ICOT Technical Report: TM-0027

TM-0027

An Ordered Linear Resolution Theorem Proving Program im Prolog

by

Kuniaki Mukai
and Koichi Furukawa

September, 1983

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

An Ordered Linear Resolution Theorem Proving Program in Prolog

Kunifaki Mukai

Koichi Furukawa

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

Abstract

This is a documentation of an ordered linear resolution theorem prover in Prolog. The program is a complete implementation in a theoretical sense. The program adopted a top down mixed search strategy. In spite of the complete implementation, the prover has shown a good performance for some sample problems. This is one evidence, we think, for that Prolog is a promising implementation language even in the theorem proving field. The document has a full program listing of the prover as an appendix including a powerful translator between logical formulas and conjunctive normal forms.

1 Introduction

This is a complete program of a theorem prover based on ordered linear resolution [Chang & Lee]. The program is written in Prolog. The search strategy used in the program is a kind of top down mixed search strategy consisting of depth_first and breadth_first [Chang & Lee]. For a given list of clauses, the prover try to find a proof tree. The proof tree can be displayed if a user want to see it. A user can give some control parameters to the prover. The first, say d, is a "local" height of a proof tree. That is, d is a parameter for the mixed strategy. Pure depth-first and breadth-first strategies are the cases in which d is an infinity and 1, respectively. In general, let N(i) be the set of all nodes of a proof tree whose heights from the root node of a tree are greater than $d^{*}(i-1)$ and less than or equal to $d^{*}i$. For the d, the prover generates nodes in N(1),N(2),.... in this order to build a proof tree. The second is the limit of the term height occurring in a proof tree. The third is the limit of the reference count for the clause for a proof tree. The second and third parameter are interpreted to be an infinity in case not specified, respectively.

2 Data Structure

A conjunctive normal form is represented by a list of clauses. A clause is represented by a list of literals. In an ordered linear proving process, a clause is represented as a duplication free list of literals

[Chang & Lee].

A Prolog variable is used as a logical variable. This makes substitution process unnecessary for the prover.

A center clause is the same as a clause except that a framed literal is allowed in it. A framed literal is the form of $(<literal>).

In the current version of the prover, the real form of a clause is <reference-counter> + <list-of-literals>, and <reference-counter> is the form s(s(...s(<variable>)...)). The number of the occurrences of s is the reference count of the clause concerned. The variable <variable> is instantiated to be the term s(X) when the clause is referred by the prover, where X is a new free variable.

3 Basic Operation

We give brief explanations to each basic operations of the prover.

a) renaming: renaming (copying) process is a very frequent operation. A simple method to rename using "assert" and "retract" was known. But we implemented another method without using them. It is very surprising that our naiive method is rather more efficient than the known version. The procedure name is "rename_term".

b) unification: the unifier unifies two terms given with occur_check, the unifier returns the height of the unified terms. "unify" is the procedure name.

c) factoring: generally, for a clause of length n, factoring process takes 2**n order steps. The prover, however, performs the process for the completeness of the prover.

d) merging: the prover must maintain a clause to be duplication free list. The maintenance is performed by merging the clause right. See "merge_right".

e) resolution: let C and D be a center clause and a side clause. The prover computes (ordered) reolvents R of D and C as follows. Let L be the left most literal of C. We suppose that L is not framed. The prover takes in a nondeterministic way such elements out of D that each of them is unified with ~L. So, the basic procedure for factoring process is "remove". And then, the prover performs the same operation on the pair of the literal L and the tail (cdr) part of the center clause C. Let C' and D' be the clauses obtained by the above process. The target resolvent R is the clause D' and $(L) and C' merged right together in this order. In this way, one partner of the pair to be unified in the factoring process is always the left most literal of the current center clause. This method gives some improvement of efficiency.

Although we have not tried yet, it seems to be possible that pre-factoring for each input clause may improve efficiency of the prover.

f) reduce: when the left most literal L in the current center clause

C is unifiable with the complement of some framed literal L' in C, the prover picks L out of C unifying L with L'. Then the prover will merge right the center clause since, as a side effect, of the unification, the center clause may become to have some duplicate elements.

g) tautology check: the prover gives a tautology check for the current center clause. The prover cuts any search path at the first center clause which has a complementary pair in it. The method of the check is a naiive one which takes n**2 order steps.

h) subsumption: the prover takes a subsumption test whether one clause C subsumes another one as follows. At first the prover fixes all the variables occurring in the clause D by applying the Prolog evaluable predicate "numbervars" to D. Next, for each element in C, the prover trys to find some unifiable literal in D with the complement of the element. By composing these "mesh" unifier obtained by the unifcations, the prover can know the existence of the substitution s such that s(C) is a sublist of D. The prover has to only choice the partner for each element in D such that the mapping is an order preserving function from the set C to D. This gives the prover some improvements.

i) conjunctive normal form: the prover has a translator as a utility between a first order logical formula and its conjunctive normal form. The translator optimizes the arities of Skolem functions generated during a translation. The translator produces a minimized normal form by applying well known Boolean laws.

4 Experiment

The table 1 is a comparison of our prover with the SENRI [Yamaguchi etal] of its old version. The SENRI has many proving strategies, and the data of SENRI in the table 1 are those of experiments under ordered linear mode. On the other hand our's has only ordered linear strategy. The SENRI is written in Fortran. Our prover is compiled Prolog codes. So this is not a serious comparison. Examples are from the end of the book [Chang & Lee]. It has nine examples. The machines of our prover and the SENRI are a DEC-2060 and a ACOS system 1000.

Our prover succeeded to prove six problems out of the nine, and four is the case with the SENRI. This difference, we think, may come from some differences of each implementations. The SENRI does not have full factoring process. Our prover has subsumption process as some kind of deletion strategy. Our prover suppressed output in the time measurement.

example	our prover	SENRI in OL
1	137 ms	X
2	X	X
3	X	X
4	X	X
5	83 ms	417 ms
6	87 ms	152 ms
7	298 ms	121 ms

8	572 ms	X
9	1123 ms	10254 ms

Table 1. run time statistics for our prover and SEMEI in ordered linear search mode.

The letter X found in the table means that the relevant example could not be proved by the relevant prover withih a reasonable time and space.

ACKNOWLEDGEMENT

We thank Dr. T. Yamaguchi for offering the data found in the table 1. The original unifier of our prover was written by Dr. H. Nagiya of R.I.M.S. Our colleagues, especially M. Asou and M. Kondou gave us some valuable comments.

REFERENCE

[Chang & Lee] Chang,C.L.,Lee,R.C.: Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York, 1973.

[Yamaguchi etal] Yamaguchi, Nishioka, Uchinami,Tezuka: Teiri Shoumei Shisutemu SEMEI no Kousei, Zinkou-Chinou to Taiwa-Gihou, 22-1, September,1981, (in Japanese).

