ICOT Technical Report: TM-0052

TM-0052

Prologによる Belief System の実現方法

北上　始，国藤　進，古川康一，宮地泰造

April, 1984

ICOT

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

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Prolog によるBelief System の実現方法

北上　始，国藤進，古川康一，宮地泰造
（ ICOT 第２研究室 ）

［概要］
本報告では、Prologを使い、Belief System を最も自然な形でインプリメントする場合の実現方法についてのべる。著者らは、メタ推論機能が、それを実現する機能として十二分に満足のいく機能であると主張している。また、統合性制約に反する（矛盾する）知識についても、このメタ推論機能を使って、探すことができることを主張している。何故なら、メタ推論の実行中にゴールの証明プロセスを収集するという最も基礎的な技術を適用することができるからである。
なお、ここでは、知識を、オブジェクト知識とメタ知識の二種類に分類し、前者を、修正の対象としており、後者（メタ知識）およびユーザが与える修正のためのファクトを、正しい知識であるとしている。
オブジェクト知識およびメタ知識が、双方、正しい事を前提に、ユーザが登録したい入力知識を、同化するか否かの問題を解決する方法については、既に発表ずみのICOTの技術報告書を参照されたい。

１．はじめに

人間の知的問題解決能力を計算機システム上に実現するためには、その問題解決をするための知識が知識ベースに蓄積されていなければならない。知識ベースに蓄積されている知識には、変化し得る知識と変化しない知識の２種類があると考えることにする。ここでは、前者を、仮定（assume）型知識と呼び、後者を前提（premise)型知識と呼ぶ[1,16]。ここでは、信念を、仮定型知識で表現する。
オブジェクト知識を、ファクト（Facts)とルール（Rules)とし、それらに対するメタ知識の一つを、統合性制約(Integrity Constraints) とすると、メタ知識を真とする場合のオブジェクト知識の修正において、２種類の知識が変化する。それらは、ファクトとルールである。
ファクトの修正は、それを反転すれば良い[17]。ルールの修正は、メタ知識と矛盾が生じないような、仮説を生成し選択する事により達成される[7,16]。この仮説生成に関して、ユーザがある種の情報をもっている場合は、仮説生成のための探索空間を小さくし、妥当な仮説を早く見つける事ができる。
古くから議論されている信念システム(Belief System) は、TMS，[2,3] RMS[4] と呼ばれ、Lispでインプリメントされていたが、Prologでインプリメントすると、Data

Dependencyの扱いを、あまり意識する必要がなく自然に吸収する事ができる。即ち、ある証明をする際に、 Data Dependencyを使用し、前提と帰結の関係を利用する推論問題は、Prologのインタプリンタそのものを動かすことにより達成される。また、もしその証明問題を解くために、ある種の制御[5]又は、証明プロセスの記録取得を必要とするときは、Prologの処理系の上にdemo述語[6,7]と呼ばれる述語を作成すれば、容易にそれが可能になる。

本報告では、LispでインプリメントされていたBelief System を、上記解釈に基づきPrologでインプリメントした結果について述べる。また、本報告では、信念の修正を対象とする仮定型知識（修正されるオブジェクト知識）を、オブジェクト知識中のファクトと例外処理を前提とするルールの二つに限定する。信念の修正に際し、ファクトの反転操作は、正負の両方向が許されている。

一般のルールに対する修正については、Shapiro のMIS を改善する事で実現できる事を、著者らは、既に、報告済みである[8]。

2．オブジェクト知識とメタ知識の構造

ここでは、オブジェクト知識とメタ知識は、異なるフレームに蓄積されているとし、どの知識も、前提型知識あるいは、仮定型知識のどちらかに区別されているものとする。このフレームは、既に著者らが報告した知識ベース[8]であり、ある特定の知識を蓄積する枠組みである。したがって、本報告で扱う知識は、オブジェクト知識とメタ知識の区別がなく、以下の構造をしている。

フレームの名前（ホーン節，知識の種類）

・ホーン節　::= Prologの節形式
ただし、メタ知識と見做している統合性制約は、次の形をしている[8,18]。
inconsistent (Action__list, Concept): -Goals.

・知識の種類　::= { premise または assume }.

