X).
% X="regalo che viene da lei non è in ferro bianco"

logicisti(X) :-
       init,
         premesse("nessuno dei vostri bambini è un possibile logicista"),
         premesse("tutti gli individui sani di mente sono possibile logicisti"),
         premesse("nessun infermo di mente è un giurato possibile"),
       premesse("i non infermo di mente sono individui sani di mente"),
```

```
	stampaSoluzione(X).
% X="il vostro bambino non è giurato possibile"

scuola(X) :-
	init,
	premesse("nessun bambino di meno dodici anni in questa scuola è interno"),
	premesse("tutti i bambini studiosi sono rossi"),
	premesse("nessun esterno è un elleniste"),
	premesse("soli i bambini di meno di dodici anni sono pigri"),
	premesse("i non esterni sono interni"),
	premesse("i non pigri sono studiosi"),
	stampaSoluzione(X).
% --> X = 'il ellenista è rosso'


fox(X) :-
	init,
	premesse("nessun fox-terrier è tra i segni del zodiaco"),
	premesse("nessun oggetto che non è un segno del zodiaco è una cometa"),
	premesse("soli i fox-terrier sono con una coda ricciuta"),
	stampaSoluzione(X).
% X="il oggetto con coda ricciuta non è cometa"

esercizi(X) :-
	init,
	premesse("gli esercizi che sono risoluti senza brontolare sono quei che
capisco"),
	premesse("questo esercizio non è disposto regolarmente"),
	premesse("nessun esercizio che è facile è emicranante"),
	premesse("gli esercizi che non sono disposti regolarmente non sono quei che
capisco"),
	premesse("gli esercizi sono risoluti senza brontolare salvo se siano
emicranante"),
	stampaSoluzione(X).
% X = 'il esercizio: questo esercizio non è facile'

anatre(X) :-
	init,
	premesse("le anatre di questo villaggio non sono a collo verde salvo se sono
a Mme Martin"),
	premesse("nessun anatra di questo vallaggio che è a Mme Martin è grigio"),
	stampaSoluzione(X).
% X = 'il anatra questo villaggio grigio è a collo verde"
```