ICOT Technical Report: TM-0064

TM-0064

知識の適応過程を支援する知識獲得システム

北上　始

June, 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

知識の適応過程を支援する知識獲得システム

1984/4/12 ICOT 第2研究室 北上 始

1．知識ベースの構造.

2．知識の適応過程 (Knowledge Adaptation Process).

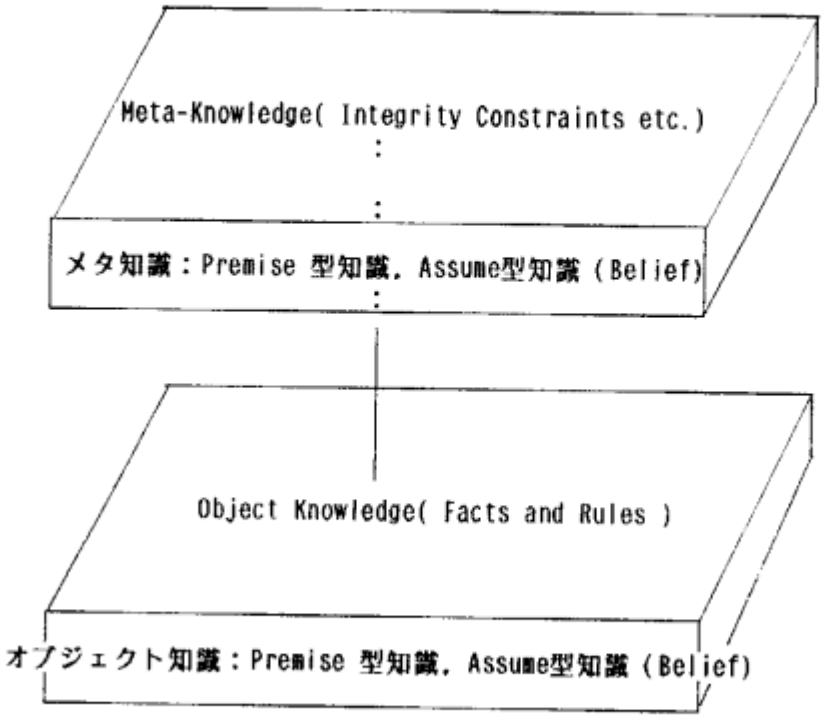
3．信念の修正 (Belief Revision).

4．知識獲得とDCG.

著者らが研究試作している知識獲得システムを知識適応プロセスにおいて、どのように利用するのかについて報告する。

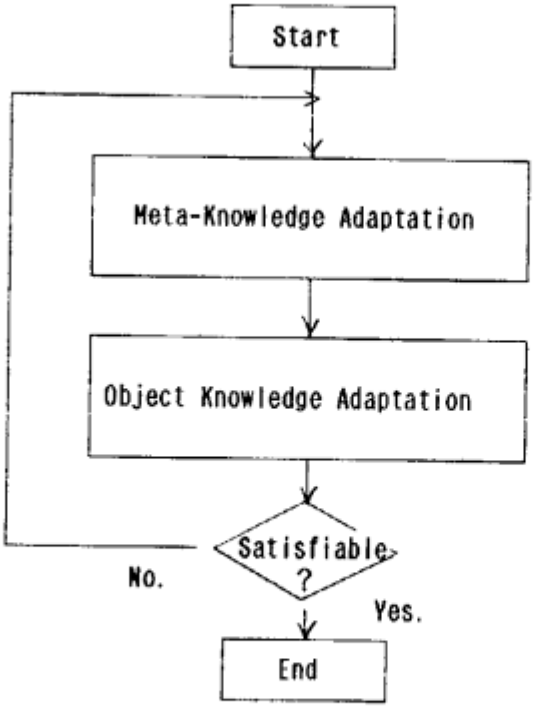
本報告で述べる知識適応プロセスは、認知心理学における人間（生態）と外界の環境間の相互作用をモデル化し、これを、第5世代コンピュータの知識ベースとエキスパート間の相互作用に写像した概念である。

