Sue:
This is now ready to go to the printer

Robin Gibson

(to be charged to LPA16 Strat Grant)
use binding, double side as in TR 87-33

Printing Requisition/Graphic Services

26241

1. **Please complete unshaded areas** on form as applicable.
2. **Distribute copies** as follows: White and Yellow to Graphic Services. **Retain Pink Copies for your records.**
3. On completion of order the **Yellow copy will be returned** with the printed material.
4. **Please direct enquiries, quoting requisition number and account number,** to **extension 3451.**

TITLE OR DESCRIPTION
An Implementation of the Argument Understanding System (AUS) CS-88-40

DATE REQUISITIONED	DATE REQUIRED	ACCOUNT NO.
Nov. 4/88	ASAP	1 2 6 6 6 1 4 4 1

REQUISITIONER – PRINT	PHONE	SIGNING AUTHORITY
R. Cohen	4457	S. DeAngelis / R. Cohen

MAILING INFO – NAME	DEPT.	BLDG. & ROOM NO.	[X] DELIVER [ ] PICK-UP
Sue DeAngelis	C.S.	DC 2314	

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES: 67 NUMBER OF COPIES: 100

TYPE OF PAPER STOCK: [X] BOND [ ] NCR ____ PT. [X] COVER [ ] BRISTOL [X] SUPPLIED [ ] ____

PAPER SIZE: [X] 8½ x 11 [ ] 8½ x 14 [ ] 11 x 17 [ ] ____

PAPER COLOUR: [X] WHITE [ ] ____ INK: [X] BLACK [ ] ____

PRINTING: [ ] 1 SIDE ____ PGS. [X] 2 SIDES ____ PGS. NUMBERING: FROM TO

BINDING/FINISHING: [X] COLLATING [ ] STAPLING [ ] ____ HOLE PUNCHED [X] PLASTIC RING

FOLDING/PADDING CUTTING SIZE

Special Instructions

Please bind with plastic ring

Math fronts and backs enclosed

COPY CENTRE OPER. NO. BLDG. MACH. NO.

DESIGN & PASTE-UP OPER. NO. TIME LABOUR CODE
D01
D01
D01

TYPESETTING QUANTITY
PAP000000 T01
PAP000000 T01
PAP000000 T01

PROOF
PRF
PRF
PRF

NEGATIVES QUANTITY OPER. NO. TIME LABOUR CODE
FLM C01
FLM C01
FLM C01
FLM C01
FLM C01

PMT
PMT C01
PMT C01
PMT C01

PLATES
PLT P01
PLT P01
PLT P01

STOCK
001
001
001
001

BINDERY
RNG B01
RNG B01
RNG B01
MIS000000 B01

OUTSIDE SERVICES

$ ____ COST

TAXES – PROVINCIAL [ ] FEDERAL [ ]

GRAPHIC SERV. OCT. 85 482-2

An Implementation of the Argument Understanding System (AUS)

-- Integrating the Proposition Analyser with the Clue Interpreter, and the Evidence Oracle

Fei Song

Research Report CS-88-40
Department of Computer Science
October 1988

An Implementation of the Argument Understanding System (AUS)

— Integrating the Proposition Analyzer with the Clue Interpreter, and the Evidence Oracle

Fei Song

Logic Programming and Artificial Intelligence Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario, CANADA
N2L 3G1

Oct. 7, 1988

An Implementation of the Argument Understanding System
— Integrating the Proposition Analyzer with
the Clue Interpreter, and the Evidence Oracle

Fei Song

1 Introduction

This project is an implementation of the argument understanding system (AUS) presented in [Cohen83], which is shown by the following figure:

```
n.l.argument ---> Proposition Analyzer <---> argument rep.
                     /             \
                    /               \
        Evidence Oracle          Clue Interpreter
           /        \
          /          \
  speaker model   knowledge base
```

The Proposition Analyzer with Clue Interpreter, and the Evidence Oracle have been implemented respectively in [Smedley86, Smedley87], and in [Young87]. These two modules are integrated in this project by embedding the Evidence Oracle in the Proposition Analyzer with the Clue Interpreter such that whenever we need to decide the claim-evidence relation between two propositions, the Evidence Oracle will be called, rather than asking the user to provide the "yes" or "no" answer as is the case in [Smedley87].

The AUS parses an argument (in the form of a monologue, consisting of propositions) into an 'argument tree', with each node for a proposition and each link from a father node to its son node for the claim-evidence relation between the two corresponding propositions. When a proposition in an argument is passed to the Proposition Analyzer (PA), the PA will search, based on the restrictions of the transmission strategy, for a sequence of nodes from the established partial argument tree, and end this process when a claim-evidence relation between the current proposition and a node in the sequence is found. The claim-evidence relation between two propositions is decided by the Evidence Oracle (EO), but sometimes a call to EO can be saved if some clues, which are those words or phrases used to suggest how or where the propositions fit together, are provided. For example, the clue "as a result" indicates that the proposition which follows has a previous proposition as evidence.

There are three transmission strategies the AUS uses:

```
1. claim first -- each claim is followed by the evidence for it.
2. claim last -- each claim is preceded by the evidence for it.
3. hybrid -- each claim is either preceded or followed by its evidence.
```

Since the first two strategies are actually the special cases of the last one, only the algorithm for strategy 3 is used in Smedley's and our programs.

The EO takes two propositions P and Q, and answers the question 'Is P meant to be evidence for Q?'. The basic ideas for EO are from the logical rules of inference, e.g. Modus Ponens says: if A → B (major) and A (minor), then B (conclusion). However in natural language, some parts are often missing, as shown in the following list.

Cases	Given Premises	Conclusion
Normal	P -> Q, P	Q
Missing Minor	P -> Q	Q
Missing Major	P	Q

For these cases, EO needs to search for some other rules in the knowledge base and a model of the speaker. They are classified as five modules:

(1) facts -- propositions mutually believed by the system and the speaker.
(2) explicit -- propositions provided by the speaker from an argument.
(3) missing -- propositions derived from the argument and the knowledge of the system, and attributed to the speaker.
(4) stereotype -- propositions believed by a typical speaker.
(5) hearer -- propositions believed by the system. They are used as the stereotype for the speaker in our current implementation.

The rules in these modules have different priorities, called belief-levels, in determining the evidence relations. This is reflected in the table below, with 0 for truth and a small number for the most likely case:

Truth value in explicit/missing	Truth value in stereotype True	Unknown	False
True/True	0	1	2
True/Unknown	3	4	5
Unknown/True	6	7	8
Unknown/Unknown	9	10	11

There may be more than one group of rules matched for the Modus Ponens. For each group of them, the EO will evaluate the Max(imum) and the Sum of belief-levels of all the rules in the group (also called the missing evidences) and select the group with the smallest Max as the best answer, or if some Max's for those groups are equal, select the group with the smallest Sum as the answer. Note that in Young's program, Sum is calculated and used implicitly. Here we make it explicit because it also plays an important role in determining the best answer among a set of choices.

2 Integration of the two modules: PA and EO

In our implementation of the AUS system, we basically follow Young's suggestion in [Young87] and connect the two modules: PA and EO through a call from the "evidence_oracle" predicate in PA to the "oracle" predicate in EO. The result is whenever we need to decide the claim-evidence relation between two propositions, a call to the "oracle" in EO will give the answer automatically. In Smedley's implementation of PA, this is decided by asking the user to answer "yes" or "no".

The actual code for this connection can be seen as follows:

```
%
% the symmetric test of "yes" and "no" for evidence_oracle
%
not_evidence_oracle(E, A) <-
        unasserted_evidence(E, A);
not_evidence_oracle(E, A) <-
        oracle_answer(E, A, GoodX)&
        cut &
        eq(GoodX, 'n');

evidence_oracle(E, A, _) <-
        asserted_evidence(E, A);
evidence_oracle(E, A) <-
        oracle_answer(E, A, GoodX)&
        cut &
        eq(GoodX, 'y');
%
% if an evidence relation is not asserted, call the "oracle" in
% Young's program.
%
oracle_answer(E, A, GoodX) <-
        not(asserted_evidence(E, A))&
        not(unasserted_evidence(E, A))&
        call_oracle( E, A, GoodX )&
        assert_evidence(GoodX, E, A);
%
% This predicate is connected to the "oracle" in Young's program.
%
call_oracle( E, A, 'y' ) <-
        oracle( A, E );
call_oracle( E, A, 'n' );
```

Note that the predicate "evidence_oracle" is largely simplified since the "yes" or "no" answer will come from the call to EO, rather than from the user.

One major change from [Smedley87] is the input format to the PA module. In [Smedley87], the input format for an argument is simply a list of sentences, which are again a list of words. For example, the following argument:

```
(1)  Not everyone who is laughed at is a genius.
(2)  Sure, they laughed at Columbus;
(3)  but they also laughed at Bozo the Clown.
```

will be represented as an input list:

```
[[not everyone who is laughed at is a genius],
 [sure, they laughed at Columbus],
 [but they also laughed at Bozo the Clown]]
```

But in our AUS system, since we need to call EO for claim-evidence relations, we have to adapt the input into the format for Young's program [Young87], which is, in general, a list of propositions in a form of facts or rules. A fact is a simple predicate such as "genius(Columbus)". A rule is either an implication ($A \rightarrow B$) or a generalization ($A \not\rightarrow B$). Their formats can be shown by the following examples:

```
for_all( dangerous(X), [shark(X)] );
        % dangerous(X) <- shark(X)
for_most( [X], dangerous(X), [shark(X)] );
        % dangerous(X) ↛ shark(X)
```

The precondition part, e.g. shark(X) above, is, in general, a list of predicates. It's possible to write a piece of code to transform the list of words into a fact or a rule, but for simplicity, we will use Young's input format for the AUS system as well. However, we still use the function form for connective clues as in [Smedley87]. For example, a parallel clue in the sentence:

```
In addition, a tornado is passing through.
```

can be represented as:

```
parallel(passing_through(tornado))
```

For some other types of connective clues, see [Cohen87].

Another minor change to Smedley's program is the predicate: "assert_father_on_left". It is used to assert evidences to a father on the left. In [Smedley86], the predicate is:

```
assert_father_on_left(F C) <-
       not(rightmost_child(F _))
       assert(usr father(F C) [])
       assert(usr rightmost_child(F C) []);
assert_father_on_left(F C) <-
       rightmost_child(F X)
       assert(usr left_brother(C X) [])
       assert(usr father(F C) []);
```

Unfortunately, this predicate is only true for less than three evidences. For the more general case, we will change it into:

```
assert_father_on_left(F, C) <-
      not(rightmost_child(F, _))&
      assert(new, father(F, C))&
      assert(new, rightmost_child(F, C));
assert_father_on_left(F, C) <-
      rightmost_child(F, X)&
      find_leftmost(F, X, Z)&
      assert(new, left_brother(C, Z))&
      assert(new, father(F, C));

find_leftmost(F, X, Z) <-
      not(left_brother(_, X));
find_leftmost(F, X, Z) <-
      left_brother(Y, X)&
      find_leftmost(F, Y, Z);
```

Note that there are some syntactic changes in our code as well. This is required by the new version of WUP (Waterloo Unix-based Prolog). Both Smedley's and Young's programs are written in WUP2.0, but when we wrote our program, the system had developed into version 3.1. The basic changes, which you may have noticed from the above code, are in syntax. Here a comma is required to separate the arguments in a function or a predicate, and an ampersand, &, is used to separate the clauses in the body of a rule. There are small changes to some built-in predicates as well.

Generally, Smedley's program is well written and the predicates used are clearly defined and well commented. For Young's program, we made some changes, mainly on style, to split or merge or redefine some predicates to make them more clear and add comments for them. Since there are too many details involved, we will not show them here. For more information, you can refer to the code of our program in section 4.

3 How to use AUS

To use our integrated program, the user needs to know four user-level predicates: "add_beliefs", "del_beliefs", "analyse", and "parse". The first two predicates are used to add or delete beliefs from one of the four modules: "explicit", "missing", "hearer", and "facts". The general formats are as follows:

```
add_beliefs( M, L );
del_beliefs( M, L );
```

where M is one of the four modules above, and L is a list of beliefs or propositions. Beliefs or propositions are usually facts, and rules of implication (A $\rightarrow$ B) and generalization (A $\rightarrow$ B).