オブジェクトとメタの知識は、お互いに関連していなければならないが、両者を関連づけるリンクは、今のところ特に用意していないので、ここでは、ユーザがその関連ずけを実行時に（信念の修正時に）与えるものとする。

３．否定的知識としてのファクト

ここでは、否定的知識の一般的扱いが非常にむずかしいことから、否定的知識としては、単に述語名が肯定的知識と異なるという程度の解釈にとどめておく。例えば、canfly (ostrich)の否定知識は、 not_canfly (ostrich)であり、この形式以外の意味的な解釈は、何もないとする。前者は、“ダチョウは、飛べる”と読み、後者は、“ダチョウは、飛べない”と読む。

４．メタ推論

メタ推論の方式は、demo述語と呼ばれる述語により実現される事を、既に種々の形で報告しているので、それらを参照されたい[6,7]。ここでは、信念システムの本質を明らかにするために、オブジェクト知識を単純なものとし（論理和、カットオペレータ、その他の複雑な処理を省く）、できるだけ単純なdemo述語の上で、説明を試みる。

メタ知識との矛盾により、修正されるassume型のオブジェクト知識を、取得するために、その述語の第３引数に、d-listの方式を使用し、証明プロセスで情報取得をする機能を与えている。第４引数には、証明結果がtrue,exception_false で返される。通常のルールを前提としてゴールの証明をする場合、偽の証明通知は、demo述語そのものが fail する事により通知される。一方、例外値の扱いができるルールが証明の途中に、前提として使用される場合の偽の通知は、demo述語が、それ自身、 fail して通知するのではなく、そのdemo述語の第４引数に exception_false を返すことにより、通知される。

このdemo述語のプログラム例は、付録－１のプログラムリストを参照されたい。

５．信念としてのオブジェクト知識の修正

オブジェクト知識がファクトのときは、単にその否定的知識を作成し、修正対象の知識と置換すれば、信念の修正が達成される。ただし、修正の対象となる知識は、d-listにより取得[9,10,11]された知識(assume 型知識）である。例えば、canfly(ostrich) の修正後は、 not_canfly(ostrich) であり、 not_canfly(swallow) の修正後は、canfly(swallow) である。

オブジェクト知識がルールのときは、一般に精密化オペレータ[9]を適用すれば修正を実施できるが、ここでは、ルールの修正は、通常のルールと例外値処理をするルールとの間の変換であるとする。また、この例外値処理をするルールは、次の様な形式の節であるとする。

```
concept:- goals,except( フレームの名前 , not_concept ).
```

上記、concept に記述される述語名は、このルールの概念名である。このルールは、例外値処理をするために、concept に対する否定述語 not_concept がexcept述語の中に組み込まれている。以上から、信念の修正に当っては、次の様にルールを、修正すれば良いことが判る。即ち、ルールが例外値処理するルールでないときは、例外値処理するルールに変更するために、except述語を、そのルールのゴールに論理積で結合し、もとのルールをそれに置き換え、さらに例外知識を追加する。

例えば、既存知識が

```
{ bird (ostrich).
{ canfly(X):- bird(X).
```

であれば、

```
{ bird(ostrich).
{ canfly(X):- bird(X),except( フレームの名前,not_canfly(X) ).
{ not _canfly(ostrich).
```

と修正する。

ただし、そのフレームの名前は、except述語の中に現われている述語 not_canfly(X) に対するファクトが蓄積されているフレームの名前を指す。また、既にルール中に例外値処理をする述語がある場合は、単に例外知識（ファクト）だけを追加すれば良い。以上のように、知識の修正に際し、反転したファクトを、追加する処理は、修正知識がファクト、ルールを問わず必要である。

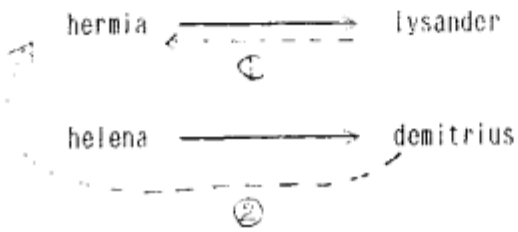
具体的なインプリメントは、付録－１のプログラムリストを参照されたい。

６．適用例

ここでは、２つの問題をとりあげ、その実行結果について説明する。１つは、Jon Doyle がとりあげたシェークスピアの物語で知られている「真夏の夜の夢」であり[3]、他の１つは、default reasoning[12,13,14]の話題で有名な、「飛べる鳥と飛べない鳥」の問題である。