1．知識ベースの構造

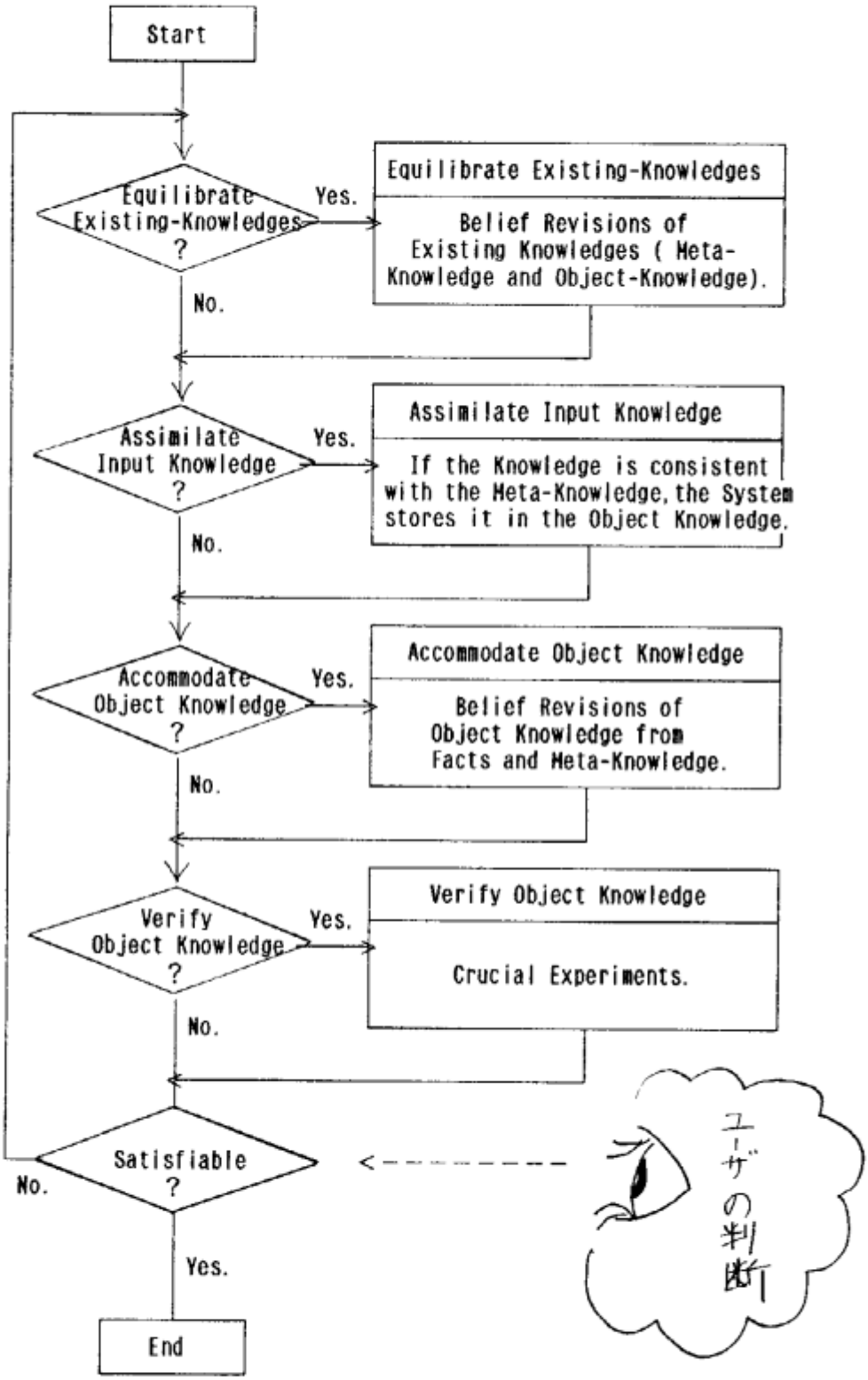


2．知識の適応過程(Knowledge Adaptation Process)