The predicate "analyse" has a list of propositions, i.e. the whole argument, as its input. As mentioned in the introduction, an argument can be given in a claim first or a claim last or a hybrid order of both. It can also include some types of connective clues, such as: detail(P), parallel(P), inference(P), and summary(P), in which P is any proposition. The "parse" corresponds to Young's initial implementation. It does a similar work as "analyse", but is restricted to an argument in the claim first order without clues.

There are some examples available in the machine of "watdragon" under the directory:"/u/natural/simple /new1". The examples consisting of "add_beliefs" and "parse" to test Young's program are prefixed by "mex", while the examples for the integrated program, i.e. consisting of "add_beliefs" and "analyse", are prefixed by "tex". These examples can be followed by the suffix "pre" or "post" or "hyb" to denote whether the arguments are given in pre-order (also claim first) or post-order (also claim last) or hybrid-order. If there is more than one case of an argument in the same order, they will be distinguished further by a number after "pre" or "post" or "hyb".

The integrated program is stored in "watdragon" at: "/u/natural/simple/new1". In order to test those examples, you need to do the following steps:

```
[1]% wup new1
new1: ?setup;
{}
new: ?consult(Example);
......
```

Note that the predicate "setup" is used to create and load the program modules into a new working module, called "new", and then initialize some predicates which are needed for the entire program. The Example above should be replaced by some example files under the directory: "/u/natural/simple/new1".

A typical start of an example is to use "add_beliefs(M, L)" to initialize the knowledge base in the system, i.e. to add some beliefs into the module M, usually in module "hearer", or "facts" if you are sure that the beliefs in L are mutually believed by both the speaker and the hearer. The module "explicit" is used to hold beliefs from the input argument, while the module "missing" to keep beliefs derived from the existing beliefs in the system. Then we use "analyse" or "parse" to input an argument. The AUS system will process

the input propositions one by one, and list the intermediate results about the tests done for possible claim-evidence relations between the current proposition and a previous one in the argument. When all propositions are processed, AUS will print out the analyzed tree structure for the argument if it is coherent.

There are also other output predicates available, such as:

```
print_beliefs( M );
```

where M is one of the four modules. They are usually used after the processing of an argument to check what are included in these modules. The last predicate is "list_all" in WUP3.1. It can be used to check the analyzed tree structure after the processing of an argument.

For more detailed information on how to use our program, consult the examples in section 5.

4 The Code of the AUS in WUP3.1

Start of Our Program

```
%  The predicate "setup " creates a new working module "new" and load mark
%  and trevor's programs into "new", then create and initialize the module
%  "beliefs", which is used to store beliefs in "explicit", "missing,
%  "hearer", and "facts".
%
setup <-   create(root, new)&
        load(new, b1, "/u/natural/simple/new1/mayoung/mark")&
        load(new, b2, "/u/natural/simple/new1/tjsmedley/trevor")&
        create(root, beliefs)&
        assert(beliefs, speaker(explicit, []))&
        assert(beliefs, speaker(missing, []))&
        assert(beliefs, hearer([]))&
        assert(beliefs, facts([]))&
        consult(new, keyboard);
```