（問題１）真夏の夜の夢

loves(X,Y)という述語をＸ →Ｙというグラフで表現すると、ここで登場する４人の人々の愛情関係は、次の様になっている。ただしassume型の知識は、点線で表現する。



この他に、“lysanderがジェラシーの持ち主である”、“ジェラシーをもつ人は他人を殺すことがある”、“愛情関係を持つ人は競合する相手を殺すことがある”、“hermiaを愛していない人は、helenaを愛している”、というルールが与えられている。

また、この物語は、人が１人でも死ねば悲劇になり、悲劇になる事をさけるための制約がメタ知識として他のフレームに蓄積されている。オブジェクト知識、メタ知識を蓄積するフレームは、各々、object1,meta_object1 と名ずけられている。

問題は、この場（オブジェクト知識が表現する場）の設定によると悲劇になるので、それを避けるためのオブジェクト知識の修正をいかにすべきかという事である。 付録－２に、その実行トレースを示す。

［実行結果］

信念の修正は、

equilibrate (オブジェクトのフレーム名、メタのフレーム名)．

の述語により実現されている。この結果、②のパスがhelenaに修正されることで、信念の修正を終了している。

（問題２）飛べる鳥と飛べない鳥

ここで現われる知識は、オブジェクト知識だけとし、この問題は、ユーザがファクトを与える事により達成されるものとする。すなわち、この問題は、与えられたファクトから、それを証明する新しいルールを作成する問題である。ただし、修正される知識としてのルールは、例外値処理をするルールにかぎり、取得されたルールは、Defalult Reasoningとして利用される。以下、付録－１，２を参照の事。

［実行結果］

最初、飛べるものは、

```
{ canfly(X) :-bird(X).
{ canfly(bat).
```

と定義されているため、ostrich とpenguin が飛べない事が、ある時点で判ると、それを表現するルールを修正しなければならなくなる。前述したように、exceptという述語がそのルールに組み込まれ、同時に not_canfly(ostrich), not_canfly(penguin) の２つのファクトがそのフレームに追加される。

７．まとめ

本報告では、インプリメンテーション言語Prologを使い、Belief System を最も自然な形式で実現する場合の実現方法を述べた。メタ推論は、その基礎として十二分に満足のいくものである。統合性制約と矛盾するオブジェクト世界の知識は、そのメタ推論の際に証明プロセスを収集するという最も基礎的な技術で取得できる。

なお、ここでは、オブジェクト知識だけを修正の対象としており、メタ知識及びユーザが与える修正のための入力ファクトを、正しい知識であるとしている。オブジェクトもメタも正しくて、ユーザが登録したい入力知識を受け入れるか否かの問題については、既に著者らによる発表済みの論文等を参照されたい[8,15]。

８．謝辞

本研究の機会を与えて下さった渕一博所長に深謝致します。

9．参考文献

[1] 鈴木　浩之：　論理の一貫性を重視した文脈処理、東京大学大学院工学系研究科修士論文　(1982).

[2] Jon Doyle:　Truth Maintenance System, Artificial Intelligence,　Vol. 12, No.3 (1979).

[3] Jon Doyle:　Truth Maintenance System for Problem Solving, MIT Technical report AI-TR-419 (1978).

[4] E. Charniak et al.: Artificial Intelligence Programming, Lawrence Erlbaum Associates (1980).

[5] James W. Goodwin: An Improved Algorithm for Non-monotonic Dependency Net Update, Linkoping Institute of Technology,Technical Report (1982).

[6] 国藤　進，麻生盛敏，竹内彰一，宮地泰造，北上　始，横田治夫，安川秀樹，古川康一：　Prologによる対象知識とメタ知識の融合とその応用，情報処理学会知識工学と人工知能研究会資料30-1，1983年 6月。

[7] 北上　始，麻生盛敏，国藤　進，宮地泰造，古川康一：知識同化機構の一実現法，情報処理学会知識工学と人工知能研究会資料 30-2，　1983年 6月。

[8] H. Kitakami, S. Kunifuji, T. Miyachi and　K. Furukawa : A Methodology for Implementation of A Knowledge Acquisition System, in Proc. of the 1984 International　Symposium on Logic Programming, Feb.　6-9 ,1984, New Jersey, also available from ICOT as ICOT Technical Report TR-037 (1983).