[Loveland 78] Loveland,D.W.: Automated Theorem Proving, North-Holland, 1978.

Appendix-1: a listing of the prover.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Theorem prover based on ordered linear resolution. %
% Top down mixed search version.                      %
%    by  K.Mukai '83 July.                            %
%                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% logical operators:
:-op(900,fy,~).
:-op(910,xfy,and).
:-op(910,xfy,or).
:-op(920,xfy,'->').
:-op(920,xfy,'<-').
:-op(930,xfy,'<->').

:-op(800,fx,'@').   % @(P) means that the literal P is 'framed'.

:-public refute/5.
:-public prove/4.
:-public unsat/4.
:-public imply/5.

unsat(C_Set,Dep,Fun,Sel):-
        member(C,C_Set),
        refute(C_Set,C,Dep,Fun,Sel),
        !.

prove(Formula,Dep,Fun,Sel):-
        neg_translate(Formula,C_Set),
        member(C,C_Set),
        refute(C_Set,C,Dep,Fun,Sel),
        !.

imply(Left,Right,Dep,Fun,Sel):-
        universary_quantify(Left,L),
        universary_quantify(Right,R),
        neg_translate('->'(L,R),C_Set),
        unsat(C_Set,Dep,Fun,Sel).

refute(CJ,Center,Dep,Fun,Sel):-
        timer,
        abolish(event,1),
        abolish(goal,2),
        set_counter(CJ,CJC),
        rename_term(Center,RC,_),
        keep([],initial(RC),P),
        (mixed_strategy(CJC,RC,Dep,Dep,Fun,Sel,P,Proof),!,
         display(' a proof found !!'),nl,
         proof_tree(Proof);
         display(' no proof found !!'),nl  ),
        runtime(' time = ').

mixed_strategy(T,[],_,_,_,_,X,X):-!.
mixed_strategy(T,C,Max,0,Fun,Sel,X,_):-
        integer(Max),
        assertz(goal(C,X)),
```

```
        fail.
mixed_strategy(T,C,Max,Cur,Fun,Sel,X,Y):-
        (integer(Max),Cur>0,
          Cur1 is Cur-1;
         var(Max)           ),
        tautology_free(C),
        history(C),
        (reduce(C,R,Fun),
         keep(X,reduced(R),Z);
         C=[TC|_],
         complementary(TC,CC),
         candidate(T,CC,D,Sel),
         rename_term(D,D1,_),
         resolvent(D1,C,R,Fun),
         keep(X,resol(D,R),Z)),
        cutoff(R,R1) ,
        mixed_strategy(T,R1,Max,Cur1,Fun,Sel,Z,Y).
mixed_strategy(T,_,Max,Max1,Fun,Sel,_,Y):-
        integer(Max),
        Max==Max1,
        retract(goal(C,X)),
        mixed_strategy(T,C,Max,Max,Fun,Sel,X,Y).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Resolution, reduction, searching candidates             %
% in the given axiom set.                                  %
% All of the center clauses generated are stored          %
% to cut a 'loop'.                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-mode history(+).
history(C):-
        numbervars(C,1,_),
        event(D),
        subsume(D,C),
        !,fail.
history(C):-asserta(event(C)).

:-mode subsume(+,+).
subsume([],_):-!.
subsume([L|D],[M|C]):-
        unify_with_bound(L,M,_),
        subsume(D,[M|C]).
subsume(X,[_|Y]):-subsume(X,Y).

:-mode tautology_free(+).
tautology_free([]):-!.
tautology_free([X|Y]):-
        complementary(X,X1),
        \+ memq(X1,Y),
        tautology_free(Y).

:-mode set_counter(+,-).
set_counter([],[]):-!.
set_counter([X|Y],[Counter+X|Z]):-
        set_counter(Y,Z).
```

```
:-mode candidate(+,+,-,?).
candidate([Counter+C|_],Li,C,Sel):-
        similar_in(Li,C),
        count_up_to(Counter,Sel).
candidate([_|T],Li,C,Sel):-candidate(T,Li,C,Sel).

:-mode count_up_to(?,?).
count_up_to(X,Y):-var(Y),!.
count_up_to(X,Y):-var(X),!,Y>0,X=s(_).
count_up_to(s(X),Y):-Y>0,Z is Y-1,
        count_up_to(X,Z).

:- mode similar_in(+,+).
similar_in(C,[D|_]):-similar(C,D),!.
similar_in(C,[_|X]):-similar_in(C,X).

:-mode similar(+,+).
similar( ~ (A), ~ (B)):-!,
        functor(A,X,_),functor(B,X,_).
similar(A,B):-
        functor(A,X,_),functor(B,X,_).

:- mode resolvent(+,+,-,?).
resolvent(X,[Y|Z],U,Fun):-
        (integer(Fun),
          (functor(Y,(~),_),
          Fun1 is Fun+2,
          Fun2 is Fun+1;
          Fun1 is Fun+1,
          Fun2 is Fun+2 ),!;
         true                ),
        complementary(Y,CY),
        remove(Y,Z,V,_,Fun1),
        remove(CY,X,W,N,Fun2),
        N>0,
        union(['$'(Y)|V],[],V1),
        union(W,V1,U).

:-mode remove(+,+,-,-,?).
remove(X,[],[],0,_):-!.
remove(X,[Y|Z],U,N,Fun):-
        remove(X,Z,V,M,Fun),
        (unify_with_bound(X,Y,Fun),
         U=V,N is M+1;
         U=[Y|V],N=M    ).

:-mode cutoff(+,-).
cutoff(['$'(_)|X],Y):-!,cutoff(X,Y).
cutoff(X,X).

:-mode complementary(+,-).
complementary( ~ X,X):-!.
complementary(X, ~ X).

:-mode reduce(+,-,?).
reduce([L|X],Y,Fun):-
       complementary(L,M),
```

```
        reducible(N,X,Fun),
        delete_merge(L,X,Y).

:- mode reducible(+,+,?).
reducible(X,['$'(Y)|_],Fun):-
        unify_with_bound(X,Y,Fun).
reducible(X,[_|Y],Fun):-reducible(X,Y,Fun).

:- mode delete_merge(+,+,-).
delete_merge(X,[],[]):-!.
delete_merge(X,[Y|Z],U):-
        X==Y,!,
        delete_merge(X,Z,U).
delete_merge(X,[Y|Z],U):-
        delete_merge(X,Z,V),
        (member(Y,V),!,U=V;
         U=[Y|V]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% unifier with occur check and optional%
% depth check of functor nesting.      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-mode unify_with_bound(?,?,?).
unify_with_bound(X,Y,N):-var(N),!,
        unify(X,Y,_).
unify_with_bound(X,Y,N):-
        unify(X,Y,L),N>=L.