Evidence Oracle

```
%  add_beliefs( M, L ) -- add the list of beliefs L to the belief_list of
%  module M, which could be one of "explicit", "missing", "hearer", and
%  "fatcs".
%
add_beliefs( M, L ) <-
        belief_list( M, B )&
        expand_rules( L, L' )&
        del_pairs( L', B, LB )&
        new_assert( M, LB );

%  del_beliefs( M, L ) -- delete the list of beliefs L from the beleifs_
%  list of module M.
%
del_beliefs( M, L ) <-
        belief_list( M, B )&
        remove_pairs( B, L, BL )&
        new_assert( M, BL );

%  Assert the resulting added or deleted belief_list into module M.
%
new_assert( hearer, New ) <-
        retract( beliefs, hearer(Old), [] )&
        assert( beliefs, hearer(New) ) & cut;
```

```
new_assert( facts, New ) <-
        retract( beliefs, facts(Old), [] )&
        assert( beliefs, facts(New) ) & cut;
new_assert( M, New ) <-
        retract( beliefs, speaker(M, Old), [] )&
        assert( beliefs, speaker(M, New) );

%  del_pairs(L, B, LB) -- delete commom elements in L and B, and merge
%  the two lists into LB.
%
del_pairs( [], L, L);

del_pairs( [H|T], L, L') <-
        member( H, L)&
        cut &
        del_pairs( T, L, L');
del_pairs( [H|T], L, [H|L']) <-
        del_pairs( T, L, L');

%  remove_pairs(B, L, BL) -- remove L from B, put the result into BL.
%
remove_pairs( [], _, [] );

remove_pairs( [H|T], L, L' ) <-
        member( H, L )&
        cut &
        remove_pairs( T, L, L' );
remove_pairs( [H|T], L, [H|T'] ) <-
        remove_pairs( T, L, T' );

%  This predicate is used to collect all of the possible sets of evidence
%  for P <- Q, and put them in L.
%
all_evidence( P, Q, L ) <-
        assert( doing_evidence([]) )&
        evidence( P, Q, M )&
        report(P, Q, M)&
        believable( speaker, M )&
        retract( doing_evidence(L) )&
        include( M, L, L' )&
        assert( doing_evidence(L'), [] )&
        fail;
all_evidence( P, Q, L ) <-
        retract( doing_evidence(L) );
```

```
%  This predicate checks if M is included in the list L'.  If not, add it
%  to L'.
%
include( E, [], [E] );

include( E, [E|L], [E|L] ) <-
        cut;
include( E, [F|L], [F|L'] ) <-
        include( E, L ,L' );

%  belief_level( L, Max, Sum ) -- calculate the max and sum of belief
%  levels for the belief list L.
%
belief_level( [], 0, 0 ) <-
        cut;
belief_level( [H|T], Max, Sum ) <-
        cut &
        belief_level_( H, B )&
        belief_level( T, B', T' )&
        max( B, B', Max )&
        sum( B, T', Sum );

%  This predicate determines the belief level for the proposition P.
%
belief_level_( P, B ) <-
        not( counter_example(speaker, P) )& % from denial
        explicit_level( P, Be )&
        missing_level( P, Bm )&
        stereotype_level( P, Bs )&
        sum( Be, Bm, B' )&
        sum( B', Bs, B );

explicit_level( P, 0 ) <-
        example( explicit, P )& % from belief
        cut;
explicit_level( P, 6 );

missing_level( P, 0 ) <-
        example( missing, P )& % from belief
        cut;
missing_level( P, 3 );
```

```
stereotype_level( P, 0 ) <-
        example( stereotype, P )& % from belief
        cut;
stereotype_level( P, 2 ) <-
        counter_example( stereotype, P )& % from denial
        cut;
stereotype_level( P, 1 );

%  belief_list( M, L ) -- L is the list of beliefs in module M.
%
belief_list( speaker, L ) <-
       cut &
       retrieve( beliefs, speaker(explicit, L1), [])&
       retrieve( beliefs, speaker(missing, L2), [])&
       retrieve( beliefs, hearer(L3), []) &
       append( L1, L2, L')&
       append( L', L3, L );
belief_list( hearer, L ) <-
       cut &
       retrieve( beliefs, hearer(L), [] );
belief_list( facts, L ) <-
       cut &
       retrieve( beliefs, facts(L), [] );
belief_list( M, L ) <-
       retrieve( beliefs, speaker(M, L), [] );
belief_list( stereotype, L ) <-
       retrieve( beliefs, hearer(L), [] );

%  example( M, P ) -- P is an element of the belief list of module M.
%  examples( M, L ) -- example(M P) holds for each P in the list L.
%
example( M, P ) <-
        beliefs( M, L )&
        member( P, L );

examples( M, [] );
examples( M, [H|T] ) <-
        example( M, H )&
        examples( M, T );

%  believable( M, L ) -- no element of L is contradicted in M.
%
believable( M, L ) <-
        not( counter_example_one( M, L ) );
```

```
%  counter_example( M, P ) -- lnot(P) is an element of the belief
%     list of module M.
%  counter_example_one( M, L ) -- counter_example(M P) holds for
%     some P in list L.
%
counter_example( M, P ) <-
        lnot( P, NotP )&
        beliefs( M, L )&
        member( NotP, L );

counter_example_one(M, L) <-
        member(P, L)&
        counter_example(M, P);

%  clause( M, H, B ) -- H is implied by B in module M.
%
clause( M, H, B ) <-
        belief_list( M, L )&
        member( for_all(H, B), L );

%  beliefs( M, L ) -- L is a list of beliefs holding in module M.
%
beliefs( speaker, L ) <-
        belief_list( speaker, L );
beliefs( M, L ) <-
        belief_list( M, L );
beliefs( M, L ) <-
        ne( M, facts )&
        belief_list( facts, L );

%  This predicate is used to select the best set of beliefs B from the
%  list L, and put the remaining sets in the list R.
%
best_of( L, B, R ) <-
        best_of_( L, B, Bb, Tb, R );
best_of_( [H], H, Bh, Th, [] ) <-
        belief_level( H, Bh, Th )&
        cut;
best_of_( [H|T], H, Bh, Th, [S|Rs] ) <-
        belief_level( H, Bh, Th )&
        best_of_( T, S, Bs, Ts, Rs )&
        lt( Bh, Bs )&
        cut;
```

```
best_of_( [H|T], H, Bh, Th, [S|Rs] ) <-
        belief_level( H, Bh, Th )&
        best_of_( T, S, Bs, Ts, Rs )&
        eq( Bh, Bs )&
        le( Th, Ts )&
        cut;
best_of_( [H|T], S, Bs, Ts, [H|Rs] ) <-
        belief_level( H, Bh, Th )&
        best_of_( T, S, Bs, Ts, Rs );

%  evidence( P, Q, Missing ) -- means that Q^Missing together form
%     an evidence for P.
%
evidence( for_all(H, B), E, M ) <- % evidence in the form
        cut &  % of examples.
        example_( H, B, E, M );
evidence( lnot( for_all(H, B) ), E, M ) <- % evidence for disbelief
        cut & % in a rule.
        count_example( H, B, E, M );
evidence( for_most(V, H, B), E, M ) <- % evidence for a gener-
        cut & % alization: examples and
        example_count( H, B, E, M );            % counters.
evidence( lnot( for_most(V, H, B) ), E, M ) <- % evidence against same.
        cut & % counters and examples.
        count_example( H, B, E, M );
evidence( H, for_all(H, B), B ) <- % missing minor premise.
        cut;
evidence( H, for_most(V, H, B), [for_most(V, H, B)|B] ) <-
        cut; % using generalization.
evidence( H, E, M ) <- % missing major premise.
        from( E, [H], [], M);                   % E ^ M -> H.
evidence( H, E, [for_most(V, H, B)|B'] ) <-     % missing a rule of
        beliefs( speaker, L )&                  %  generalizations.
        member( for_most(V, H, B), L )&
        delete( E, B, B' );

%  count_example( H, B, E, M ) -- returns first count_example_, then
%    then examples of ( H B E M ).
%  example_count( H, B, E, M ) -- returns first example, then
%    count_example_ of ( H B E M ).
%
count_example( H, B, E, M ) <-
        count_example_( H, B, E, M );
```

```
count_example( H, B, E, M ) <-
        example_( H, B, E, M );

example_count( H, B, E, M ) <-
        example_( H, B, E, M );
example_count( H, B, E, M ) <-
        count_example_( H, B, E, M );

example_( H, B, E, M ) <-
        contraposals( for_all(H,B), L )&
        cut &
        member( for_all(NewH, NewB), L )&
        from( E, [NewH|NewB], [], M);

%  count_example_( H, B, E, M ) -- means that E^M -> ~(B -> H).
%
count_example_( H, B, E, M ) <-
        lnot( H, NH )&
        from( E, [NH|B], [], M );

%  from( Q, Fringe, Visited, Missing ) -- searches through known rules
%    for Q as evidence to some predicate in Fringe. Search method is
%    the depth-first-search with loop checking.
%
from( Q, F, VL, M ) <-                % Q is one of the missing evidences.
        delete( Q, F, M )&
        cut;
from( Q, F, VL, M ) <-                % Q couldn't be one of the missing
        lnot( Q, NQ )&                %  evidences.
        member( NQ, F )&
        cut &
        fail;
from( Q, F, VL, M ) <-                % find a related rule of implication
        delete( H, F, M1 )&           %  in module "facts".
        not( member( H, VL ) )&
        clause( facts, H, B )&
        append( M1, M2, M )&
        from( Q, B, [H|VL], M2 );
from( Q, F, VL, M ) <-
        delete( H, F, M1 )&
        not( member( H, VL ) )&
        clause( explicit, H, B )&
        append( M1, M2, M )&
        from( Q, B, [H|VL], M2 );
```

```
from( Q, F, VL, M ) <-
        delete( H, F, M1 )&
        not( member( H, VL ) )&
        clause( missing, H, B )&
        append( M1, M2, M )&
        from( Q, B, [H|VL], M2 );
from( Q, F, VL, [for_all(H, B)|M] ) <-
        delete( H, F, M1 )&
        not( member( H, VL ) )&
        clause( stereotype, H, B )&
        append( M1, M2, M )&
        from( Q, B, [H|VL], M2 );

%  The predicate expand_rules(L, L') is used to expand L into L'.
%  Here we only consider the expansion of the rules: for_all(H, B).
%
expand_rules( [], [] );
expand_rules( [for_all(H, B)|L], EREL ) <-
        cut &
        contraposals( for_all(H, B), ER )&
        append( ER, EL, EREL )&
        expand_rules( L, EL );
expand_rules( [H|T], [H|ET] ) <-
        expand_rules( T, ET );

%  Find all of the contrapositive rules of for_all(H, B).
%
contraposals( for_all(H, B), [for_all(H, B)|L] ) <-
        lnot( H, NH )&
        contraposals_( B, [NH], L );

contraposals_( [], _, [] );
contraposals_( [H|T], D, [for_all(NH, DT)|L] ) <-
        lnot( H, NH )&
        append( D, T, DT )&
        contraposals_( T, [H|D], L );

implied( M, G ) <-
        implied_( M, G, [] );

implied_( M, G, A ) <-            % if not(G) is in A, we find a rule.
        lnot( G, NG )&
        member( NG, A );
```

```
implied_( M, G, A ) <-                % may be succeed in the first time.
        example( M, G );              % this is the ground case.
implied_( M, G, A ) <-                    % for a rule of for_all(G, B)
        example( M, for_all(G, B) )&      % test all elements in B.
        implied_all( M, B, [G|A] );

%  Check the body list B to see if all elements are implied_(M, H, A),
%    where H is in B.
%
implied_all( M, [], A );

implied_all( M, [H|T], A ) <-
        implied_( M, H, A )&
        implied_all( M, T, A );

%  lnot( P, Q ) -- P is the logical not of Q.
%
lnot( P, lnot(P) ) <-
        ne( P, lnot(_) );
lnot( lnot(P), P ) <-
        ne( P, lnot(_) );

%  The evidence_oracle takes two propositions P and Q, and decide whether
%    Q is the evidence for P.
%
oracle( P, Q ) <-    % don't accept Q as an
        member(P, [for_all(H,B),for_most(V,H,B)])& % example if there is
        above( Q, P )&    % a place for it above
        cut(oracle(P,Q)) & fail;    % in the tree.
oracle( P, Q ) <-
        believable( speaker, [P, Q] )&
        add_beliefs( explicit, [P, Q] )&
        cut &
        all_evidence( P, Q, L )&
        return( P, Q, L );

above( Q, P ) <-
        assertion( done( A, P, B, R ), [] )&
        or( member( Q, B ), above( Q, A ) );
```

```
%  parse( L ) -- make sense of this argument (claim-first order only).
%
parse( [H|T] ) <-
        parse_( T, e(H, []), R, [] )&
        printree( R );

parse_( [H|T], e(L, B), R, T' ) <-
        printf( "considering % for %\n", [H, L] )&
        oracle( L, H )&
        printf( "success % for %\n\n", [H, L] )&
        parse_( T, e(H, []), S, T'' )&
        parse_( T'', e(L, [S|B]), R, T' );
parse_( [H|T], e(L, B), e(L, B), R ) <-
        printf( "failure % for %\n\n", [H, L] )&
        eq( R, [H|T] );
parse_( [], R, R, [] );

%  print the argument tree.
%
printree( e(H, L) ) <-
        printf( "The argument tree is:\n\n", [])&
        printree_( 0, e(H, L) );
printree_( I, e(H, L) ) <-
        tab( I )&
        printf( "%\n", [H] )&
        sum( I, 1, I1 )&
        printree_( I1, L );
printree_( _, [] );
printree_( I, [H|T] ) <-
        printree_( I, T )&
        printree_( I, H );

%  print_beliefs( M ) -- list the beliefs in module M.
%
print_beliefs( M ) <-
        belief_list( M, L )&
        printf( "\n module: %\n\n", [M] )&
        print_list( L );

print_list( [] );
print_list( [H|T] ) <-
        printf( "\t%\n", [H] )&
        print_list( T );
```

```
%  Display that Q is the evidence for P with M missing, and report the
%  belief_level for each predicate in M and for the whole conjunction.
%
report( P, Q, M ) <-
        printf("\n% is evidence for\n% with\n% missing.\n",[Q,P,M])&
        report_each(M, 0, 0);

%  B1 is the maximum level among those of all predicates in M, while B2
%    is the sum of belief_levels of all predicates in M.
%
report_each( [], B1, B2 ) <-
        printf("The conjunction has belief level Max:%,Sum:%\n",[B1,B2]);
report_each( [M|M'], B1, B2 ) <-
        belief_level_( M, B )&
        cut &
        printf( "  % has belief level %\n", [M, B] )&
        max( B1, B, B1' )&
        sum( B2, B, B2' )&
        report_each( M', B1', B2' );
report_each( [M|_], _, _ ) <-
        printf( "  % is not believed by the model\n", [M] );

%  This predicate is used in "oracle" in the form of return(P, Q, L).
%  In the done(P, Q, B, R),  P, Q are given predicates, B, R from
%  best_of( L, B, R).
%
return( P, Q, L ) <-
        best_of( L, B, R )&
        belief_list( stereotype, M )&
        remove_pairs( B, M, BM )&
        add_beliefs( missing, BM )&
        assert( done( P, Q, B, R ) );
return( P, Q, L ) <-
        retract( done( P, Q, B, [] ) )&
        cut &
        to_generalize( P, Q, R )&
        add_beliefs( stereotype, [R] )&
        eq( R, for_most(V, P, [Q]) )&
        add_beliefs( missing, [R] );
return( P, Q, L ) <-
        retract( done( P, Q, B, R ) )&
        del_beliefs( missing, B )&
        return( P, Q, R );
```

```
to_generalize( Q, P, for_most(V, Q', [P']) ) <-
        generalize_pair( Q, P, Q', P', V )&
        plausible( Q', P', V );

plausible( Q, P, V ) <-
        lnot( Q, Not_Q )&
        lnot( P, Not_P )&
        rule_examples( P, Q, V, Npq )&
        rule_examples( Not_Q, Not_P, V, Nnqnp )&
        rule_examples( Not_Q, P, V, Nnqp )&
        gt( Npq, Nnqp )&
        gt( Nnqnp, Nnqp );

rule_examples( Q, P, V, 1 ) <-
        printf( "examples of (% ^ %)\n", [Q, P] )&
        implied( hearer, Q )&
        implied( hearer, P )&
        cut &
        printf( "#(% ^ %) = %\n\n", [Q, P, 1] );
rule_examples( Q, P, V, 0 ) <-
        printf( "#(% ^ %) = %\n\n", [Q, P, 0] );

generalize_pair( Q, P, Gq, Gp, V ) <-
        is_functor( Q )&
        is_functor( P )&
        functor( Q, [Qf|Qlist] )&
        functor( P, [Pf|Plist] )&
        pairings( Qlist, Plist, Qnew, Pnew, V )&
        functor( Gq, [Qf|Qnew] )&
        functor( Gp, [Pf|Pnew] );

pairings( L1, L2, L1', L2', V ) <-
        isolate_variables( L1, V1, G1 )&
        isolate_variables( L2, V2, G2 )&
        find_pairs( G1, G2, Replace, V )&
        replace( L1, Replace, L1' )&
        replace( L2, Replace, L2' );

%  isolate_variables( L, V, G ) -- separate variables from L, and put
%  them into V and remaining (terms) into G.
%
isolate_variables( [], [], [] );
```

```
isolate_variables( [X|L], [X|V], G ) <-
        is_var( X )&
        cut &
        isolate_variables( L, V, G );
isolate_variables( [C|L], V, [C|G] ) <-
        is_const( A )&
        cut &
        isolate_variables( L, V, G );
isolate_variables( [List|L], V, G ) <-
        is_list( List )&
        cut &
        append( List, L, NewL )&
        isolate_variables( NewL, V, G );
isolate_variables( [Q|L], V, G ) <-
        is_functor( Q )&
        cut &
        functor( Q, [Qf|Qlist] )&
        append( Qlist, L, NewL )&
        isolate_variables( NewL, V, G );

%  find_pairs( G1, G2, Replace, V ) -- finds common terms in G1 and G2,
%    and put them into Replace in a form of [H|V'], while V' in V.
%
find_pairs( [], _, [], [] ) <-
        cut;
find_pairs( _, [], [], [] ) <-
        cut;
find_pairs( [H|T], L, [[H|V]|R], [V|Lv] ) <-
        member( H, L )&
        cut &
        find_pairs( T, L, R, Lv );
find_pairs( [H|T], L, R, Lv ) <-
        find_pairs( T, L, R, Lv );

%  replace( L, Replace, L' ) -- replaces L into L' according to Replace
%    list. If some element in L is a var, member of Replace, functor, or
%    list, replace them on the following rules, else copy it into L'.
%
replace( [], L, [] );                   % terminate here.

replace( [H|T], L, [H|T'] ) <-          % copy the variables.
        is_var( H )&
        cut &
        replace( T, L, T' );
```

```
replace( [H|T], L, [V|T'] ) <-          % replace H by V if [H|V] in L.
        member( [H|V], L )&
        cut &
        replace( T, L, T' );
replace( [H|T], L, [H'|T'] ) <-         % replace the functor by L.
        is_functor( H )&
        cut &
        functor( H, [Hf|Hlist] )&
        replace( Hlist, L, Hlist' )&
        functor( H', [Hf|Hlist'] )&
        replace( T, L, T' );
replace( [H|T], L, [H'|T'] ) <-         % replace the list by L.
        is_list( H )&
        cut &
        replace( H, L, H' )&
        replace( T, L, T' );
replace( [H|T], L, [H|T'] ) <-          % copy anything else.
        replace( T, L, T' );
```