[9] E.Y. Shapiro: Algorithmic Program Debugging, An ACM Distinguishied Dessertation 1982, MIT Press.

[10] 田中　譲，堀内雄二，田川遼三郎：　推論システムとデータベースシステムとの部分評価機構による結合，計測自動制御学会，第１回知識工学シンポジウム資料 (1983).

[11] H. Yokota, S. Kunifuji, T. Kakuta, N. Miyazaki, S. Shibayama and K. Murakami: An Enhanced Inference Mechanism for Generating Relational Algebra Query, ICOT Technical Reprt TR-026 (1983)

[12] R. Reiter: A Logic for Defalult Reasoning, Artificial Intelligence (1980)

[13] 田中穂積 他： 自然言語処理技術と言語理論，電子総合研究所調査報告第 205号 (1981).

[14] 国藤 進，宮地泰造，北上 始，古川康一： 知識獲得とメタ推論，情報処理学会第27回全国大会，名古屋大学， 1983年10月。

[15] Miyachi, T., Kunifuji, S.,Kitakami, H., Furukawa, K., Takeuchi,A., and Yokota, H.: Knowledge Assimilation Method for Logic Databases, Proc. of 1984 International Symposium on Logic Programming, Feb 6-9, 1984, New Jersey.

[16] 白井英俊： ＲＭＳ（理由保持機構）を用いた知識表現システム，情報処理学会第27回全国大会 (1983).

[17] 新谷虎松： 知識ベース管理機構について（その２），情報処理学会第２７回全国大会 (1983).

[18] 北上 始，平川秀樹，古川康一： 知識獲得とＤＣＧ，情報処理学会第２８回全国大会 (1984).