:-mode unify(?,?).
unify(X,Y,1) :- var(X),var(Y),!,X=Y.
unify(X,Y,N) :- var(X),!,occur_check(X,Y,N),X=Y.
unify(X,Y,N) :- var(Y),!,occur_check(Y,X,N),X=Y.
unify(X,Y,N) :- X=..[F|Xs],Y=..[F|Ys],
       unify_list(Xs,Ys,N),N is N+1.

:-mode uniffy_list(+,+).
unify_list([],[],0) :- !.
unify_list([X1|X2],[Y1|Y2],N) :-
      unify(X1,Y1,N),unify_list(X2,Y2,L),
      max(N,L,N).

:-mode occur_check(_,?).
occur_check(X,Y,1) :- var(Y),!,\+(X==Y).
occur_check(X,Y,N) :- Y=..[F|Ys],
      occur_check_list(X,Ys,N),
      N is N+1.

:-mode occur_check_list(-,+).
occur_check_list(X,[],0) :- !.
occur_check_list(X,[Y1|Y2],N) :-
      occur_check(X,Y1,N),
      occur_check_list(X,Y2,L),
      max(N,L,N).

:-mode max(+,+,-).
max(N,M,N):-N>M,!.
```

```
max(_,M,M).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Renaming of variables in terms.                              %
% Surprisingly,the short version using 'assert-retract' does %
% not work so efficiently as this version.                     %
% The reason is not known yet.                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-mode rename_term(?,-,?).
rename_term(X,Y,Z):-var(X),!,
        strict_in(X-Y,Z).
rename_term([],[],_):-!.
rename_term([X|Y],[Z|U],V):-!,
        rename_term(X,Z,V),
        rename_term(Y,U,V).
rename_term(X,Y,V):-X=..[A|L],
        rename_term(L,M,V),Y=..[A|M].

:- mode strict_in(?,?).
strict_in(X,Y):-var(Y),!,Y=[X|_].
strict_in(X-Y,[Z-Y|_]):-X==Z,!.
strict_in(X,[_|Y]):-strict_in(X,Y).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% predicates for primitive operations %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-mode member(?,+).
member(X,[X|_]).
member(X,[_|Y]):-member(X,Y).


:-mode union(+,+,-).
union([],X,X):-!.
union([X|Y],Z,U):-
         union(Y,Z,V),
         (memq(X,V),!,U=V;
          U=[X|V]).

:- mode memq(+,+).
memq(X,[Y|_]):-X==Y,!.
memq(X,[_|Y]):-memq(X,Y).

timer:-statistics(runtime,_).

runtime(M):-statistics(runtime,[_,T]),
        print(M),print(T),
        print(' millisec.'),nl.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Holding  resolutions chain and displaying the proof         %
% tree whose data representation is nearly the list of pairs %
% of center and side clause.                                  %
```

$$

```
:- mode keep(+,+,-).
keep(X,Y,Z):-display_proof,!,keep1(X,Y,Z).
keep(_,_,_).

:- mode keep1(+,+,-).
keep1(X,reduced(R),[reduced(R1)|X]):-!,
        rename_term(R,R1,_).
keep1(X,resol(S,C),[resol(S,C1)|X]):-
        rename_term(C,C1,_).
keep1(X,initial(C),[initial(C1)|X]):-
        rename_term(C,C1,_).

:- mode proof_tree(+).
proof_tree(X):-display_proof,!,
        proof_tree1(X),display('QED'),
        nl,nl.
proof_tree(_).

:- mode proof_tree1(+).
proof_tree1([]).
proof_tree1([X|Y]):-
        proof_tree1(Y),
        proof_tree2(X).

:- mode proof_trees2(+).
proof_tree2(initial(X)):-
        nl,nl,
        message('### linear proof list ####'),
        print1(X),nl,nl.
proof_tree2(reduced(R)):-
        message('+++ reduced +++'),
        print1(R),nl,nl.
proof_tree2(resol(S,C)):-
        tab(40),
        message('+++ side +++'),
        tab(40),
        print1(S),nl,nl,
        message('+++ center +++'),
        print1(C),nl,nl.

:- mode print1(+).
print1(X):-
        numbervars(X,1,_),print(X),nl,fail.
print1(_).

:- mode message(+).
message(M):-print(M),nl.
```

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$%
$ Example problems from the end of the book [Chan & and Lee 73] $
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$


ex1(Dep,Siz,Use):-
refute([ [p(g(X,Y),X,Y)],
         [p(X,h(X,Y),Y)],
         [~p(k(X),X,k(X))],
         [~p(X,Y,U),~p(Y,Z,V),~p(X,V,W),p(U,Z,W)],
         [~p(X,Y,U),~p(Y,Z,V),~p(U,Z,W),p(X,V,W)]],
         [~p(k(X),X,k(X))],
         Dep,Siz,Use).


ex2(Dep,Siz,Use):-
refute([ [p(e,X,X)],
         [p(X,e,X)],
         [p(X,X,e)],
         [p(a,b,c)],
         [~p(b,a,c)],
         [~p(X,Y,U),~p(Y,Z,V),~p(X,V,W),p(U,Z,W)],
         [~p(X,Y,U),~p(Y,Z,V),~p(U,Z,W),p(X,V,W)]],
         [~p(b,a,c)],
         Dep,Siz,Use).

ex3(Dep,Siz,Use):-
refute([ [p(e,X,X)],
         [p(l(X),X,e)],
         [~p(a,e,a)],
         [~p(X,Y,U),~p(Y,Z,V),~p(X,V,W),p(U,Z,W)],
         [~p(Y,Y,U),~p(Y,Z,V),~p(U,Z,W),p(X,V,W)]],
         [~p(a,e,a)],
         Dep,Siz,Use).


ex4(Dep,Siz,Use):-
refute([ [p(e,X,X)],
         [p(l(X),X,e)],
         [~p(a,X,e)],
         [~p(X,Y,U),~p(Y,Z,V),~p(X,V,W),p(U,Z,W)],
         [~p(X,Y,U),~p(Y,Z,V),~p(U,Z,W),p(X,V,W)]],
         [~p(a,X,e)],
         Dep,Siz,Use).


ex5(Dep,Siz,Use):-
refute([ [p(e,X,X)],
         [p(X,l(X),e)],
         [p(l(X),X,e)],
         [s(a)],
         [~s(e)],
         [~s(X),~s(Y),~p(X,l(Y),Z),s(Z)],
         [~p(X,Y,U),~p(Y,Z,V),~p(X,V,W),p(U,Z,W)],
         [~p(X,Y,U),~p(Y,Z,V),~p(U,Z,W),p(X,V,W)]],
         [s(a)],
         Dep,Siz,Use).
```

```
ex6(Dep,Siz,Use):-
refute([ [p(e,X,X)],
         [p(X,1(X),e)],
         [p(1(X),X,e)],
         [s(a)],
         [~s(e)],
         [~s(X),~s(Y),~p(X,1(Y),Z),s(Z)],
         [~p(X,Y,U),~p(Y,Z,V),~p(X,V,W),p(U,Z,W)],
         [~p(X,Y,U),~p(Y,Z,V),~p(U,Z,W),p(X,V,W)]],
         [s(a)],
         Dep,Siz,Use).

ex7(Dep,Siz,Use):-
refute([[p(a)],
         [m(a,s(c),s(b))],
         [m(X,X,s(X))],
         [~d(a,b)],
         [~m(X,Y,Z),m(Y,X,Z)],
         [~m(X,Y,Z),d(X,Z)],
         [~p(X),~m(Y,Z,U),~d(X,U),d(X,Y),d(X,Z)]],
         [~d(a,b)],
         Dep,Siz,Use).

ex8(L,F,M):-
refute([
         [1(1,a)],
         [d(X,X)],
         [~ d(X,Y),~ d(Y,Z),~ d(X,Z)],
         [p(X),d(g(X),X)],
         [p(X),1(1,g(X))],
         [p(X),1(g(X),X)],
         [~ p(X),~ d(X,a)],
         [~ 1(1,X1),d(f(X1),X1),~ 1(X1,a)],
         [~ 1(1,X),~ 1(X,a),p(f(X))]],
         [~ 1(1,X1),d(f(X1),X1),~ 1(X1,a)],
         L,F,M).


ex9(L,F,M):-
refute([
         [1(X,f(X))],
         [~ 1(X,X)],
         [~ 1(X,Y),~1(Y,X)],
         [~ d(X,f(Y)), 1(Y,X)],
         [p(X),d(h(X),X)],
         [p(X),p(h(X))],
         [~ p(X),~ 1(a,X), 1(f(a),X)],
         [p(X),1(h(X),X)]],
         [~ p(X),~ 1(a,X), 1(f(a),X)],
         L,F,M).
```