Proposition Analyzer with Clues

```
%  This is the user level predicate of the agrgument analysis package.
%  The argument is passed as a list of sentences, each sentence being a
%  form of predicates. For example, the sentence "Socrates is a man" will
%  be represented as "man(socrates)".  The "analyse" predicate will first
%  clear the auxillary data-base and insert three assertions in order
%  that "unknown predicate" errors do not occur, then print out the given
%  argument, and call the predicate for buliding the argument tree. The
%  method used here is the hybrid order of claim-first and claim-last
%  with the connective clue processing.  The predicate "build_tree" will
%  place assertions in the database, which will be of the form root(X) or
%  father(X Y) where these assertions define the argument tree. Finally,
%  the argument tree is printed out.
%
analyse(Argument) <-
        clear_aux(new) &
        assert(new, asserted_evidence([], []))&
        assert(new, unasserted_evidence([], []))&
        assert(new, father([], []))&
        print_argument(Argument)&
        build_tree(Argument)&
        nl&
        printf("Argument tree:", [])&
        nl(2)&
        root(X)&
        write(X)&
        nl&
        print_tree(X,1)&
        nl;

%  This predicate puts the information from the evidence oracle into the
%  database. This prevents redundant questions from being asked. It also
%  stores the information that one sentence is *not* evidence for another.
%
assert_evidence('y', E, A) <-
        assert(new, asserted_evidence(E, A));
assert_evidence('n', E, A) <-
        assert(new, unasserted_evidence(E, A));

%  This predicate puts the information that F is the father of C in the
%  database. It keeps the appropriate information for obtaining the
%  rightmost child of the father.
%
```

```
assert_father(F, C) <-
        not(rightmost_child(F, _))&
        assert(new, father(F, C))&
        assert(new, rightmost_child(F, C));
assert_father(F, C) <-
        rightmost_child(F, X)&
        delax(new, rightmost_child(F, X), [])&
        assert(new, left_brother(X, C))&
        assert(new, rightmost_child(F, C))&
        assert(new, father(F, C));

%  This predicate puts the information that F is the father of C in the
%  database. It keeps the appropriate information for obtaining the
%  rightmost child of the father.
%
assert_father_on_left(F, C) <-
        not(rightmost_child(F, _))&
        assert(new, father(F, C))&
        assert(new, rightmost_child(F, C));
assert_father_on_left(F, C) <-
        rightmost_child(F, X)&
        find_leftmost(F, X, Z)&
        assert(new, left_brother(C, Z))&
        assert(new, father(F, C));

%  This predicate finds the left most child for a father.
%
find_leftmost(F, X, X) <-
        not(left_brother(_, X));
find_leftmost(F, X, Z) <-
        left_brother(Y, X)&
        find_leftmost(F, Y, Z);

%  This predicate will attach all the sons of L which are evidence for N
%  below N. It will also make them no longer sons of L. The sons are added
%  in "reverse" order so that the correct rightmost_child relations are
%  preserved.
%
attach_sons(L, _) <-
        not(father(L, _));
attach_sons(L, N) <-
        rightmost_child(L, Son)&
        not_evidence_oracle(Son, N);
```

```
attach_sons(L, N) <-
        rightmost_child(L, Son)&
        evidence_oracle(Son, N)&
        assert_father_on_left(N, Son)&
        remove_rightmost(L)&
        attach_sons(L, N);

%  Calls the predicate for building the argument tree.  The first calls to
%  assert are to prevent unknown predicate errors from occurring. After
%  building the tree with the predicate "hybrid", the root is found
%  (failing if there %is more than one root), and the proper assertion is
%  put in the database.
%
build_tree([H | T]) <-
        assert(new, asserted_evidence(X, dummy))&
        assert(new, left_brother(nil, nil))&
        assert(new, rightmost_child(nil, nil))&
        hybrid_(T, H, dummy)&
        cut &
        all_of(Roots, X, father(dummy, X))&
        eq(Roots, [Root])&
        assert(new, root(Root));

%  This evidence oracle will symmetrically check "yes" or "no" claim-
%  evidence relation between E and A. The first clause checks whether the
%  information is already in the database.  The predicate "oracle_answer"
%  will first make sure that the information is not already in the
%  database. Then it calls the "oracle" in mark's program to decide the
%  claim-evidence relation automatically.
%
not_evidence_oracle(E, A) <-
        unasserted_evidence(E, A);
not_evidence_oracle(E, A) <-
        oracle_answer(E, A, GoodX)&
        cut &
        eq(GoodX, 'n');
evidence_oracle(E, A) <-
        asserted_evidence(E, A);
evidence_oracle(E, A) <-
        oracle_answer(E, A, GoodX)&
        cut &
        eq(GoodX, 'y');
```

```
%  This predicate will call the evidence oracle to decide the relation
%  between E and A.
%
oracle_answer(E, A, GoodX) <-
        not(asserted_evidence(E, A))&
        not(unasserted_evidence(E, A))&
        call_oracle( E, A, GoodX )&
        assert_evidence(GoodX, E, A);

%  This predicate is connected to the predicate "oracle" in mark's
%  program.
%
call_oracle( E, A, 'y' ) <-
        printf("considering % for %\n", [E, A])&
        oracle( A, E )&
        printf("success % for %\n\n", [E, A]);
call_oracle( E, A, 'n' ) <-
        printf("failure % for %\n\n", [E, A]);

%  This predicate will get the list of children of Root in the order in
%  which they are to be printed out.
%
get_children(Root, Children) <-
        get_children_(Root, [], Children);

%  This predicate gets the children of Root ordered with the leftmost
%  first
%
get_children_(Root, [], []) <-
        not(rightmost_child(Root, _));
get_children_(Root, [], Children) <-
        rightmost_child(Root, Child)&
        get_children_(Root, [Child], Children);
get_children_(Root, [H | T], [H | T]) <-
        not(left_brother(_, H));
get_children_(Root, [H | T], Children) <-
        left_brother(Left, H)&
        get_children_(Root, [Left, H | T], Children);

%  This builds the argument tree for hybrid type of arguments. The
%  algorithm is from R. Cohen [Cohen, 1987].  The first entries handle
%  connective clues.
```

```
hybrid_(_, detail(_), dummy) <-
        printf("failed due to detail clue", [])&
        nl&
        printf("no father found", [])&
        nl&
        printf("tree so far is:", [])&
        nl;
hybrid_(_, parallel(_), dummy) <-
        not(rightmost_child(dummy, _))&
        printf("failed due to parallel clue", [])&
        nl&
        printf("no possible brothers", [])&
        nl&
        printf("tree so far is:", [])&
        nl;
hybrid_(_, inference(_), dummy) <-
        not(rightmost_child(dummy, _))&
        printf("failed due to inference clue", [])&
        nl&
        printf("no possible son to re-attach", [])&
        nl&
        printf("tree so far is:", [])&
        nl;
hybrid_(_, summary(_), dummy) <-
        not(rightmost_child(dummy, _))&
        printf("failed due to summary clue", [])&
        nl&
        printf("no possible sons to re-attach", [])&
        nl&
        printf("tree so far is:", [])&
        nl;
%
hybrid_(S, parallel(T), L) <-
        not(rightmost_child(L, _))&
        printf("parallel clue saves an oracle call", [])&
        nl&
        printf("``Last'' has no sons to be brothers", [])&
        nl&
        father(X, L)&
        cut &
        hybrid_(S, parallel(T), X);
```

```
hybrid_(S, inference(T), L) <-
        not(rightmost_child(L, _))&
        printf("inference clue saves an oracle call", [])&
        nl&
        printf("``Last'' has no sons available to be re-attached", [])&
        nl&
        father(X, L)&
        cut &
        hybrid_(S, inference(T), X);
hybrid_(S, summary(T), L) <-
        not(rightmost_child(L, _))&
        printf("summary clue saves an oracle call", [])&
        nl&
        printf("``Last'' has no sons available to be re-attached", [])&
        nl&
        father(X, L)&
        cut &
        hybrid_(S, summary(T), X);
%
hybrid_(_, inference(T), dummy) <-
        no_sons_evidence(dummy, T)&
        printf("failed due to inference clue", [])&
        nl&
        printf("no possible sons to re-attach", [])&
        nl&
        printf("tree so far is:", [])&
        nl;
hybrid_(_, summary(T), dummy) <-
        no_sons_evidence(dummy, T)&
        printf("failed due to summary clue", [])&
        nl&
        printf("no possible sons to re-attach", [])&
        nl&
        printf("tree so far is:", [])&
        nl;
%
hybrid_(S, inference(T), L) <-
        evidence_oracle(T, L)&
        no_sons_evidence(L, T)&
        printf("inference clue causes continuation", [])&
        nl&
        father(X, L)&
        cut &
        hybrid_(S, inference(T), X);
```

```
hybrid_(S, summary(T), L) <-
        evidence_oracle(T, L)&
        no_sons_evidence(L, T)&
        printf("summary clue causes continuation", [])&
        nl&
        father(X, L)&
        cut &
        hybrid_(S, summary(T), X);

%  Eliminate the prefix of connective clues so that the standard
%  hybrid_ can be used to decide the claim-evidence relation between
%  two propositions.
%
hybrid_(S, detail(T), L) <-
        hybrid_(S, T, L);
hybrid_(S, parallel(T), L) <-
        hybrid_(S, T, L);
hybrid_(S, inference(T), L) <-
        hybrid_(S, T, L);
hybrid_(S, summary(T), L) <-
        hybrid_(S, T, L);

%  standard algorithm
%
hybrid_([], N, L) <-
        evidence_oracle(N, L)&
        no_sons_evidence(L, N)&
        assert_father(L, N);
hybrid_([], N, L) <-
        evidence_oracle(N, L)&
        attach_sons(L, N)&
        assert_father(L, N);
hybrid_([H | T], N, L) <-
        evidence_oracle(N, L)&
        no_sons_evidence(L, N)&
        assert_father(L, N)&
        hybrid_(T, H, N);
hybrid_([H | T], N, L) <-
        evidence_oracle(N, L)&
        attach_sons(L, N)&
        assert_father(L, N)&
        hybrid_(T, H, L);
```

```
hybrid_(S, N, L) <-
        not_evidence_oracle(N, L)&
        father(X, L)&
        hybrid_(S, N, X);

%  This succeedes only if the rightmost child of L is not evidence for
%  N, or if L has no children.
%
no_sons_evidence(L, _) <-
        not(father(L, _));
no_sons_evidence(L, N) <-
        rightmost_child(L, Son)&
        not_evidence_oracle(Son, N);

%  This predicate will print out the input agrument using the predicate
%  print_argument_.
%
print_argument(X) <-
        nl&
        printf("The argument is:\n", [])&
        print_argument_(X)&
        nl;

%  This predicate will print out the sentences in the argument.
%
print_argument_([]);
print_argument_([H | T]) <-
        write(H)&
        nl&
        print_argument_(T);

%  This predicate prints one child of a list, then prints the tree below
%  that child (at one more level of indentation), and then prints the
%  remaining children in the same way.
%
print_children([], _);
print_children([ H | T ], Depth) <-
        tab(Depth)&
        write(H)&
        nl&
        plus(Depth, 1, New_Depth)&
        print_tree(H, New_Depth)&
        print_children(T, Depth);
```

```
%  This predicate will print out the completed argument tree. It assumes
%  that the root (Root) has beed printed, finds all the children of the
%  root, and then calls the predicate print_children which prints out the
%  chidren, and the subtrees beneath them.  The predicate get_children is
%  specific to the routines for the type of analysis being done in order
%  to make sure that the list is returned in the correct order.
%
print_tree(Root, Depth) <-
        get_children(Root, Children)&
        print_children(Children, Depth);

%  This predicate will remove the rightmost child of L. If it has no
%  children, then it succeedes automatically. Otherwise the assertions
%  making the sentence the rightmost child of L are removed, and the
%  next to rightmost (if it exists) is made the rightmost.
%
remove_rightmost(L) <-
        not(rightmost_child(L, _));
remove_rightmost(L) <-
        rightmost_child(L, Son)&
        not(left_brother(_, Son))&
        delax(new, rightmost_child(L, Son), [])&
        delax(new, father(L, Son), []);
remove_rightmost(L) <-
        rightmost_child(L, Son)&
        left_brother(New_son, Son)&
        delax(new, left_brother(New_son, Son), [])&
        delax(new, rightmost_child(L, Son), [])&
        delax(new, father(L, Son), [])&
        assert(new, rightmost_child(L, New_son));
```

End of Our Program

5 Some Examples

The examples in this section are used to test our integrated program. They are organized into two parts, basically drawn respectively from [Young87] and [Smedley86, Smedley87]. Messages in the output have been edited slightly to make them more readable. For example, variables represented by _numbers will be replaced by capital letters. Some longer lines are cut down into several small lines. Finally some unimportant messages are eliminated as well.

5.1 Examples from [Young87]

Example 1 Same as example 5 in [Young87]. Suppose that the hearer (our stereotype for the speaker) has the rule:

```
francophone(X) <- french(X)
```

and the speaker's argument is:

```
(1) francophone( trudeau )          Trudeau is a francophone.

(2) french( trudeau )               He is French.
```

then we ask whether french(trudeau) is evidence for francophone(trudeau).

```
-----------------------------------------------------------------
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;

? add_beliefs( hearer, [for_all(francophone(X), [french(X)])] );

? oracle( francophone(trudeau), french(trudeau) );

french(trudeau) is evidence for
  francophone(trudeau) with
    [for_all(francophone(trudeau),[french(trudeau)])] missing.
  for_all(francophone(trudeau),[french(trudeau)]) has belief level 9
The conjunction has belief level Max:9, Sum:9
-----------------------------------------------------------------
```

This example is a simple one, only used to decide the claim-evidence relation between two propositions. The result is found to be the missing major premise of Modus Ponens. The appropriate rule does not appear in "facts", but does appear in the stereotype; it has, thus, a belief level of 9.