知識の適応過程のイメージ



知識の適応過程の概略フロー



知識の適応過程における、主要機能の説明

（１）Equilibrate Existing-Knowledge
この命令は、二つ以上の既存知識（Assume型の知識と Premise型の知識）群、の間の均衡化をはかるために、矛盾するAsumme型の既存知識を、修正する命令である。

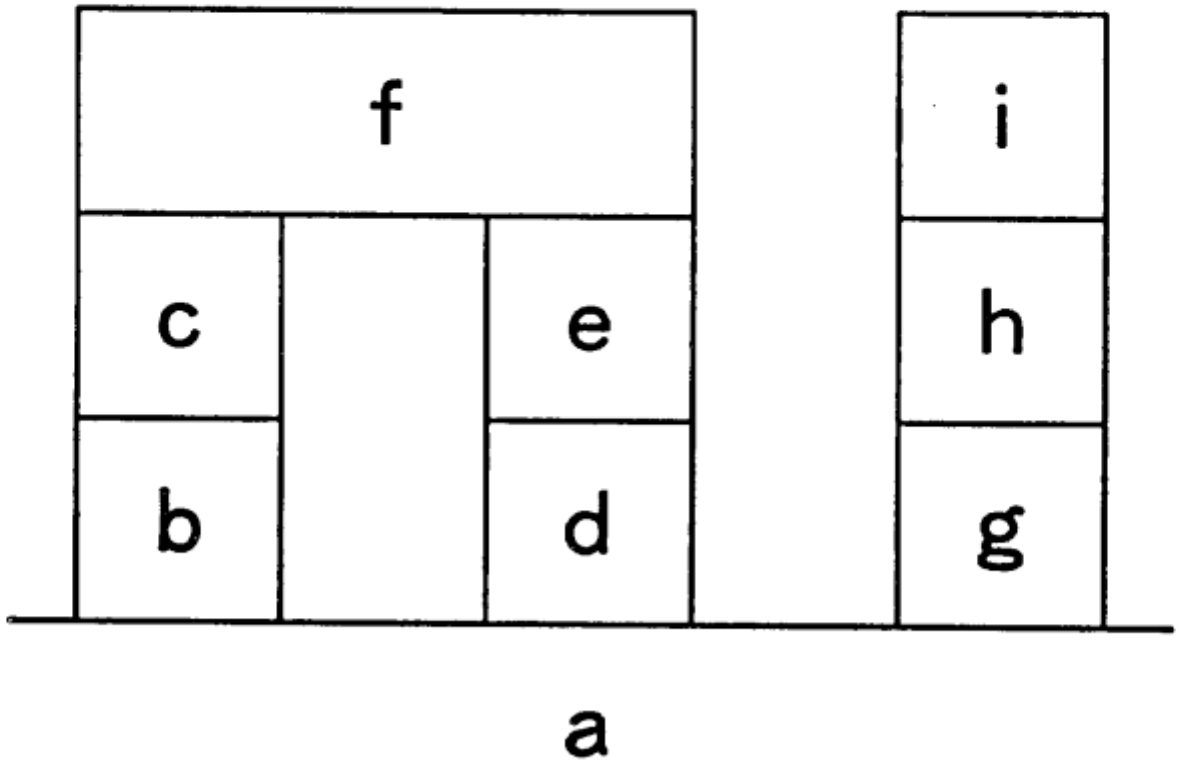
（２）Assimilate Input Knowledge
ユーザが、Object Knowledgeについて、ある程度、正確な入力知識をもっている場合、この命令は、その入力知識の中で、Meta-Knowledgeと矛盾しない知識を、Object Knowledgeとして蓄積する。また、もし、この入力知識の蓄積が、Object Knowlegde内に、冗長性を与えるならば、必要に応じて、その冗長性を、除去することができる。