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$ Run tests for example problems from [Chan & Lee 73]              $
$                                                                  $
$ This is a slight modification of the logging file of the        $
$ tests.                                                           $
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$


Prolog-20  version 1.0
Copyright (C) 1981, 1983 by D. Warren, F. Pereira and L. Byrd
| ?- compile(tp).
tp compiled: 2909 words,     5.48 sec.
yes
| ?- [-example].
example reconsulted   1873 words      1.64 sec.
yes


$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$ Case of output suppress   $
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

| ?- ex1(_,_,_).
 a proof found !!
 time = 137 millisec.
yes
| ?- ex5(_,_,_).
 a proof found !!
 time = 83 millisec.
yes
| ?- ex6(_,_,_).
 a proof found !!
 time = 87 millisec.
yes
| ?- ex7(_,_,_).
 a proof found !!
 time = 298 millisec.
yes
| ?- ex8(_,_,_).
 a proof found !!
 time = 572 millisec.
yes
| ?- ex9(_,_,_).
 a proof found !!
 time = 1123 millisec.

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$ Run tests diplaying proof trees for the same examples $
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

yes
| ?- assert(display_proof).
yes
| ?- ex1(_,_,_).
 a proof found !!
*** linear proof list ****
[~p(k(_1),_1,k(_1))]
```

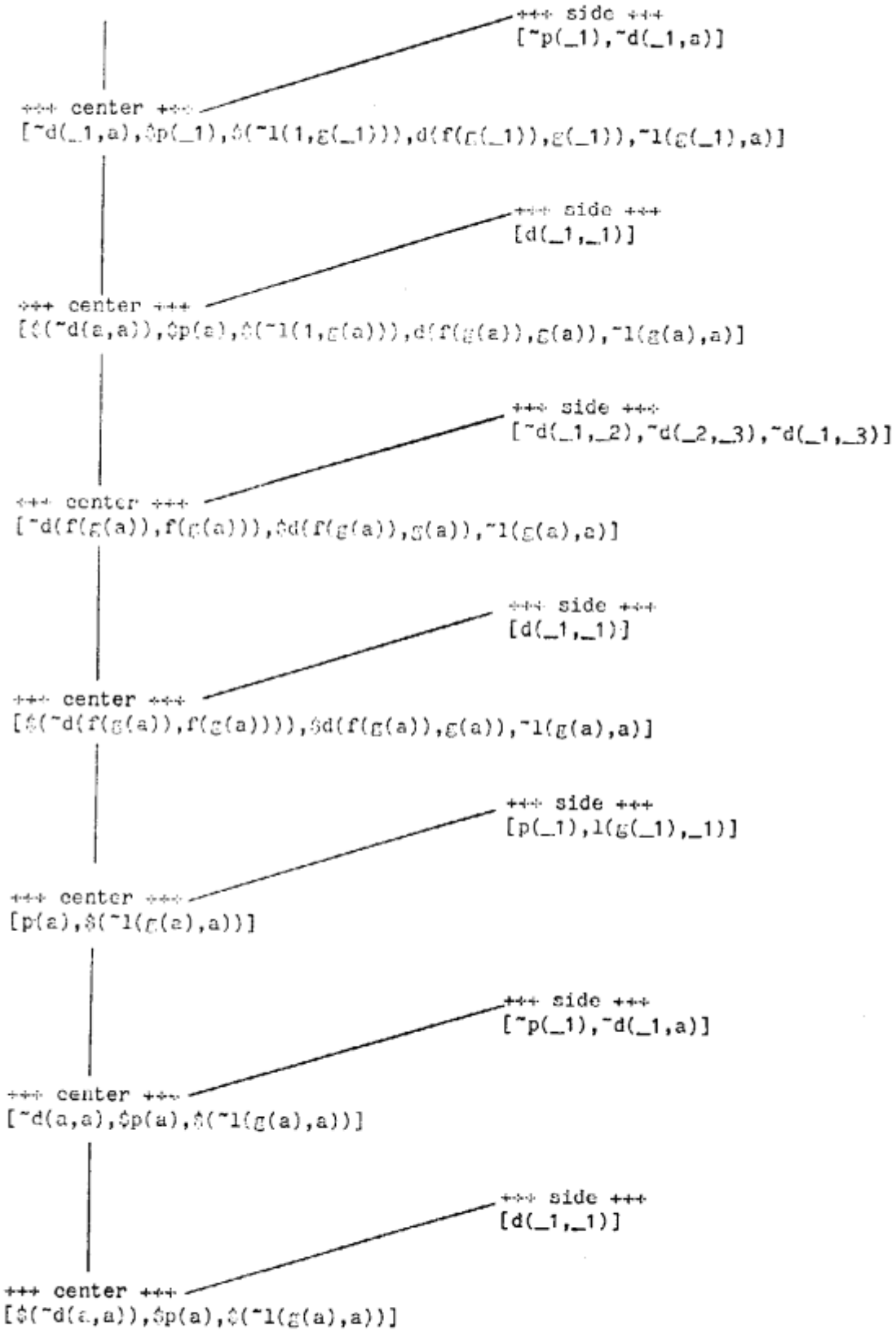

```
QED

 time = 310 millisec.

yes
| ?- ex5(_,_,_).
 a proof found !!