Example 2 Same as example 7 in [Young87]. Suppose the following generalizations appear in "missing":

```
adult(X) ∧ american(X) → owns_car(X);

adult(X) ∧ canadian(X) → owns_car(X);
```

Suppose further that the hearer believes Greg to be a Canadian. We need to decide the relation between the following two propositions:

```
(1) owns_car(greg)          Greg owns a car.

(2) adult(greg)             He is an adult.
```

```
-----------------------------------------------------------------
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;

? add_beliefs(missing, [for_most([X],owns_car(X),[adult(X),american(X)])]);

? add_beliefs(missing, [for_most([X],owns_car(X),[adult(X),canadian(X)])]);

? add_beliefs(hearer, [canadian(greg)]);

? oracle( owns_car(greg), adult(greg) );

adult(greg) is evidence for
  owns_car(greg) with
    [for_most([greg],owns_car(greg),[adult(greg),canadian(greg)]),
    canadian(greg)] missing.
  for_most([greg],owns_car(greg),[adult(greg),canadian(greg)]) has belief
    level 7
  canadian(greg) has belief level 9
The conjunction has belief level Max:9, Sum:16

adult(greg) is evidence for
  owns_car(greg) with
    [for_most([greg],owns_car(greg),[adult(greg),american(greg)]),
    american(greg)] missing.
  for_most([greg],owns_car(greg),[adult(greg),american(greg)]) has belief
    level 7
  american(greg) has belief level 10
The conjunction has belief level Max:10, Sum:17
-----------------------------------------------------------------
```

The oracle sometimes has some choices to make. In this case, the hearer believes Greg to be Canadian, and is using his own beliefs as a stereotype for the speaker, therefore assumes that the speaker also believes Greg to be Canadian. This process is decided by comparing the belief levels for both cases, and the one with smallest Max is chosen as the result.

Example 3a Same as example 10 in [Young87]. This argument is given in the claim-first order, which is required by Young's program.

```
(1) lnot(genius(X) <-  laughed_at(X));
                Not everyone who is laughed at is a genius.
(2) laughed_at(columbus);
                Sure, they laughed at Columbus;
(3) laughed_at(bozo_the_clown);
                but they also laughed at Bozo the Clown.
```

```
-------------------------------------------------------------------
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;

? add_beliefs(facts, [lnot(genius(bozo_the_clown))]);

? add_beliefs(hearer, [genius(columbus)]);

? parse([ lnot(for_all(genius(X), [laughed_at(X)])),
        laughed_at(columbus), laughed_at(bozo_the_clown) ]);

considering laughed_at(columbus) for lnot(for_all(genius(X),
    [laughed_at(X)]))

laughed_at(columbus) is evidence for
  lnot(for_all(genius(columbus),[laughed_at(columbus)])) with
    [lnot(genius(columbus))] missing.
  lnot(genius(columbus)) is not believed by the model

laughed_at(columbus) is evidence for
  lnot(for_all(genius(columbus),[laughed_at(columbus)])) with
    [genius(columbus)] missing.
  genius(columbus) has belief level 9
The conjunction has belief level Max:9, Sum:9
success laughed_at(columbus) for lnot(for_all(genius(X),
  [laughed_at(X)]))

considering laughed_at(bozo_the_clown) for laughed_at(columbus)
failure laughed_at(bozo_the_clown) for laughed_at(columbus)
```

```
considering laughed_at(bozo_the_clown)
    for lnot(for_all(genius(X),[laughed_at(X)]))

laughed_at(bozo_the_clown) is evidence for
  lnot(for_all(genius(bozo_the_clown),
    [laughed_at(bozo_the_clown)])) with
    [lnot(genius(bozo_the_clown))] missing.
  lnot(genius(bozo_the_clown)) has belief level 0
The conjunction has belief level Max:0, Sum:0

laughed_at(bozo_the_clown) is evidence for
  lnot(for_all(genius(bozo_the_clown),[laughed_at(bozo_the_clown)])) with
    [genius(bozo_the_clown)] missing.
  genius(bozo_the_clown) is not believed by the model
success laughed_at(bozo_the_clown) for
    lnot(for_all(genius(X),[laughed_at(X)]))

The argument tree is:

lnot(for_all(genius(X),[laughed_at(X)]))
        laughed_at(columbus)
        laughed_at(bozo_the_clown)


? print_beliefs( missing );

 module: missing

        lnot(genius(bozo_the_clown))
-----------------------------------------------------------------
```

The argument tree shows the final structure of the given argument. In each consideration, "success" confirms the claim-evidence relation between two propositions with the case having smallest Max or Sum as the result. The predicate "print_beliefs(M)" is used to show how modules have been updated. Note that in this example, some missing evidence is said "not believed by the model". This means that it contradicts with some existing beliefs in one of the four modules. Of course, this case should be eliminated from consideration.

Example 3b Similar to example 3a above, but this time, the argument is given in claim-last order and is decided by our integrated program.

```
-----------------------------------------------------------------
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;
```

```
? add_beliefs(facts, [lnot(genius(bozo_the_clown))]);

? add_beliefs(hearer, [genius(columbus)]);

? analyse([ laughed_at(columbus),
        laughed_at(bozo_the_clown),
        lnot(for_all(genius(X), [laughed_at(X)]))
        ]);

The argument is:
laughed_at(columbus)
laughed_at(bozo_the_clown)
lnot(for_all(genius(X),[laughed_at(X)]))

considering laughed_at(bozo_the_clown) for laughed_at(columbus)
failure laughed_at(bozo_the_clown) for laughed_at(columbus)

considering laughed_at(columbus) for laughed_at(bozo_the_clown)
failure laughed_at(columbus) for laughed_at(bozo_the_clown)

considering lnot(for_all(genius(X),[laughed_at(X)]))
    for laughed_at(bozo_the_clown)
failure lnot(for_all(genius(X),[laughed_at(X)]))
    for laughed_at(bozo_the_clown)

considering laughed_at(bozo_the_clown)
    for lnot(for_all(genius(X),[laughed_at(X)]))

laughed_at(bozo_the_clown) is evidence for
  lnot(for_all(genius(bozo_the_clown),[laughed_at(bozo_the_clown)])) with
    [lnot(genius(bozo_the_clown))] missing.
  lnot(genius(bozo_the_clown)) has belief level 0
The conjunction has belief level Max:0, Sum:0

laughed_at(bozo_the_clown) is evidence for
  lnot(for_all(genius(bozo_the_clown),[laughed_at(bozo_the_clown)])) with
    [genius(bozo_the_clown)] missing.
  genius(bozo_the_clown) is not believed by the model
success laughed_at(bozo_the_clown) for
    lnot(for_all(genius(X),[laughed_at(X)]))

considering laughed_at(columbus)
    for lnot(for_all(genius(X),[laughed_at(X)]))
```

```
laughed_at(columbus) is evidence for
  lnot(for_all(genius(columbus),[laughed_at(columbus)])) with
    [lnot(genius(columbus))] missing.
  lnot(genius(columbus)) is not believed by the model

laughed_at(columbus) is evidence for
  lnot(for_all(genius(columbus),[laughed_at(columbus)])) with
    [genius(columbus)] missing.
  genius(columbus) has belief level 9
The conjunction has belief level Max:9, Sum:9
success laughed_at(columbus) for
    lnot(for_all(genius(X),[laughed_at(X)]))


Argument tree:

lnot(for_all(genius(X),[laughed_at(X)]))
        laughed_at(columbus)
        laughed_at(bozo_the_clown)


? print_beliefs( missing );

 module: missing

        lnot(genius(bozo_the_clown))
-------------------------------------------------------------------
```

This time we use the predicate "analyse", which can deal with not only claim-first order, but also claim-last order and the hybrid order of both. The resulting argument tree is the same as above, but the intermediate outputs are different from the previous example.

Example 4a Same as example 13 in [Young87]. The main claim of this long argument is that Chretien will never lead the Liberals.

```
(1) lnot(leader(chretien, grits, T0));
            Chretien will never lead the grits.
(2) lnot(leader(X, P, T1)) <-
       lnot(english(X)), lnot(charismatic(X));
            No one can lead any party unless he is English
              or charismatic.
(3) lnot(english(wagner));
            Wagner, e.g.,was not English,
(4) lnot(leader(wagner, tories, T2));
            and so did not get to lead the Conservatives.
```

```
     (5) lnot(charismatic(clark));
                 Clark wasn't charismatic,
     (6) english(clark);
                 but he was English.
     (7) charismatic(trudeau);
                 Trudeau was charismatic.

-----------------------------------------------------------------
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;

? add_beliefs(hearer,
        [leader(mulroney, tories, t0),
         leader(turner, grits, t0),
         leader(trudeau, grits, t1),
         leader(mulroney, tories, t1),
         leader(clark, tories, t2),
         leader(trudeau, grits, t2),
         english(mulroney),
         english(turner),
         french(trudeau),
         english(clark) ] );

? parse( [ lnot( leader( chretien, grits, t0 ) ),
        for_all(lnot(leader(P,Q,R)),[lnot(english(P)),lnot(charismatic(P))]),
        lnot( english(wagner)),
        lnot( leader(wagner, tories, t1) ),
        lnot( charismatic(clark) ),
        english( clark ),
        charismatic( trudeau ) ] );

considering for_all(lnot(leader(P,Q,R)),[lnot(english(P)),
    lnot(charismatic(P))]) for lnot(leader(chretien,grits,t0))

for_all(lnot(leader(chretien,grits,t0)),[lnot(english(chretien)),
  lnot(charismatic(chretien))]) is evidence for
  lnot(leader(chretien,grits,t0)) with
    [lnot(english(chretien)),lnot(charismatic(chretien))] missing.
  lnot(english(chretien)) has belief level 10
  lnot(charismatic(chretien)) has belief level 10
The conjunction has belief level Max:10, Sum:20
success for_all(lnot(leader(P,Q,R)),[lnot(english(P)),
  lnot(charismatic(P))]) for lnot(leader(chretien,grits,t0))
```

```
considering lnot(english(wagner)) for for_all(lnot(leader(P,Q,R)),
    [lnot(english(P)),lnot(charismatic(P))])

lnot(english(wagner)) is evidence for
  for_all(lnot(leader(wagner,Q,R)),[lnot(english(wagner)),
    lnot(charismatic(wagner))]) with
    [lnot(leader(wagner,Q,R)),lnot(charismatic(wagner))] missing.
  lnot(leader(wagner,Q,R)) has belief level 10
  lnot(charismatic(wagner)) has belief level 10
The conjunction has belief level Max:10, Sum:20

lnot(english(wagner)) is evidence for
  for_all(lnot(leader(wagner,Q,R)),[lnot(english(wagner)),
    lnot(charismatic(wagner))]) with
    [charismatic(wagner),leader(wagner,Q,R)] missing.
  charismatic(wagner) has belief level 10
  leader(wagner,Q,R) has belief level 10
The conjunction has belief level Max:10, Sum:20
success lnot(english(wagner)) for for_all(lnot(leader(P,Q,R)),
  [lnot(english(P)),lnot(charismatic(P))])

considering lnot(leader(wagner,tories,t1)) for lnot(english(wagner))
failure lnot(leader(wagner,tories,t1)) for lnot(english(wagner))

considering lnot(leader(wagner,tories,t1)) for for_all(lnot(leader(P,Q,R)),
    [lnot(english(P)),lnot(charismatic(P))])

lnot(leader(wagner,tories,t1)) is evidence for
  for_all(lnot(leader(wagner,tories,t1)),[lnot(english(wagner)),
    lnot(charismatic(wagner))]) with
    [lnot(english(wagner)),lnot(charismatic(wagner))] missing.
  lnot(english(wagner)) has belief level 4
  lnot(charismatic(wagner)) has belief level 7
The conjunction has belief level Max:7, Sum:11
success lnot(leader(wagner,tories,t1)) for for_all(lnot(leader(P,Q,R)),
  [lnot(english(P)),lnot(charismatic(P))])

considering lnot(charismatic(clark)) for lnot(leader(wagner,tories,t1))
failure lnot(charismatic(clark)) for lnot(leader(wagner,tories,t1))

considering lnot(charismatic(clark)) for for_all(lnot(leader(P,Q,R)),
    [lnot(english(P)),lnot(charismatic(P))])
```

```
lnot(charismatic(clark)) is evidence for
  for_all(lnot(leader(clark,Q,R)),[lnot(english(clark)),
    lnot(charismatic(clark))]) with
    [lnot(leader(clark,Q,R)),lnot(english(clark))] missing.
  lnot(leader(clark,Q,R)) is not believed by the model

lnot(charismatic(clark)) is evidence for
  for_all(lnot(leader(clark,Q,R)),[lnot(english(clark)),
    lnot(charismatic(clark))]) with
    [english(clark),leader(clark,Q,R)] missing.
  english(clark) has belief level 9
  leader(clark,tories,t2) has belief level 9
The conjunction has belief level Max:9, Sum:18
success lnot(charismatic(clark)) for for_all(lnot(leader(P,Q,R)),
  [lnot(english(P)),lnot(charismatic(P))])

considering english(clark) for lnot(charismatic(clark))
failure english(clark) for lnot(charismatic(clark))

considering english(clark) for for_all(lnot(leader(P,Q,R)),
    [lnot(english(P)),lnot(charismatic(P))])

english(clark) is evidence for
  for_all(lnot(leader(clark,Q,R)),[lnot(english(clark)),
    lnot(charismatic(clark))]) with
    [leader(clark,Q,R),lnot(charismatic(clark))] missing.
  leader(clark,tories,t2) has belief level 9
  lnot(charismatic(clark)) has belief level 4
The conjunction has belief level Max:9, Sum:13
success english(clark) for for_all(lnot(leader(P,Q,R)),[lnot(english(P)),
  lnot(charismatic(P))])

considering charismatic(trudeau) for english(clark)
failure charismatic(trudeau) for english(clark)

considering charismatic(trudeau) for for_all(lnot(leader(P,Q,R)),
    [lnot(english(P)),lnot(charismatic(P))])

charismatic(trudeau) is evidence for
  for_all(lnot(leader(trudeau,Q,R)),[lnot(english(trudeau)),
    lnot(charismatic(trudeau))]) with
    [lnot(english(trudeau)),leader(trudeau,Q,R)] missing.
  lnot(english(trudeau)) has belief level 10
  leader(trudeau,grits,t1) has belief level 9
```

```
The conjunction has belief level Max:10, Sum:19
success charismatic(trudeau) for for_all(lnot(leader(P,Q,R)),
  [lnot(english(P)),lnot(charismatic(P))])

The argument tree is:

lnot(leader(chretien,grits,t0))
    for_all(lnot(leader(P,Q,R)),[lnot(english(P)),lnot(charismatic(P))])
        lnot(english(wagner))
        lnot(leader(wagner,tories,t1))
        lnot(charismatic(clark))
        english(clark)
        charismatic(trudeau)


? print_beliefs( missing );

 module: missing

        lnot(english(trudeau))
        lnot(charismatic(clark))
        lnot(english(wagner))
        lnot(leader(wagner,Q,R))
        lnot(charismatic(wagner))
        lnot(english(chretien))
        lnot(charismatic(chretien))
```