（３）Accommodate Object Knowledge
ユーザが、複雑な既存知識から演繹される種々のインスタンスに関して、意図に反する結果を、いくつか演繹しているという意味で、不満をもっており、その既存知識を、いかに修正すべきかわからない場合に、利用される。ユーザは、その知識の修正のために正または負のインスタンスを与えなければならない。ユーザが与える正または負のインスタンスおよびMeta-Knowledgeを、正しい事を前提に、その既存知識が、修正される。このとき、Meta-Knowledgeと矛盾しないように、該当する知識が、修正される。

（４）Verify Object Knowledge
Object Knowledgeの内部状態を、演繹結果として得られたインスタンスまたは知識のパターンを知ることで確認する。これにより、既存知識がユーザの意図にあった知識であるか否かを、具体的に調べることができる。

◎ 次ページに積木の上下関係を示す概念(above(X,Y))の獲得について、
知識適応過程の例題をしめす。

Arrangement of Building Blocks



```
/* rectangular blocks */

floor(a).
rectangular_block(f).
rectangular_block(j).

/* square blocks */

square_block(b).
square_block(c).
square_block(d).
square_block(e).
square_block(g).
square_block(h).
square_block(i).
square_block(k).

/* block */

block(X):-
   square_block(X);
   rectangular_block(X);
   floor(X).

/* on(X,Y) : X is on Y */

on(b,a).
on(d,a).
on(c,b).
on(f,c).
on(e,d).
on(f,e).
on(g,a).
on(h,g).
on(i,h).

/*   tower(X,Y) : The tower consists of block X */
/*              on top of tower Y.              */

tower(X,Y):-tower1(X,Y), Y\==[].
tower1(X,[]):-floor(X).
tower1(X,[Y|Z]):-block(X),on(X,Y),tower1(Y,Z).
```

```
/* Integrity Constraints */

inconsistent([delete],floor(_)):- \+meta_demo(build,floor(a)).

inconsistent([delete,insert,update],on(_,_)):-
rectangular_block(X),
setof_kb(build,[Y],(tower(X,Y),ne(Y,[])),S), length(S,N),N<2.
```

The "setof_kb" and "meta_demo" predicates are defined below:

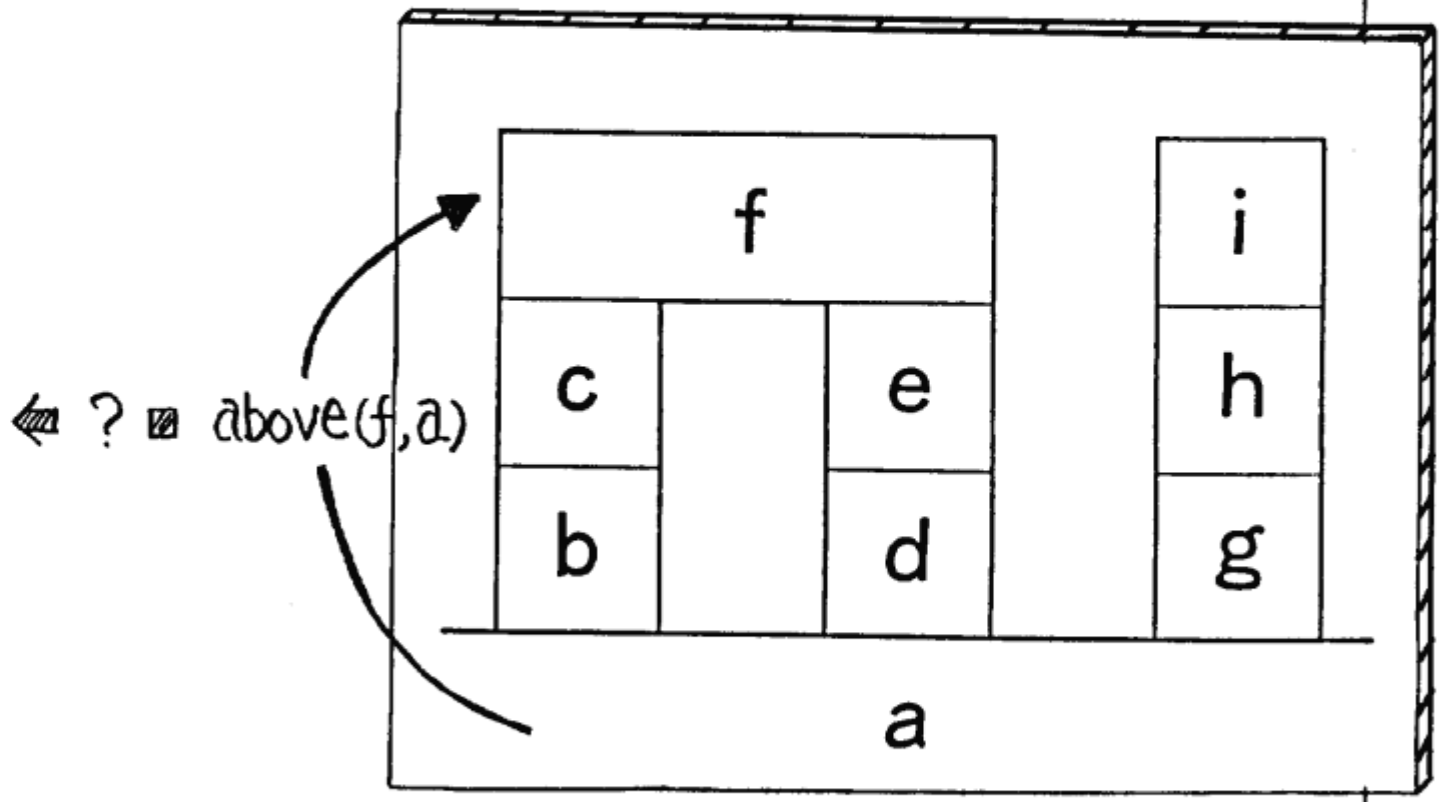
```
setof_kb(KBNL,X,Goals,Set):-setof(X,meta_demo(KBNL,Goals),Set).
meta_demo(KBNL,Goals):-demo(KBNL,Goals,[[],[],Cut,50],[true,[]]).
```

```
% ?- insert_knowledge(ic,
% ?- (inconsistent([insert],above(_,_)):- above(X,X)),_).

yes
% ?- list_all(ic).

(inconsistent([delete],floor(X)):- \+meta_demo(build,floor(a))).
(inconsistent([delete,insert,update],on(X,Y)):-
     rectangular_block(Z),
     setof_kb(build,[U],(tower(Z,U),ne(U,[])),Y1),length(Y1,U1),U1<2).
(inconsistent([insert],above(X,Y)):- above(Z,Z)).

yes
% ?- assimilate(build,ic,(above(X,Y):-on(X,Y))).

* Eliminate Redundancy ? (y/n) n.

* The Input_Knowledge "(above(_0,_1):-on(_0,_1))"  was assimilat

yes
% ?- assimilate(build,ic,(above(X,Y):-on(X,Z),on(Z,Y))).

... ? (y/n)
```

was as

yes
% ?- retrieve(build,above(X,Y),[X,Y]).

b,a
c,a
c,b
d,a
e,a
e,d
f,b
f,c
f,d
f,e
g,a
h,a
h,g
i,g
i,h



yes
% ?- accommodate(build,ic,above(_,_)).

* Initialize /n)? n.

...ize parameter(y) =

* The Knowledge_base "solutions" was ...