### linear proof list ####
[s(a)]
```

```
+++ center +++
[$s(e),$s(a)]


QED

 time = 508 millisec.

yes
| ?- ex6(_,_,_).
 a proof found !!


### linear proof list ####
[s(a)]
```

```
+++ side +++
[~s(_1),~s(_2),~p(_1,l(_2),_3),s(_3)]

+++ center +++
[~p(a,l(a),_1),s(_1),$s(a)]

+++ side +++
[p(_1,l(_1),e)]

+++ center +++
[$(~p(a,l(a),e)),s(e),$s(a)]

+++ side +++
[~s(e)]

+++ center +++
[$s(e),$s(a)]


QED

 time = 563 millisec.

yes
| ?- ex7(_,_,_).
 a proof found !!


### linear proof list ####
[~d(a,b)]
```

```
+++ side +++
[~p(_1),~m(_2,_3,_4),~d(_1,_4),d(_1,_2),
```

```
                                    d(_1,_3)]

+++ center +++
[~p(a),~m(b,b,_1),~d(a,_1),$(~d(a,b))]

                                    +++ side +++
                                    [p(a)]

+++ center +++
[$(~p(a)),~m(b,b,_1),~d(a,_1),$(~d(a,b))]

                                    +++ side +++
                                    [m(_1,_1,s(_1))]

+++ center +++
[$(~m(b,b,s(b))),~d(a,s(b)),$(~d(a,b))]

                                    +++ side +++
                                    [~m(_1,_2,_3),d(_1,_3)]

+++ center +++
[~m(a,_1,s(b)),$(~d(a,s(b))),$(~d(a,b))]

                                    +++ side +++
                                    [m(a,s(c),s(b))]

+++ center +++
[$(~m(a,s(c),s(b))),$(~d(a,s(b))),$(~d(a,b))]


QED

 time = 1219 millisec.

yes
| ?- ex8(_,_,_).
 a proof found !!


### linear proof list ####
[~l(1,_1),d(f(_1),_1),~l(_1,a)]

                                    +++ side +++
                                    [p(_1),l(1,g(_1))]

+++ center +++
[p(_1),$(~l(1,g(_1))),d(f(g(_1)),g(_1)),~l(g(_1),a)]
```


+++ side +++
[~p(_1),~d(_1,a)]
+++ center +++
[~d(_1,a),◊p(_1),□(~l(1,g(_1))),d(f(g(_1)),g(_1)),~l(g(_1),a)]
+++ side +++
[d(_1,_1)]
+++ center +++
[□(~d(a,a)),◊p(a),□(~l(1,g(a))),d(f(g(a)),g(a)),~l(g(a),a)]
+++ side +++
[~d(_1,_2),~d(_2,_3),~d(_1,_3)]
+++ center +++
[~d(f(g(a)),f(g(a))),◊d(f(g(a)),g(a)),~l(g(a),a)]
+++ side +++
[d(_1,_1)]
+++ center +++
[□(~d(f(g(a)),f(g(a)))),◊d(f(g(a)),g(a)),~l(g(a),a)]
+++ side +++
[p(_1),l(g(_1),_1)]
+++ center +++
[p(a),□(~l(g(a),a))]
+++ side +++
[~p(_1),~d(_1,a)]
+++ center +++
[~d(a,a),◊p(a),□(~l(g(a),a))]
+++ side +++
[d(_1,_1)]
+++ center +++
[□(~d(a,a)),◊p(a),□(~l(g(a),a))]

```
QED

 time = 2009 millisec.

yes
| ?- ex9(_,_,_).
 a proof found !!


### linear proof list ####
[~p(_1),~l(a,_1),l(f(a),_1)]
```



```
+++ center +++
[d(h(_1),_1),$(~p(_1)),~l(a,_1),l(f(a),_1)]
```


```
+++ center +++
[l(_1,h(f(_1))),$d(h(f(_1)),f(_1)),$(~p(f(_1))),~l(a,f(_1)),l(f(a),f(_1))]
```


```
+++ center +++
[~p(h(f(a))),l(f(a),h(f(a))),$l(a,h(f(a))),$d(h(f(a)),f(a)),$(~p(f(a))),~l(a,f(a
)),l(f(a),f(a))]
```


```
+++ center +++
[p(f(a)),$(~p(h(f(a)))),l(f(a),h(f(a))),$l(a,h(f(a))),$d(h(f(a)),f(a)),$(~p(f(a)
)),~l(a,f(a)),l(f(a),f(a))]

+++ reduced +++
[$(~p(h(f(a)))),l(f(a),h(f(a))),$l(a,h(f(a))),$d(h(f(a)),f(a)),$(~p(f(a))),~l(a,
f(a)),l(f(a),f(a))]
```


```
+++ center +++
[~l(h(f(a)),f(a)),$l(f(a),h(f(a))),$l(a,h(f(a))),$d(h(f(a)),f(a)),$(~p(f(a))),~l
```

```
(a,f(a)),1(f(a),f(a))]
```


```
QED

 time = 3389 millisec.

yes
| ?-
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Monkey-bananas problem from Loveland [78]          %
% m: monkey                                          %
% b: bananas                                         %
% c: chair                                           %
% f: floor                                           %
%                                                    %
% r(X,Y): X can reach Y                              %
% a(X): X is a dexterous animal                      %
% cl(X,Y): X is close to Y                           %
% u(X,Y): X is under Y                               %
% t(X): X is tall                                    %
% i(X): X is in the room                             %
% mv(X,Y,Z): X can move Y near Z                     %
% cb(X,Y): X can climb Y                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

monkey_bananas([
        [~ r(m,b)],
        [~ a(X),~ cl(X,Y),r(X,Y)],
        [~ o(X,Y),~ u(Y,b),~ t(Y),cl(X,b)],
        [~ i(X),~ i(Y),~ i(Z),~ mv(X,Y,Z),cl(Z,f),u(Y,Z)],
        [~ cb(X,Y),o(X,Y)],
        [a(m)],
        [t(c)],
        [i(m)],
        [i(b)],
        [i(c)],
        [mv(m,c,b)],
        [~ cl(b,f)],
        [cb(m,c)]]).