Example 4b Same as the above example, but the predicate "analyse " is used.

```
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;

? add_beliefs(hearer,
        [leader(mulroney, tories, t0),
         leader(turner, grits, t0),
         leader(trudeau, grits, t1),
         leader(mulroney, tories, t1),
         leader(clark, tories, t2),
         leader(trudeau, grits, t2),
         english(mulroney),
         english(turner),
         french(trudeau),
         english(clark) ] );
```

```
? analyse( [ lnot( leader( chretien, grits, t0 ) ),
        for_all(lnot(leader(P,Q,R)),[lnot(english(P)),lnot(charismatic(P))]),
        lnot( english(wagner)),
        lnot( leader(wagner, tories, t1) ),
        lnot( charismatic(clark) ),
        english( clark ),
        charismatic( trudeau ) ] );

The argument is:
lnot(leader(chretien,grits,t0))
for_all(lnot(leader(P,Q,R)),[lnot(english(P)),lnot(charismatic(P))])
lnot(english(wagner))
lnot(leader(wagner,tories,t1))
lnot(charismatic(clark))
english(clark)
charismatic(trudeau)

considering for_all(lnot(leader(P,Q,R)),[lnot(english(P)),
    lnot(charismatic(P))]) for lnot(leader(chretien,grits,t0))

for_all(lnot(leader(chretien,grits,t0)),[lnot(english(chretien)),
  lnot(charismatic(chretien))]) is evidence for
  lnot(leader(chretien,grits,t0)) with
    [lnot(english(chretien)),lnot(charismatic(chretien))] missing.
  lnot(english(chretien)) has belief level 10
  lnot(charismatic(chretien)) has belief level 10
The conjunction has belief level Max:10, Sum:20
success for_all(lnot(leader(P,Q,R)),[lnot(english(P)),
  lnot(charismatic(P))]) for lnot(leader(chretien,grits,t0))

considering lnot(english(wagner)) for for_all(lnot(leader(P,Q,R)),
    [lnot(english(P)),lnot(charismatic(P))])

lnot(english(wagner)) is evidence for
  for_all(lnot(leader(wagner,Q,R)),[lnot(english(wagner)),
    lnot(charismatic(wagner))]) with
    [lnot(leader(wagner,Q,R)),lnot(charismatic(wagner))] missing.
  lnot(leader(wagner,Q,R)) has belief level 10
  lnot(charismatic(wagner)) has belief level 10
The conjunction has belief level Max:10, Sum:20
```

```
lnot(english(wagner)) is evidence for
  for_all(lnot(leader(wagner,Q,R)),[lnot(english(wagner)),
    lnot(charismatic(wagner))]) with
    [charismatic(wagner),leader(wagner,Q,R)] missing.
  charismatic(wagner) has belief level 10
  leader(wagner,Q,R) has belief level 10
The conjunction has belief level Max:10, Sum:20
success lnot(english(wagner)) for for_all(lnot(leader(P,Q,R)),
  [lnot(english(P)),lnot(charismatic(P))])

considering lnot(leader(wagner,tories,t1)) for lnot(english(wagner))
failure lnot(leader(wagner,tories,t1)) for lnot(english(wagner))

considering lnot(leader(wagner,tories,t1)) for for_all(lnot(leader(P,Q,R)),
    [lnot(english(P)),lnot(charismatic(P))])

lnot(leader(wagner,tories,t1)) is evidence for
  for_all(lnot(leader(wagner,tories,t1)),[lnot(english(wagner)),
    lnot(charismatic(wagner))]) with
    [lnot(english(wagner)),lnot(charismatic(wagner))] missing.
  lnot(english(wagner)) has belief level 4
  lnot(charismatic(wagner)) has belief level 7
The conjunction has belief level Max:7, Sum:11
success lnot(leader(wagner,tories,t1)) for for_all(lnot(leader(P,Q,R)),
  [lnot(english(P)),lnot(charismatic(P))])

considering lnot(english(wagner)) for lnot(leader(wagner,tories,t1))

lnot(english(wagner)) is evidence for
  lnot(leader(wagner,tories,t1)) with
    [lnot(charismatic(wagner))] missing.
  lnot(charismatic(wagner)) has belief level 7
The conjunction has belief level Max:7, Sum:7
success lnot(english(wagner)) for lnot(leader(wagner,tories,t1))

considering lnot(charismatic(clark)) for for_all(lnot(leader(P,Q,R)),
    [lnot(english(P)),lnot(charismatic(P))])

lnot(charismatic(clark)) is evidence for
  for_all(lnot(leader(clark,Q,R)),[lnot(english(clark)),
    lnot(charismatic(clark))]) with
    [lnot(leader(clark,Q,R)),lnot(english(clark))] missing.
  lnot(leader(clark,Q,R)) is not believed by the model
```

```
lnot(charismatic(clark)) is evidence for
  for_all(lnot(leader(clark,Q,R)),[lnot(english(clark)),
    lnot(charismatic(clark))]) with
    [english(clark),leader(clark,Q,R)] missing.
  english(clark) has belief level 9
  leader(clark,tories,t2) has belief level 9
The conjunction has belief level Max:9, Sum:18
success lnot(charismatic(clark)) for for_all(lnot(leader(P,Q,R)),
  [lnot(english(P)),lnot(charismatic(P))])

considering lnot(leader(wagner,tories,t1)) for lnot(charismatic(clark))
failure lnot(leader(wagner,tories,t1)) for lnot(charismatic(clark))

considering english(clark) for lnot(charismatic(clark))
failure english(clark) for lnot(charismatic(clark))

considering english(clark) for for_all(lnot(leader(P,Q,R)),
    [lnot(english(P)),lnot(charismatic(P))])

english(clark) is evidence for
  for_all(lnot(leader(clark,Q,R)),[lnot(english(clark)),
    lnot(charismatic(clark))]) with
    [leader(clark,Q,R),lnot(charismatic(clark))] missing.
  leader(clark,tories,t2) has belief level 9
  lnot(charismatic(clark)) has belief level 4
The conjunction has belief level Max:9, Sum:13
success english(clark) for for_all(lnot(leader(P,Q,R)),[lnot(english(P)),
  lnot(charismatic(P))])

considering lnot(charismatic(clark)) for english(clark)

lnot(charismatic(clark)) is evidence for
  english(clark) with
    [leader(clark,Q,R)] missing.
  leader(clark,tories,t2) has belief level 9
The conjunction has belief level Max:9, Sum:9
success lnot(charismatic(clark)) for english(clark)

considering lnot(leader(wagner,tories,t1)) for english(clark)
failure lnot(leader(wagner,tories,t1)) for english(clark)

considering charismatic(trudeau) for for_all(lnot(leader(P,Q,R)),
    [lnot(english(P)),lnot(charismatic(P))])
```

```
charismatic(trudeau) is evidence for
  for_all(lnot(leader(trudeau,Q,R)),[lnot(english(trudeau)),
    lnot(charismatic(trudeau))]) with
    [lnot(english(trudeau)),leader(trudeau,Q,R)] missing.
  lnot(english(trudeau)) has belief level 10
  leader(trudeau,grits,t1) has belief level 9
The conjunction has belief level Max:10, Sum:19
success charismatic(trudeau) for for_all(lnot(leader(P,Q,R)),
  [lnot(english(P)),lnot(charismatic(P))])

considering english(clark) for charismatic(trudeau)
failure english(clark) for charismatic(trudeau)


Argument tree:

lnot(leader(chretien,grits,t0))
    for_all(lnot(leader(P,Q,R)),[lnot(english(P)),lnot(charismatic(P))])
        lnot(leader(wagner,tories,t1))
            lnot(english(wagner))
        english(clark)
            lnot(charismatic(clark))
        charismatic(trudeau)


? print_beliefs( missing );

 module: missing

        lnot(english(trudeau))
        lnot(charismatic(clark))
        lnot(english(wagner))
        lnot(leader(wagner,_11,_12))
        lnot(charismatic(wagner))
        lnot(english(chretien))
        lnot(charismatic(chretien))
-----------------------------------------------------------------
```

The final tree structure is slightly different from the previous example. The reason is that the predicate "analyse" treats an argument as one in the hybrid order in general, thus it can look back and further decide the relation between "lnot(leader(wagner, tories, t1))" and "lnot(english(wagner))", etc., while the predicate "parse" assumes that an argument is always given in claim-first order.