付録－１

プログラムリスト

```
:-initialized->true;[library,timer].

/*-------------------------------+
|                               |
| (1) A Midsummer Night's Dream |
|                               |
+-------------------------------*/

/*---------------------+
|  Knowledge Base.    |
+---------------------*/

meta_object1(premise,(inconsistent(_,_):-tragedy)).

object1(premise,loves(hermia,lysander)).
object1(premise,loves(helena,demitrius)).
object1(assume,loves(demitrius,hermia)).
object1(assume,loves(lysander,hermia)).
object1(premise,jealous(lysander)).
object1(premise,(loves(X,helena):-not_loves(X,hermia))).
object1(premise,(kill(X,Z):-jealous(X),loves(X,Y),loves(Z,Y),X\==Z)).
object1(premise,(kill(X,X):-loves(X,Y),loves(Y,Z),X\==Z)).
object1(premise,(tragedy:-kill(X,Y))).


/*-----------------------+
|                        |
| (2) Default reasoning  |
|                        |
+-----------------------*/

/*---------------------+
|  Knowledge Base.    |
+---------------------*/

object2(premise,bird(ostrich)).
object2(premise,mammal(bat)).
object2(premise,bird(condor)).
object2(premise,bird(swallow)).
object2(premise,bird(penguin)).
object2(assume,(normally(X):-bird(X))).
object2(premise,normally(bat)).


/*-------------------------------------------------------+
|                  System Manager                        |
+-------------------------------------------------------*/

retrieve(KBNL,Goals):-
     open_timer,
     select_variable_list(Goals,X),
     setof_kb(KBNL,X,Goals,Set),
     subtract_tail(Set,Set1),
     write_list(Set1),nl,
     nl,close_timer.

equilibrate(Frame,Object):-
     open_timer,
     start_equilibrate(Frame,Object,[]).
equilibrate(Frame,Object):-
     nl,write('End Processing.'),nl,
     nl,close_timer.
```

```
start_equilibrate(Frame,Object,A):-
     next_clause(Frame,inconsistent(_,_),Goals,Degree),
     demo(Object,Goals,d(U,Y),true),Y=[],
     member(X,U),
     not_revised_already(Object,X,[Z|A]),
     delete_object(Object,X),
     add_reversed_fact(Object,X),
     nl,list(Object,X),
     next_time,
     start_equilibrate(Frame,Object,[Z|A]).

accommodate(Object,Concept):-
      open_timer,
      ( ask_for_code('# Default Reasoning(y/n) ? ',121)->true ;
        nl,write('# Not Implemented.'),!,fail ),
      mgt(Concept,Concept0),start_accommodation(Object,Concept0),
      nl,close_timer.

start_accommodation(Object,Concept):-
      ask_for_string('Next Fact(sentence,true/false) or end ? ',Fact)
      ( Fact=end ;
        ( (Fact=(P,V),mgt(P,P0),P0=Concept;
          nl,write('# Error Functor Name.'),fail ),!,
          (V=true;V=false)->
           (revising_process(Object,Concept,Fact),
           nl,nl,write('no error found.'),nl);
          write('!Illegal input'),nl ),!,
       start_accommodation(Object,Concept) ).

revising_process(Object,Concept,(P,V)):-
     ( V=true->(demo(Object,P,_,true)->true;
       add_fact(Object,P),nl,list(Object,Concept),next_time );
       (V=false->(demo(Object,P,d(X,Y),true)->