test_monkey(L,F,M):-monkey_bananas(T),unsat(T,L,F,M).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plane geometry problem from [Loveland 73]:                %
% an isoceles triangle has two equal anngles.               %
%                                                           %
% t(X,Y,Z): points X,Y and Z form a triangle.               %
% c(U,V,W,X,Y,Z): triangle UVW is congruent to triangle XYZ %
% s(U,V,X,Y): segment UV equals segment XY                  %
% a(U,V,W,X,Y,Z): angles UVW equals angle XYZ               %
%                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

geometry([
        [~ a(c,a,b,c,b,a)],
        [t(a,b,c)],
        [s(a,c,b,c)],
        [s(X,Y,Y,X)],
        [~ s(U,V,X,Y),s(X,Y,U,V)],
        [~ s(U,V,X,Y),s(V,U,X,Y)],
        [~ t(X,Y,Z),t(Y,Z,X)],
        [~ t(X,Y,Z),t(Y,X,Z)],
        [~ c(U,V,W,X,Y,Z),a(U,V,W,X,Y,Z)],
        [~ c(U,V,W,X,Y,Z),a(V,U,W,Y,X,Z)],
        [~ c(U,V,W,X,Y,Z),a(U,W,V,X,Z,Y)],
```

```
[~ t(U,V,W),~ t(X,Y,Z),~ s(U,V,X,Y),~ s(V,W,Y,Z),
 ~ s(U,W,X,Z),c(U,V,W,X,Y,Z)]]).

test_geometry(L,F,M):-geometry(T),unsat(T,L,F,M).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% executions of "monkey-bananas" and "plane-geometry" %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

| ?- test_monkey(_,_,_).
 a proof found !!
 time = 855 millisec.

yes
| ?- test_geometry(_,_,_).
 a proof found !!
 time = 1844 millisec.