5.2 Examples from [Smedley86, Smedley87]

Example 5a Similar to example 1 for claim-first order in [Smedley86], but the argument is given in a list of predicates, rather than a list of lists of words in [Smedley86].

```
(1)  good_president(jones);
                Jones is a good president.
(2)  has_experience(jones);
                He has lots of experience.
(3)  tenyears_onboard(jones);
                He was on the board for ten years,
(4)  honest(jones);
                and he is honest.
(5)  refuse_bribes(jones);
                He refused bribes many times.
```

```
-----------------------------------------------------------------
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;

? add_beliefs(hearer, [
        for_all(good_president(X), [has_experience(X)]),
        for_all(good_president(X), [honest(X)]),
        for_most([X], has_experience(X), [tenyears_onboard(X)]),
        for_most([X], honest(X), [refuse_bribes(X)])
        ]);

? analyse([
        good_president(jones),
        has_experience(jones),
        tenyears_onboard(jones),
        honest(jones),
        refuse_bribes(jones)
        ]);

The argument is:
good_president(jones)
has_experience(jones)
tenyears_onboard(jones)
honest(jones)
refuse_bribes(jones)

considering has_experience(jones) for good_president(jones)
```

```
has_experience(jones) is evidence for
  good_president(jones) with
    [for_all(good_president(jones),[has_experience(jones)])] missing.
  for_all(good_president(jones),[has_experience(jones)]) has belief
    level 9
The conjunction has belief level Max:9, Sum:9
success has_experience(jones) for good_president(jones)

considering tenyears_onboard(jones) for has_experience(jones)

tenyears_onboard(jones) is evidence for
  has_experience(jones) with
    [for_most([jones],has_experience(jones),
      [tenyears_onboard(jones)])] missing.
  for_most([jones],has_experience(jones),[tenyears_onboard(jones)]) has
    belief level 9
The conjunction has belief level Max:9, Sum:9
success tenyears_onboard(jones) for has_experience(jones)

considering honest(jones) for tenyears_onboard(jones)
failure honest(jones) for tenyears_onboard(jones)

considering honest(jones) for has_experience(jones)
failure honest(jones) for has_experience(jones)

considering honest(jones) for good_president(jones)

honest(jones) is evidence for
  good_president(jones) with
    [for_all(good_president(jones),[honest(jones)])] missing.
  for_all(good_president(jones),[honest(jones)]) has belief level 9
The conjunction has belief level Max:9, Sum:9
success honest(jones) for good_president(jones)

considering has_experience(jones) for honest(jones)
failure has_experience(jones) for honest(jones)

considering refuse_bribes(jones) for honest(jones)

refuse_bribes(jones) is evidence for
  honest(jones) with
    [for_most([jones],honest(jones),[refuse_bribes(jones)])] missing.
  for_most([jones],honest(jones),[refuse_bribes(jones)]) has belief
    level 9
```

```
The conjunction has belief level Max:9, Sum:9
success refuse_bribes(jones) for honest(jones)


Argument tree:

good_president(jones)
        has_experience(jones)
                tenyears_onboard(jones)
        honest(jones)
                refuse_bribes(jones)
```

Example 5b Similar to example 5a, but is given in the claim-last order.

```
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;

? add_beliefs(hearer, [
        for_all(good_president(X), [has_experience(X)]),
        for_all(good_president(X), [honest(X)]),
        for_most([X], has_experience(X), [tenyears_onboard(X)]),
        for_most([X], honest(X), [refuse_bribes(X)])
        ]);

? analyse([
        tenyears_onboard(jones),
        has_experience(jones),
        refuse_bribes(jones),
        honest(jones),
        good_president(jones)
        ]);

The argument is:
tenyears_onboard(jones)
has_experience(jones)
refuse_bribes(jones)
honest(jones)
good_president(jones)

considering has_experience(jones) for tenyears_onboard(jones)
failure has_experience(jones) for tenyears_onboard(jones)

considering tenyears_onboard(jones) for has_experience(jones)
```

```
tenyears_onboard(jones) is evidence for
  has_experience(jones) with
    [for_most([jones],has_experience(jones),
      [tenyears_onboard(jones)])] missing.
  for_most([jones],has_experience(jones),[tenyears_onboard(jones)]) has
    belief level 9
The conjunction has belief level Max:9, Sum:9
success tenyears_onboard(jones) for has_experience(jones)

considering has_experience(jones) for refuse_bribes(jones)
failure has_experience(jones) for refuse_bribes(jones)

considering honest(jones) for refuse_bribes(jones)
failure honest(jones) for refuse_bribes(jones)

considering refuse_bribes(jones) for honest(jones)

refuse_bribes(jones) is evidence for
  honest(jones) with
    [for_most([jones],honest(jones),[refuse_bribes(jones)])] missing.
  for_most([jones],honest(jones),[refuse_bribes(jones)]) has belief
    level 9
The conjunction has belief level Max:9, Sum:9
success refuse_bribes(jones) for honest(jones)

considering has_experience(jones) for honest(jones)
failure has_experience(jones) for honest(jones)

considering honest(jones) for good_president(jones)

honest(jones) is evidence for
  good_president(jones) with
    [for_all(good_president(jones),[honest(jones)])] missing.
  for_all(good_president(jones),[honest(jones)]) has belief level 9
The conjunction has belief level Max:9, Sum:9
success honest(jones) for good_president(jones)

considering has_experience(jones) for good_president(jones)

has_experience(jones) is evidence for
  good_president(jones) with
    [for_all(good_president(jones),[has_experience(jones)])] missing.
```

```
  for_all(good_president(jones),[has_experience(jones)]) has belief
    level 9
The conjunction has belief level Max:9, Sum:9
success has_experience(jones) for good_president(jones)


Argument tree:

good_president(jones)
        has_experience(jones)
                tenyears_onboard(jones)
        honest(jones)
                refuse_bribes(jones)
-------------------------------------------------------------------
```

Example 5c Now we change the claim-last order in the above example into a hybrid order as follows.

```
-------------------------------------------------------------------
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;

? add_beliefs(hearer, [
        for_all(good_president(X), [has_experience(X)]),
        for_all(good_president(X), [honest(X)]),
        for_most([X], has_experience(X), [tenyears_onboard(X)]),
        for_most([X], honest(X), [refuse_bribes(X)])
        ]);

? analyse([
        has_experience(jones),
        tenyears_onboard(jones),
        honest(jones),
        refuse_bribes(jones),
        good_president(jones)
        ]);

The argument is:
has_experience(jones)
tenyears_onboard(jones)
honest(jones)
refuse_bribes(jones)
good_president(jones)

considering tenyears_onboard(jones) for has_experience(jones)
```

```
tenyears_onboard(jones) is evidence for
  has_experience(jones) with
    [for_most([jones],has_experience(jones),
      [tenyears_onboard(jones)])] missing.
  for_most([jones],has_experience(jones),[tenyears_onboard(jones)]) has
    belief level 9
The conjunction has belief level Max:9, Sum:9
success tenyears_onboard(jones) for has_experience(jones)

considering honest(jones) for tenyears_onboard(jones)
failure honest(jones) for tenyears_onboard(jones)

considering honest(jones) for has_experience(jones)
failure honest(jones) for has_experience(jones)

considering has_experience(jones) for honest(jones)
failure has_experience(jones) for honest(jones)

considering refuse_bribes(jones) for honest(jones)

refuse_bribes(jones) is evidence for
  honest(jones) with
    [for_most([jones],honest(jones),[refuse_bribes(jones)])] missing.
  for_most([jones],honest(jones),[refuse_bribes(jones)]) has belief
    level 9
The conjunction has belief level Max:9, Sum:9
success refuse_bribes(jones) for honest(jones)

considering good_president(jones) for refuse_bribes(jones)
failure good_president(jones) for refuse_bribes(jones)

considering good_president(jones) for honest(jones)
failure good_president(jones) for honest(jones)

considering honest(jones) for good_president(jones)

honest(jones) is evidence for
  good_president(jones) with
    [for_all(good_president(jones),[honest(jones)])] missing.
  for_all(good_president(jones),[honest(jones)]) has belief level 9
The conjunction has belief level Max:9, Sum:9
success honest(jones) for good_president(jones)
```

```
considering has_experience(jones) for good_president(jones)

has_experience(jones) is evidence for
  good_president(jones) with
    [for_all(good_president(jones),[has_experience(jones)])] missing.
  for_all(good_president(jones),[has_experience(jones)]) has belief
    level 9
The conjunction has belief level Max:9, Sum:9
success has_experience(jones) for good_president(jones)


Argument tree:

good_president(jones)
        has_experience(jones)
                tenyears_onboard(jones)
        honest(jones)
                refuse_bribes(jones)
-----------------------------------------------------------------
```

Example 5d Another hybrid order, along with minor changes to the starting set of module "hearer" in example 5c.

```
-----------------------------------------------------------------
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;

? add_beliefs(hearer, [
        for_all(good_president(X), [has_experience(X)]),
        for_all(good_president(X), [honest(X)]),
        for_all(has_experience(X), [tenyears_onboard(X)]),
        for_all(honest(X), [refuse_bribes(X)])
        ]);

? analyse([
        good_president(jones),
        tenyears_onboard(jones),
        has_experience(jones),
        refuse_bribes(jones),
        honest(jones)
        ]);

The argument is:
good_president(jones)
tenyears_onboard(jones)
```

```
has_experience(jones)
refuse_bribes(jones)
honest(jones)

considering tenyears_onboard(jones) for good_president(jones)

tenyears_onboard(jones) is evidence for
  good_president(jones) with
    [for_all(good_president(jones),[has_experience(jones)]),
    for_all(has_experience(jones),[tenyears_onboard(jones)])] missing.
  for_all(good_president(jones),[has_experience(jones)]) has belief
    level 9
  for_all(has_experience(jones),[tenyears_onboard(jones)]) has belief
    level 9
The conjunction has belief level Max:9, Sum:18
success tenyears_onboard(jones) for good_president(jones)

considering has_experience(jones) for tenyears_onboard(jones)
failure has_experience(jones) for tenyears_onboard(jones)

considering has_experience(jones) for good_president(jones)

has_experience(jones) is evidence for
  good_president(jones) with
    [for_all(good_president(jones),[has_experience(jones)])] missing.
  for_all(good_president(jones),[has_experience(jones)]) has belief
    level 9
The conjunction has belief level Max:9, Sum:9
success has_experience(jones) for good_president(jones)

considering tenyears_onboard(jones) for has_experience(jones)

tenyears_onboard(jones) is evidence for
  has_experience(jones) with
    [for_all(has_experience(jones),[tenyears_onboard(jones)])] missing.
  for_all(has_experience(jones),[tenyears_onboard(jones)]) has belief
    level 9
The conjunction has belief level Max:9, Sum:9
success tenyears_onboard(jones) for has_experience(jones)

considering refuse_bribes(jones) for good_president(jones)

refuse_bribes(jones) is evidence for
  good_president(jones) with
```

```
    [for_all(good_president(jones),[honest(jones)]),for_all(honest(jones),
      [refuse_bribes(jones)])] missing.
  for_all(good_president(jones),[honest(jones)]) has belief level 9
  for_all(honest(jones),[refuse_bribes(jones)]) has belief level 9
The conjunction has belief level Max:9, Sum:18
success refuse_bribes(jones) for good_president(jones)

considering has_experience(jones) for refuse_bribes(jones)
failure has_experience(jones) for refuse_bribes(jones)

considering honest(jones) for refuse_bribes(jones)
failure honest(jones) for refuse_bribes(jones)

considering honest(jones) for good_president(jones)

honest(jones) is evidence for
  good_president(jones) with
    [for_all(good_president(jones),[honest(jones)])] missing.
  for_all(good_president(jones),[honest(jones)]) has belief level 9
The conjunction has belief level Max:9, Sum:9
success honest(jones) for good_president(jones)

considering refuse_bribes(jones) for honest(jones)

refuse_bribes(jones) is evidence for
  honest(jones) with
    [for_all(honest(jones),[refuse_bribes(jones)])] missing.
  for_all(honest(jones),[refuse_bribes(jones)]) has belief level 9
The conjunction has belief level Max:9, Sum:9
success refuse_bribes(jones) for honest(jones)

considering has_experience(jones) for honest(jones)
failure has_experience(jones) for honest(jones)


Argument tree:

good_president(jones)
        has_experience(jones)
                tenyears_onboard(jones)
        honest(jones)
                refuse_bribes(jones)
-----------------------------------------------------------------
```

Note we have changed all generalizations into implications. This is because only implications are transitive, and this property is used to decide the relation between "good_president(jones)" and "tenyears_onboard(jones)", in which two rules of implications are applied. If we try this hybrid order with the previous example, we will get an incoherent result since the relation of those two predicates can not be decided there.