       (Y=[],revise_model(Object,P,X),nl,list(Object,Concept),
       next_time,
       nl,nl,write('Checking fact(s)...'),
      revising_process(Object,Concept,(P,V)));true)) ).

revise_model(Object,P,X):-
     member((P:-Goals),X),
     next_clause(Object,Head,Body,assume),
     verify(((Head:-Body)=(P:-Goals))),
     select_variable_list(Head,Z),
     (and_to_list(Body,List),
     ( \+member(except(W1,W2),List)->
       revise_rule(Object,P,(Head:-Body),Z);true ),
     add_reversed_fact(Object,P)).

revise_rule(Object,P,(Head:-Body),Z):-!,
      delete_object(Object,(Head:-Body)),
      (Z==[]->true;
      add_reversed_rule(Object,(Head:-Body))).

setof_kb(KBNL,X,Goals,Set):-
     setof([X,Y],demo(KBNL,Goals,Y,true),Set).

demo(Object,true,d(X,X),true):-!.
demo(Object,except(P),d([P|X],X),exception_false):-
     \+except(Object,P).
demo(Object,(P,Q),d(X,Z),Result):-
     demo(Object,P,d(X,Y),Result1),demo(Object,Q,d(Y,Z),Result2),
     (member(exception_false,[Result1,Result2])->
      Result=exception_false;Result=true).
demo(Object,P,d(X,Y),Result):-
```

```
            system(P)->(P,(P=..[except|_]->X=[P|Y];X=Y),Result=true);
            next_clause(Object,P,Q,Degree),
            (Degree==premise->X=Z;
             X=[(P:-Q)|Z]),demo(Object,Q,d(Z,Y),Result).

        next_clause(Object,Head,Goals,Degree):-
            X=..[Object,Degree,Clause],X,
            (Clause=(Head:-Goals)->true;
            Clause=Head,Goals=true).

system(_'\=='_).
system(_'=='_).
system(except(_,_)).
system('\+'_).

except(Object,P):- \+demo(Object,P,_,true).

subtract_tail([],[]).
subtract_tail([[X,Y]|Z],[X|W]):-subtract_tail(Z,W).

select_variable_list(Clause,Variable_list):-!,
     select_variable(Clause,[],Variable_list).
select_variable((P:-Q),Vs,Vf):-!,
     select_variable(P,Vs,Vi),select_variable(Q,Vi,Vf).
select_variable((P,Q),Vs,Vf):-!,
     select_variable(P,Vs,Vi),select_variable(Q,Vi,Vf).
select_variable((P;Q),Vs,Vf):-!,
     select_variable(P,Vs,Vi),select_variable(Q,Vi,Vf).
select_variable(P,Vs,Vf):-!,
     select_variable1(P,Vs,Vf).

select_variable1(P,Vs,Vs):-
     atomic(P),!.
select_variable1(P,Vs,Vf):-
     var(P),!,unique_variable(P,Vs,Vf).
select_variable1(P,Vs,Vf):-
     P=..[Predicate_name|Arguments],
     select_variable2(Arguments,Vs,Vf).

unique_variable(X,[],[X]).
unique_variable(X,[Y|Z],[Y|Z]):-
     X==Y,!.
unique_variable(X,[Y|Z],[Y|W]):-!,
     unique_variable(X,Z,W).

select_variable2([],Vs,Vs):-!.
select_variable2([A1|A2],Vs,Vf):-
     select_variable1(A1,Vs,Vi),
     select_variable2(A2,Vi,Vf).

write_list([]).
write_list([X|Y]):-
     nl,write(X),write_list(Y).

delete_object(Object,X):-
     (X=(H:-true)->true;X=H),
     Z=..[Object,assume,H],retract(Z),
     nl,write('Retract '),!,writev(Z).

add_reversed_fact(Object,X):-!,
     (X=(H:-true)->true;X=H),
     reverse_head(H,New_fact),
     New_Object=..[Object,assume,New_fact],
     (demo(Object,New_fact,_,true)->true;(assert(New_Object),
```

```
nl, fail ; true, nl.
```