yes
```

Appendix-2: a program list of the formula translater

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Translation of logical formula into clausal form.%
% Deep optimizations using boolean algebra laws    %
% are included. The following program list in this %
% appendix is self-contained for convenience. I.e. %
% several procedures are common with appendix-1    %
% (rename_term, for instance)                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-op(900,fy,~).
:-op(910,xfy,and).
:-op(910,xfy,or).
:-op(920,xfy,'->').
:-op(920,xfy,'<-').
:-op(930,xfy,'<->').

:-public translate/2.
:-public neg_translate/2.

:-mode translate(+,-).
translate(X,Y):-
        universary_quantify(X,UX),
        translate1(UX,Y).

:-mode neg_translate(+,-).
neg_translate(X,Y):-
        universary_quantify(X,QX),
        translate1(~QX,Y).

:-mode translate1(+,-).
translate1(X,Y):-
        eliminate_implication(X,X1),
        propagate_not(X1,X2),
        rename_bound(X2,RX2,[]),
        prenex(RX2,Formula),
        partition(Formula,Prenex,Body),
        simple(Body,Body1),
        clausal_form(Body1,Y),
        skolemize(Prenex),!.

:-mode tautology_free(+).
tautology_free([]):-!.
tautology_free([X|Y]):-
        complementary(X,X1),
        \+ memq(X1,Y),
        tautology_free(Y).

:-mode complementary(+,-).
complementary( ~  X,X):-!.
complementary(X, ~  X).

:-mode atomic_formula(?).
atomic_formula(X):-
        nonvar(X),
        functor(X,F,_),
        \+ logical_connective(F),
```

```
        \+ quantifier(F).

:-mode logical_connective(+).
logical_connective( and ).
logical_connective( or).
logical_connective( ~ ).
logical_connective(('<-')).
logical_connective(('->')).
logical_connective(('<->')).

:- mode quantifier(+).
quantifier(all).
quantifier(some).

:-mode universary_quantify(+,-).
universary_quantify(X,Y):-freevars(X,V),
        universary_quantify1(V,X,Y).

:-mode universary_quantify1(+,+,-).
universary_quantify1([],X,X):-!.
universary_quantify1([X|Y],Z,all(X,U)):-
       universary_quantify1(Y,Z,U).

:-mode freevars(?,-).
freevars(X,[]):-atomic(X),!.
freevars(X,[X]):-var(X),!.
freevars(all(X,F),Y):-!,
       freevars(F,Z),delete(X,Z,Y).
freevars(some(X,F),Y):-!,
       freevars(F,Z),delete(X,Z,Y).
freevars([],[]):-!.
freevars([X|Y],Z):-!,
      freevars(X,X1),
      freevars(Y,Y1),
      union(X1,Y1,Z).
freevars(X,Y):-X=..Z,freevars(Z,Y).

:- mode eliminate_implication(+,-).
eliminate_implication((Y<-X),Z):-!,
       eliminate_implication(( ~  X  or Y),Z).
eliminate_implication((Y->X),Z):-!,
       eliminate_implication(( ~  Y  or X),Z).
eliminate_implication((X<->Y),Z):-!,
       eliminate_implication(((X<-Y) and (Y<-X)),Z).
eliminate_implication((X or Y),(Z or U)):-!,
       eliminate_implication(X,Z),eliminate_implication(Y,U).
eliminate_implication((X and Y),(Z and U)):-!,
       eliminate_implication(X,Z),eliminate_implication(Y,U).
eliminate_implication( ~  X, ~  Y):-!,
       eliminate_implication(X,Y).
eliminate_implication(all(X,F),all(X,G)):-!,
       eliminate_implication(F,G).
eliminate_implication(some(X,F),some(X,G)):-!,
       eliminate_implication(F,G).
eliminate_implication(X,X).

:-mode propagate_not(+,-).
propagate_not(X,Y):-var(X),!,X=Y.
```

```
propagate_not( ~ (X and Y) ,(Z or U)):-!,
        propagate_not( ~ X,Z),
        propagate_not( ~ Y,U).
propagate_not( ~ (X or Y),(Z and U)):-!,
        propagate_not( ~ X,Z),
        propagate_not( ~ Y,U).
propagate_not( ~ ~ X, X):- !,
        propagate_not(X,Y).
propagate_not( ~ some(X,F), all(X,G)):-!,
        propagate_not( ~ F,G).
propagate_not( ~ all(X,F),some(X,G)):-!,
        propagate_not( ~ F,G).
propagate_not(X,Y):-X=..[F|A],
        (quantifier(F);logical_connective(F)),!,
        propagate_not(A,B),Y=..[F|B].
propagate_not([],[]):-!.
propagate_not([X|Y],[Z|U]):-!,
        propagate_not(X,Z),
        propagate_not(Y,U).
propagate_not(X,X).

:-mode rename_bound(?,-,+).
rename_bound(X,Y,Z):-var(X),!,(scan_var(X-Y,Z),!;X=Y).
rename_bound(all(X,F),all(Y,G),Z):-!,
        (scan_var(X-_,Z),!,
        rename_bound(F,G,[Y-Y|Z]);
        rename_bound(F,G,[X-X|Z]),Y=X).
rename_bound(some(X,F),some(Y,G),Z):-!,
        (scan_var(X-_,Z),!,
        rename_bound(F,G,[X-Y|Z]);
        rename_bound(F,G,[X-X|Z]),Y=X).
rename_bound([],[],_):-!.
rename_bound(X,X,_):-atomic(X),!.
rename_bound([X|Y],[Z|U],W):-!,
        rename_bound(X,Z,W),
        rename_bound(Y,U,W).
rename_bound(X,Y,Z):-X=..[F|U],
        rename_bound(U,V,Z),Y=..[F|V].

:- mode scan_var(+,+).
scan_var(X-Y,[Z-U|_]):-X==Z,!,Y=U.
scan_var(X,[_|Y]):-scan_var(X,Y).

:-mode prenex(+,-).
prenex(some(X,F),some(X,G)):-!,prenex(F,G).
prenex(all(X,F),all(X,G)):-!,prenex(F,G).
prenex((X and Y),Z):-!,prenex(X,X1),
        prenex(Y,Y1),
        merge_prenex((X1 and Y1),Z).
prenex((X or Y),Z):-!,prenex(X,X1),
        prenex(Y,Y1),
        merge_prenex((X1 or Y1),Z).
prenex(X,X).

:-mode merge_prenex(+,-).
merge_prenex((all(X,F) and all(X,G)),all(X,H)):-!,
        merge_prenex((F and G),H).
merge_prenex((some(X,F) or some(X,G)),some(X,H)):-!,
```

```
        merge_prenex((F or C),H).
merge_prenex((some(X,F) or G),some(X,K)):-!,
        merge_prenex((F or C),K).
merge_prenex((all(X,F) or C),all(X,K)):-!,
        merge_prenex((F or C),K).
merge_prenex((some(X,F) and G),some(X,K)):-!,
        merge_prenex((F and C),K).
merge_prenex((all(X,F) and G),all(X,K)):-!,
        merge_prenex((F and G),K).
merge_prenex((F or some(X,G)),some(X,K)):-!,
        merge_prenex((F or C),K).
merge_prenex((F or all(X,G)),all(X,K)):-!,
        merge_prenex((F or G),K).
merge_prenex((F and some(X,G)),some(X,K)):-!,
        merge_prenex((F and G),K).
merge_prenex((F and all(X,G)),all(X,K)):-!,
        merge_prenex((F and C),K).
merge_prenex(X,X).


:-mode partition(+,-,-).
partition(some(X,Y),[some(X)|Z],U):-!,
        partition(Y,Z,U).
partition(all(X,Y),[all(X)|Z],U):-!,
        partition(Y,Z,U).
partition(X,[],X).

:-mode skolemize(+).
skolemize(X):-sk([],X).

:-mode sk(+,+).
sk(X,[]):-!.
sk(X,[all(Y)|R]):-!,sk([Y|X],R).
sk(X,[some(T)|R]):-gensym('SK',G),!,
        T=..[G|X],sk(X,R).

:-mode remove_tautology(+,-).
remove_tautology([],[]):-!.
remove_tautology([[true]|R],S):-!,
        remove_tautology(R,S).
remove_tautology([X|Y],[X|Z]):-
        tautology_free(X),!,
        remove_tautology(Y,Z).
remove_tautology([X|Y],Z):-
        remove_tautology(Y,Z).

:-mode has_comp(+,+,-).
has_comp(F,X,Y):-
        member3(F,Z,X),
        complementary(Z,C),
        occur(F,C,Y),!.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The procedure 'simple' transforms "(X and Y) or Z"    %
% into "(X or Z) and (Y or Z)" and then simplifies      %
% the AND-OR form using boolean laws.                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
:-mode simple(+,-).
simple(~ true ,false):-!.
simple(~ false, true):-!.
simple(~ ~ X,Y):-!,simple(X,Y).
simple(X,Y):-