Example 6 Same as Example 2 in [Smedley87] and used to show the processing of a parallel clue.

```
(1) in_serious_trouble(city);
            The city is in serious trouble.
(2) going(fires);
            There are some fires going.
(3) broken_out(blazes);
            Three separate blazes have broken out.
(4) parallel(passing_through(tornado));
            In addition, a tornado is passing through.
```

```
-----------------------------------------------------------------
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
? setup;

? add_beliefs( hearer, [
        for_all(in_serious_trouble(city), [
          going(fires), passing_through(tornado) ]),
        for_all(going(fires), [broken_out(blazes)])
        ]);

? analyse([
        in_serious_trouble(city),
        going(fires),
        broken_out(blazes),
        parallel(passing_through(tornado))
        ]);

The argument is:
in_serious_trouble(city)
going(fires)
broken_out(blazes)
parallel(passing_through(tornado))

considering going(fires) for in_serious_trouble(city)

going(fires) is evidence for
  in_serious_trouble(city) with
```

```
    [for_all(in_serious_trouble(city),[going(fires),
    passing_through(tornado)]),passing_through(tornado)] missing.
  for_all(in_serious_trouble(city),[going(fires),
    passing_through(tornado)]) has belief level 9
  passing_through(tornado) has belief level 10
The conjunction has belief level Max:10, Sum:19
success going(fires) for in_serious_trouble(city)

considering broken_out(blazes) for going(fires)

broken_out(blazes) is evidence for
  going(fires) with
    [for_all(going(fires),[broken_out(blazes)])] missing.
  for_all(going(fires),[broken_out(blazes)]) has belief level 9
The conjunction has belief level Max:9, Sum:9
success broken_out(blazes) for going(fires)

parallel clue saves an oracle call
``Last'' has no sons to be brothers
considering passing_through(tornado) for going(fires)
failure passing_through(tornado) for going(fires)

considering passing_through(tornado) for in_serious_trouble(city)

passing_through(tornado) is evidence for
  in_serious_trouble(city) with
    [for_all(in_serious_trouble(city),[going(fires),
    passing_through(tornado)]),going(fires)] missing.
  for_all(in_serious_trouble(city),[going(fires),
    passing_through(tornado)]) has belief level 9
  going(fires) has belief level 4
The conjunction has belief level Max:9, Sum:13
success passing_through(tornado) for in_serious_trouble(city)

considering going(fires) for passing_through(tornado)
failure going(fires) for passing_through(tornado)

Argument tree:

in_serious_trouble(city)
        going(fires)
                broken_out(blazes)
        passing_through(tornado)
-----------------------------------------------------------------
```

Example 7 Same as example 3 in [Smedley87] and used to show an incoherent case, along with the use of an inference clue.

```
(1) messy(parks);
        The parks are a mess.
(2) messy(park_benches);
        The park benches are a mess.
(3) messy(playgrounds);
        The playgrounds are a mess.
(4) inference(messy(highways));
        As a result, the highways are a mess.
```

```
-----------------------------------------------------------------
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;

? add_beliefs( hearer, [
        for_all(messy(parks), [
          messy(park_benches), messy(playgrounds)
        ])  ]);

? analyse([
        messy(parks),
        messy(park_benches),
        messy(playgrounds),
        inference(messy(highways))
        ]);

The argument is:
messy(parks)
messy(park_benches)
messy(playgrounds)
inference(messy(highways))

considering messy(park_benches) for messy(parks)

messy(park_benches) is evidence for
  messy(parks) with
    [for_all(messy(parks),[messy(park_benches),messy(playgrounds)]),
    messy(playgrounds)] missing.
  for_all(messy(parks),[messy(park_benches),messy(playgrounds)]) has
    belief level 9
  messy(playgrounds) has belief level 10
The conjunction has belief level Max:10, Sum:19
```

```
success messy(park_benches) for messy(parks)

considering messy(playgrounds) for messy(park_benches)
failure messy(playgrounds) for messy(park_benches)

considering messy(playgrounds) for messy(parks)

messy(playgrounds) is evidence for
  messy(parks) with
    [for_all(messy(parks),[messy(park_benches),messy(playgrounds)]),
    messy(park_benches)] missing.
  for_all(messy(parks),[messy(park_benches),messy(playgrounds)]) has
    belief level 9
  messy(park_benches) has belief level 4
The conjunction has belief level Max:9, Sum:13
success messy(playgrounds) for messy(parks)

considering messy(park_benches) for messy(playgrounds)
failure messy(park_benches) for messy(playgrounds)

inference clue saves an oracle call
``Last'' has no sons available to be re-attached
considering messy(highways) for messy(parks)
failure messy(highways) for messy(parks)

considering messy(parks) for messy(highways)
failure messy(parks) for messy(highways)

no (more) answers

?list_all;
% Program (internal) database:

% Auxiliary database:
rightmost_child(nil,nil);
rightmost_child(messy(parks),messy(playgrounds));
rightmost_child(dummy,messy(highways));

left_brother(nil,nil);
left_brother(messy(park_benches),messy(playgrounds));
left_brother(messy(parks),messy(highways));

father([],[]);
father(dummy,messy(parks));
```

```
father(messy(parks),messy(park_benches));
father(messy(parks),messy(playgrounds));
father(dummy,messy(highways));

unasserted_evidence([],[]);
unasserted_evidence(messy(playgrounds),messy(park_benches));
unasserted_evidence(messy(park_benches),messy(playgrounds));
unasserted_evidence(messy(highways),messy(parks));
unasserted_evidence(messy(parks),messy(highways));

asserted_evidence([],[]);
asserted_evidence(A,dummy);
asserted_evidence(messy(park_benches),messy(parks));
asserted_evidence(messy(playgrounds),messy(parks));
-----------------------------------------------------------------
```

Here we can not get the argument tree structure since the argument is incoherent, but we can still observe the partial tree structure as intermediate results by using the built-in predicate "list_all" in WUP3.1.

Example 8a Same as example 4 in [Smedley87] and used to show the processing of both a parallel and a summary clues in the same argument.

```
(1) snow(weather, today);
        It snowed today.
(2) rain(weather, yesterday);
        It rained yesterday.
(3) parallel(hail(weather, tomorrow));
        And there is hail forecasted for tomorrow.
(4) summary(terrible(weather));
        In sum, the weather is terrible.

-----------------------------------------------------------------
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;

? add_beliefs(hearer, [
        for_all(terrible(weather), [
          snow(weather, T0), rain(weather, T1),
          hail(weather, T2) ]) ]);

? analyse([
        snow(weather,today),
        rain(weather,yesterday),
        parallel(hail(weather,tomorrow)),
        summary(terrible(weather)) ]);
```

```
The argument is:
snow(weather,today)
rain(weather,yesterday)
parallel(hail(weather,tomorrow))
summary(terrible(weather))

considering rain(weather,yesterday) for snow(weather,today)
failure rain(weather,yesterday) for snow(weather,today)

considering snow(weather,today) for rain(weather,yesterday)
failure snow(weather,today) for rain(weather,yesterday)

parallel clue saves an oracle call
``Last'' has no sons to be brothers
considering rain(weather,yesterday) for hail(weather,tomorrow)
failure rain(weather,yesterday) for hail(weather,tomorrow)

summary clue saves an oracle call
``Last'' has no sons available to be re-attached
considering hail(weather,tomorrow) for terrible(weather)

hail(weather,tomorrow) is evidence for
  terrible(weather) with
    [for_all(terrible(weather),[snow(weather,T0),rain(weather,T1),
    hail(weather,tomorrow)]),snow(weather,T0),rain(weather,T1)] missing.
  for_all(terrible(weather),[snow(weather,T0),rain(weather,T1),
    hail(weather,tomorrow)]) has belief level 9
  snow(weather,today) has belief level 4
  rain(weather,yesterday) has belief level 4
The conjunction has belief level Max:9, Sum:17
success hail(weather,tomorrow) for terrible(weather)

considering rain(weather,yesterday) for terrible(weather)

rain(weather,yesterday) is evidence for
  terrible(weather) with
    [for_all(terrible(weather),[snow(weather,T0),rain(weather,yesterday),
    hail(weather,T2)]),snow(weather,T0),hail(weather,T2)] missing.
  for_all(terrible(weather),[snow(weather,T0),rain(weather,yesterday),
    hail(weather,T2)]) has belief level 9
  snow(weather,today) has belief level 4
  hail(weather,tomorrow) has belief level 4
The conjunction has belief level Max:9, Sum:17
```

```
success rain(weather,yesterday) for terrible(weather)

considering snow(weather,today) for terrible(weather)

snow(weather,today) is evidence for
  terrible(weather) with
    [for_all(terrible(weather),[snow(weather,today),rain(weather,T1),
    hail(weather,T2)]),rain(weather,T1),hail(weather,T2)] missing.
  for_all(terrible(weather),[snow(weather,today),rain(weather,T1),
    hail(weather,T2)]) has belief level 9
  rain(weather,yesterday) has belief level 4
  hail(weather,tomorrow) has belief level 4
The conjunction has belief level Max:9, Sum:17
success snow(weather,today) for terrible(weather)


Argument tree:

terrible(weather)
        snow(weather,today)
        rain(weather,yesterday)
        hail(weather,tomorrow)
-----------------------------------------------------------------
```

Example 8b Similar to example 8a, but the rule in "hearer" is split into three separate rules.

```
-----------------------------------------------------------------
Waterloo Unix Prolog [Version 3.1a--Beta Release March 16, 1988]
?setup;

? add_beliefs(hearer, [
        for_all(terrible(weather), [
           snow(weather,T0)
        ]),
        for_all(terrible(weather), [
           rain(weather,T1)
        ]),
        for_all(terrible(weather), [
           hail(weather,T2)
        ]) ]);

? analyse([
        snow(weather,today),
        rain(weather,yesterday),
```

```
        parallel(hail(weather,tomorrow)),
        summary(terrible(weather))
        ]);

The argument is:
snow(weather,today)
rain(weather,yesterday)
parallel(hail(weather,tomorrow))
summary(terrible(weather))

considering rain(weather,yesterday) for snow(weather,today)
failure rain(weather,yesterday) for snow(weather,today)

considering snow(weather,today) for rain(weather,yesterday)
failure snow(weather,today) for rain(weather,yesterday)

parallel clue saves an oracle call
``Last'' has no sons to be brothers
considering rain(weather,yesterday) for hail(weather,tomorrow)
failure rain(weather,yesterday) for hail(weather,tomorrow)

summary clue saves an oracle call
``Last'' has no sons available to be re-attached
considering hail(weather,tomorrow) for terrible(weather)

hail(weather,tomorrow) is evidence for
  terrible(weather) with
    [for_all(terrible(weather),[hail(weather,tomorrow)])] missing.
  for_all(terrible(weather),[hail(weather,tomorrow)]) has belief level 9
The conjunction has belief level Max:9, Sum:9
success hail(weather,tomorrow) for terrible(weather)

considering rain(weather,yesterday) for terrible(weather)

rain(weather,yesterday) is evidence for
  terrible(weather) with
    [for_all(terrible(weather),[rain(weather,yesterday)])] missing.
  for_all(terrible(weather),[rain(weather,yesterday)]) has belief level 9
The conjunction has belief level Max:9, Sum:9
success rain(weather,yesterday) for terrible(weather)

considering snow(weather,today) for terrible(weather)
```

```
snow(weather,today) is evidence for
  terrible(weather) with
    [for_all(terrible(weather),[snow(weather,today)])] missing.
  for_all(terrible(weather),[snow(weather,today)]) has belief level 9
The conjunction has belief level Max:9, Sum:9
success snow(weather,today) for terrible(weather)


Argument tree:

terrible(weather)
        snow(weather,today)
        rain(weather,yesterday)
        hail(weather,tomorrow)
-----------------------------------------------------------------
```

References

[Cohen81] R. Cohen, "Investigation of Processing Strategies for the Structural Analysis of Arguments", Proceedings of ACL Conference, pp.71-75, (June, 1981).

[Cohen83] R. Cohen, "A Computational Model for the Analysis of Arguments", Univeristy of Toronto Technical Report CSRG-151,(Oct. 1983).

[Cohen84] R. Cohen, "A Theory of Discourse Coherence for Argument Understanding", Proceedings of CSCSI/SCEIO Conference, pp6-10(May, 1984).

[Cohen87] R. Cohen, "Interpreting Clues in Conjunction with Processing Restrictions in Arguments and Discourse", Proceedings AAAI-87, (July, 1987).

[Smedley86] T.J. Smedley, "An Implementation of a Computational Model for the Analysis of Arguments — An Introduction to the First Attempt", Technical Report CS-86-26, University of Waterloo, (1986).

[Smedley87] T.J. Smedley, "Integrating Connective Clue Processing into the Argument Analysis Algorithm Implementation", Technical Report CS-87-34, University of Waterloo, (1987).

[Young87] M.A. Young, "The Design and Implementation of an Evidence Oracle for the Understanding of Arguments", Technical Report CS-87-33, University of Waterloo, (1987).