付録－２

プログラムリスト

```
<<Problem 1>> A Midsummer Night's Dream.


| ?- [belief].

library consulted   3090 words      2.88 sec.

timer consulted   696 words      0.55 sec.

belief consulted   6980 words      6.75 sec.

yes
| ?- list_all(meta_object1).

  premise ... (inconsistent(X,Y):-tragedy)


yes
| ?- list_all(object1).

  premise ... loves(hermia,lysander)
  premise ... loves(helena,demitrius)
  assume .... loves(demitrius,hermia)
  assume .... loves(lysander,hermia)
  premise ... jealous(lysander)
  premise ... (loves(X,helena):-not_loves(X,hermia))
  premise ... (kill(X,Y):-jealous(X),loves(X,V),loves(Y,V),X\==Y)
  premise ... (kill(X,X):-loves(X,U),loves(U,W),X\==W)
  premise ... (tragedy:-kill(X,Y))


yes
| ?- retrieve(object1,loves(_,_)).

[demitrius,hermia]
[helena,demitrius]
[hermia,lysander]
[lysander,hermia]

0.120 sec.

yes
| ?- equilibrate(meta_object1,object1).

Retract object1(assume,loves(lysander,hermia))
Assert object1(assume,not_loves(lysander,hermia))

Listing of loves(X,Y)/not_loves(X,Y):

  premise ... loves(hermia,lysander)
  premise ... loves(helena,demitrius)
  assume .... loves(demitrius,hermia)
  premise ... (loves(X,helena):-not_loves(X,hermia))

  assume .... not_loves(lysander,hermia)

0.447 sec.

Retract object1(assume,not_loves(lysander,hermia))
Assert object1(assume,loves(lysander,hermia))

Listing of not_loves(X,Y)/loves(X,Y):
```

```
      nl,write('Assert '),writev(New_Object))).

add_fact(Object,X):-!,
     (X=(New_fact:-true)->true;X=New_fact),
     New_Object=..[Object,assume,New_fact],
     (demo(Object,New_fact,_,true)->true;(assert(New_Object),
      nl,write('Assert '),writev(New_Object))).

not_revised_already(Object,X,[Z|A]):-
     (X=(H:-true)->true;X=H),
     Z=..[Object,assume,H],!,\+member(Z,A).

reverse_head(X,Y):-!,
     X=..[P|Arg],
     name(P,S),name('not_',N),
     (include(N,S,S1)->true;append(N,S,S1)),
     name(Q,S1),Y=..[Q|Arg].

include([],Z,Z).
include([X|Y],[X|Z],W):-include(Y,Z,W).

add_reversed_rule(Object,(Head:-Body)):-
      reverse_head(Head,Np),
      P1=..[except,Object,Np],
      and_to_list(Body,List),
      append(List,[P1],List1),
      list_to_and(List1,Body1),
      New_rule=..[Object,assume,(Head:-Body1)],
      assert(New_rule),
      nl,write('Assert '),writev(New_rule).

verify(P):- \+(\+P).

ask_for_code(Message,Code):-
      nl,write(Message),ttyflush,!,
      ttyget0(Code),ttyskip(31).

ask_for_string(Message,String):-
      nl,write(Message),read(String).

list(Object,Concept) :-
     (Concept=(X:-true)->true;Concept=X),
     reverse_head(X,Nx),mgt(Nx,Nxx),
     ( X=(P:-_), !, mgt(P,P1) ; mgt(X,P1) ),
     nl,write('Listing of '),writev(P1),write('/'),writev(Nxx),
     write(':'), nl,nl,
     ( next_clause(Object,P1,Q,Degree1),
       tab(2),write(Degree1),
       (Degree1=premise->write(' ... ');write(' .... ')),
       ( Q\==true -> writev((P1:-Q)) ; writev(P1) ),
       nl, fail ; true ), nl,
     ( Nx=(Np:-_), !, mgt(Np,Np1) ; mgt(Nx,Np1) ),
     ( next_clause(Object,Np1,Nq,Degree2),
       tab(2),write(Degree2),
       (Degree2=premise->write(' ... ');write(' .... ')),
       ( Nq\==true -> writev((Np1:-Nq)) ; writev(Np1) ),
       nl, fail ; true ),!, nl.


 list_all(Object):-
      nl,next_clause(Object,P,Q,Degree),
      tab(2),write(Degree),
      (Degree=premise->write(' ... ');write(' .... ')),
      ( Q\==true -> writev((P:-Q)) ; writev(P) ),
```

```
  premise ... loves(hermia,lysander)
  premise ... loves(helena,demitrius)
  assume .... loves(demitrius,hermia)
  premise ... (loves(X,helena):-not_loves(X,hermia))
  assume .... loves(lysander,hermia)

0.530 sec.

Retract object1(assume,loves(demitrius,hermia))
Assert object1(assume,not_loves(demitrius,hermia))

Listing of loves(X,Y)/not_loves(X,Y):

  premise ... loves(hermia,lysander)
  premise ... loves(helena,demitrius)
  premise ... (loves(X,helena):-not_loves(X,hermia))
  assume .... loves(lysander,hermia)

  assume .... not_loves(demitrius,hermia)

0.497 sec.

End Processing.

2.333 sec.

yes
| ?- retrieve(object1,loves(_,_)).

[demitrius,helena]
[helena,demitrius]
[hermia,lysander]
[lysander,hermia]

0.131 sec.

yes
```

```
<<Problem 2>> Default Reasoning.

| ?- retrieve(object2,canfly(_)).

[bat]
[condor]
[ostrich]
[penguin]
[swallow]

0.94 sec.

yes
| ?- retrieve(object2,(bird(X),canfly(X))).

[condor]
[ostrich]
[penguin]
[swallow]

0.160 sec.

X = _38

yes
| ?- accommodate(object2,canfly(_)).

* Default Reasoning(y/n) ? y.

Next Fact(sentence,true/false) or end ? canfly(ostrich),false.

Retract object2(assume,(canfly(X):-bird(X)))
Assert object2(assume,(canfly(X):-bird(X),except(object2,not_canfly(X))))
Assert object2(assume,not_canfly(ostrich))

Listing of canfly(X)/not_canfly(X):

  premise ... canfly(bat)
  assume .... (canfly(X):-bird(X),except(object2,not_canfly(X)))

  assume .... not_canfly(ostrich)

0.542 sec.


Checking fact(s)...

no error found.

Next Fact(sentence,true/false) or end ? canfly(penguin),false.

Assert object2(assume,not_canfly(penguin))

Listing of canfly(X)/not_canfly(X):

  premise ... canfly(bat)
  assume .... (canfly(X):-bird(X),except(object2,not_canfly(X)))

  assume .... not_canfly(ostrich)
  assume .... not_canfly(penguin)

0.474 sec.
```

```
Checking fact(s)...

no error found.

Next Fact(sentence,true/false) or end ? end.

1.123 sec.

yes
| ?- retrieve(object2,canfly(_)).

[bat]
[condor]
[swallow]

0.156 sec.

yes
| ?- retrieve(object2,(bird(X),canfly(X))).

[condor]
[swallow]

0.242 sec.

X = _38

yes
| ?- retrieve(object2,not_canfly(_)).

[ostrich]
[penguin]

0.55 sec.

yes
```