        functor(X,F,2),
        (F=(and);F=(or)),!,
        arg(1,X,X1),
        arg(2,X,X2),
        simple(X1,Z1),
        simple(X2,Z2),
        simple1(F,Z1,Z2,Y).
simple(X,X).

:-mode simple1(+,+,+,-).
simple1(and,X,Y,Z):-!,and_merge(X,Y,Z).
simple1(or,X,Y,Z):-!,or_merge(X,Y,Z).

:-mode and_merge(+,+,-).
and_merge(X,Y,false):-has_comp(and,X,Y),!.
and_merge(true,X,X):-!.
and_merge(X,true,X):-!.
and_merge(false,_,false):-!.
and_merge(_,false,false):-!.
and_merge(X and Y,Z,U):-!,
        and_merge(Y,Z,V),
        and_absorp(X,V,U).
and_merge(X,Y,Z):-
        and_absorp(X,Y,Z).

:-mode and_absorp(+,+,-).
and_absorp(X,Y and Z,Y and Z):-
        subset(or,Y,X),!.
and_absorp(X,Y and Z,U):-
        subset(or,X,Y),
        and_absorp(X,Z,U),!.
and_absorp(X,Y and Z, Y and U):-!,
        and_absorp(X,Z,U).
and_absorp(X,Y,X):-
        subset(or,X,Y),!.
and_absorp(X,Y,Y):-
        subset(or,Y,X),!.
and_absorp(X,Y,X and Y).

:-mode or_merge(+,+,-).
or_merge(X,Y,true):-has_comp(or,X,Y),!.
or_merge(X,true,true):-!.
or_merge(true,X,true):-!.
or_merge(X,false,X):-!.
or_merge(false,X,X):-!.
or_merge(X and Y, Z,U):-!,
        or_merge(X,Z,V),
        or_merge(Y,Z,W),
        and_merge(V,W,U).
or_merge(X, Y and Z,U):-!,
        or_merge(Y and Z,X,U).
or_merge(X or Y,Z,U):-
```

```
         occur(or,X,Z),!,
         or_merge(Y,Z,U).
or_merge(X or Y,Z,X or U):-!,
         or_merge(Y,Z,U).
or_merge(X,Y,Y):-
         occur(or,X,Y),!.
or_merge(X,Y,X or Y).

:- mode complementary(+,-).
complementary( ~ X,X):-!.
complementary(X, ~ X).

:-mode clausal_form(+,-).
clausal_form(false,[[]]):-!.
clausal_form(and(X,Y),[X1|Y1]):-!,
         clausal_form1(X,X1),
         clausal_form(Y,Y1).
clausal_form(X,[Y]):-
         clausal_form1(X,Y).

:-mode clausal_form1(+,-).
clausal_form1(or(X,Y),[X|Y1]):-!,
         clausal_form1(Y,Y1).
clausal_form1(X,[X]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Renaming of variables in terms.                               %
% Surprisingly,the short version using 'assert-retract' does%
% not work so efficiently as this version.                      %
% The reason is not known yet.                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-mode rename_term(?,-,?).
rename_term(X,Y,Z):-var(X),!,
        strict_in(X-Y,Z).
rename_term([],[],_):-!.
rename_term([X|Y],[Z|U],V):-!,
        rename_term(X,Z,V),
        rename_term(Y,U,V).
rename_term(X,Y,V):-X=..[A|L],
        rename_term(L,M,V),Y=..[A|M].

:- mode strict_in(?,?).

strict_in(X,Y):-var(Y),!,Y=[X|_].
strict_in(X-Y,[Z-Y|_]):-X==Z,!.
strict_in(X,[_|Y]):-strict_in(X,Y).



%%%%%%%%%%%%%%%%%%%%%%
% basic predicates  %
%%%%%%%%%%%%%%%%%%%%%%

:-mode union(+,+,-).
union([],X,X):-!.
union([X|Y],Z,U):-
```

```
        union(Y,Z,V),
        (memq(X,V),!,U=V;
         U=[X|V]).

:- mode memq(+,+).
memq(X,[Y|_]):-X==Y,!.
memq(X,[_|Y]):-memq(X,Y).

:-mode occur(+,+,+).
occur(F,X,Y):-
        Y=..[F,Z,U],
        !,
        (X==Z;
         occur(F,X,U)),
        !.
occur(F,X,Y):-X==Y.

:-mode member3(+,-,+).
member3(F,X,Y):-Y=..[F,X,_].
member3(F,X,Y):-Y=..[F,_,Z],!,
        member3(F,X,Z).
member3(F,X,X).

:-mode subset(+,+,+).
subset(F,X,Y):-X=..[F,X1,X2],!,
        occur(F,X1,Y),subset(F,X2,Y).
subset(F,X,Y):-occur(F,X,Y).

:-mode append(+,+,-).
append([],X,X):-!.
append([X|Y],Z,[X|U]):-append(Y,Z,U).

:-mode union(+,+,-).
union([],X,X):-!.
union([X|Y],Z,U):-
        union(Y,Z,V),
        (memq(X,V),!,U=V;
         U=[X|V]).

:- mode delete(?,+,-).
delete(X,[],[]):-!.
delete(X,[Y|Z],U):-X==Y,!,delete(X,Z,U).
delete(X,[Y|Z],[Y|U]):-delete(X,Z,U).

:-mode gensym(+,-).
gensym(Root,Atom):-
        get_num(Root,Num),
        name(Root,Name1),
        name(Num,Name2),
        append(Name1,Name2,Name),
        name(Atom,Name).

:-mode get_num(+,-).
get_num(Root,Num):-
        retract(current_num(Root,Num1)),!,
        Num is Num1+1,
        asserta(current_num(Root,Num)).
get_num(Root,1):-asserta(current_num(Root,1)).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Examples of conjunctive normal form through "translate".%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

| ?- timer,translate((a->b) and (b->c) and (c->d) and (d->a)-> (a<->d),X),
|     runtime.
runtime=183 ms

X = [[true]]

yes
| ?- translate((a->b)and (b->c),M).

     M= [[~a,b],[~b,c]]

yes
| ?- translate(~(a and b),M).

M = [[~a,~b]]

yes
| ?- translate((a and b) or c,M).

M = [[a,c],[b,c]]

yes
| ?- translate(a or ~a,M).

M = [[true]]

yes
| ?- translate(a and ~a,M).

M = [[]]

yes
| ?- translate((a and b)or (a and ~b) or (~a and b) or (~a and ~b),M).

M = [[true]]

yes
| ?- prove((a and b)or (a and ~b) or (~a and b) or (~a and ~b),_,_,_).
 a proof found !!
 time = 146 millisec.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% R.C. Moor's axiom for 'know' [from IJCAI 77]. %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

| ?- timer,translate(
|        (true(P)<->t(w0,P))
|        and
|        (t(W1,P1 and P2)<->t(W1,P1)and t(W1,P2))
|        and
|        (t(W1,P1 or P2)<->t(W1,P1)or t(W1,P2))
|        and
```

```
        (t(W1,P1->P2)<->(t(W1,P1)->t(W1,P2)))
        and
        (t(W1,P1<->P2)<->(t(W1,P1)<->t(W1,P2)))
        and
        (t(W1,~ P1)<-> ~ t(W1,P))
        and
        (t(W1,know(A1,P1))<->all(W2,k(A1,W1,W2)->t(W2,P1)))
        and
        k(A1,W1,W1)
        and
        (k(A1,W1,W2)->k(A1,W2,W3)->k(A1,W1,W3))
        and
        (k(A1,W1,W2)->k(A1,W1,W3)->k(A1,W2,W3))
        and
        (t(W1,know(A,P))<->
                all(W2, k(A,W1,W2)->t(W2,P))),
        CNF),
        runtime.
runtime=1786 ms

W3 = _1006,
P1 = _115,
P2 = _138,
A = _1259,
P = _38,
W1 = _99,
A1 = _727,
W2 = _767,
CNF = [[~t(SK4(_11581,_38,_99,_1259,_1006,_767,_727,_115,_138),_38),t(_99,know(_
1259,_38))],[k(_1259,_99,SK4(_11581,_38,_99,_1259,_1006,_767,_727,_115,_138)),t(
_99,know(_1259,_38))],[~k(_727,_99,_767),~k(_727,_99,_1006),k(_727,_767,_1006)],
[~k(_727,_99,_767),~k(_727,_767,_1006),k(_727,_99,_1006)],[k(_727,_99,_99)],[~t(
_99,know(_727,_115))],~k(_727,_99,_11581),t(_11581,_115)],[k(_727,_99,SK3(_38,_99
,_1259,_1006,_767,_727,_115,_138)),t(_99,know(_727,_115))],[~t(SK3(_38,_99,_1259
,_1006,_767,_727,_115,_138),_115),t(_99,know(_727,_115))],[~t(_99,~_115),~t(_99,
_38)],[t(_99,_38),t(_99,~_115)],[~t(_99,_115),t(_99,_138),~t(_99,_115<->_138)],[
t(_99,_115),t(_99,_138),t(_99,_115<->_138)],[~t(_99,_138),~t(_99,_115),t(_99,_11
5<->_138)],[~t(_99,_138),t(_99,_115),~t(_99,_115<->_138)],[~t(_99,_115->_138),~t
(_99,_115),t(_99,_138)],[t(_99,_115),t(_99,_115->_138)],[~t(_99,_138),t(_99,_115
->_138)],[~t(_99,_115 or _138),t(_99,_115),t(_99,_138)],[~t(_99,_115),t(_99,_115
 or _138)],[~t(_99,_138),t(_99,_115 or _138)],[t(_99,_138),~t(_99,_115 and _138)
],[~t(_99,_115),~t(_99,_138),t(_99,_115 and _138)],[t(_99,_115),~t(_99,_115 and
_138)],[~true(_38),t(w0,_38)],[~t(w0,_38),true(_38)],[~t(_99,know(_1259,_38)),~k
(_1259,_99,_12803),t(_12803,_38)]]

yes
```