* The Knowledge_base "refuted_hypothesis" was created ...space.

Next fact(sentence,true/false) or end ? above(i,a),true.
Checking fact(s)...
Error: missing solution above(i,a). diagnosing...

Query: on(i,...)?

...g: (above(X,Y):-on(X,U),...
...ound clause: (above(X,Y):-on(X,U),...
after searching 20 clauses.
Listing of above(X,Y):
(above(X,Y):-on(X,Y)).
(above(X,Y):-on(X,U),on(U,Y)).
(above(X,Y):-on(X,U),above(U,Y)).

Checking fact(s)...no error found.

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

* Pure ...

...from Case ?

...ge_base solution

Purge refuted theory(y/n) ? y.

• The Knowledge_base "refuted_hypothesis" was dropped from Core_Space

• Purge refinement status(y/n) ? y.

• Eliminate Redundancy ? (y/n) y.

• The Knowledge "(above(_0,_1):-on(_0,_2),on(_2,_1))"
was eliminated from Knowledge_Base "build" .

• Construction of the Knowledge "above"
has been finished.

yes

```
% ?- list(build,above(X,Y)).

Listing of above(X,Y):
    (above(X,Y):-on(X,Y)).
    (above(X,Y):-on(X,U),above(U,Y)).

yes
% ?- retrieve(build,above(X,Y),[X,Y]).

[b,a]
[c,a]
[c,b]
[d,a]
[e,a]
[e,d]
[f,a]
[f,b]
[f,c]
[f,d]
[f,e]
[g,a]
[h,a]
[h,g]
[i,a]
[i,g]
[i,h]

yes
```

6G-5

知識獲得とDCG

北上 始, 平川秀樹, 古川康一
(財)新世代コンピュータ技術開発機構(ICOT)

1. はじめに

人間のもつ知識を組織的に蓄積、管理する知識ベースメカニズムの一機能として知識獲得の機能がある。知識獲得とは、以下の考察にもとづく知識の同化・調節・発見などによる知識の収集を意味する。

人間が外界から同化する知識は、観測事実としてのファクトの他に、ルールが挙げられる。ルールを同化する過程は、他者からの知識の伝授と捉える事ができる。

知識ベースに蓄積された知識にもとづく推論結果は、その時点で知識ベースがもっている信念であるととらえる事ができる。

人間の信念は、動として修正されるという側面をもっている。この信念の修正を実施するため、知識ベースに蓄積されている知識が修正される。修正される知識には、めったに修正される事のない知識と、流動的に変化する知識がある。

このように、知識ベースから演繹される信念が外界のモデルに一致しないとき、知識ベースの中に存在する知識が修正される。これは、知識ベースに対する調節と呼ばれている。 信念をささえる知識を修正する方法には、二つの方法が考えらる。一つは、新しい信念を説明できる仮説を生成し、妥当な仮説を既存知識とおきかえる方法であり、他の一つは既存知識を反転する方法である。知識の反転は、現在信じている知識を信じたくない知識に切り変えたり、信じたくなかった知識を、信ずる知識に切り変えたりする操作である。

知識獲得における知識の発見過程には、既存知識から、有益な概念を発見する過程が挙げられる。このレベルの発見には、特定の概念に関する学習や、類推などをともなう有益な概念の発見などがある。

本論文は、以上の議論の近似としてインプリメントされている知識獲得システム[1]を利用し、簡単な DCG (Definite Clause Grammar)の構文ルールを合成したので、その結果について報告する。以下、知識ベースの知識は、(i) ファクト(個々の事実)、(ii)ルール(一般的な性質)、(iii)Integrity Constraints(以降、ICs と呼ぶ)とする。ICs は、ファクト及びルールに関する統合性制約である。ICs は本システムに固有のメタ知識であり、DCG の構文ルールの合成に際し、会話の回数を低減する為に利用される。

2. DCG (Definite Clause Grammar)

本報告では、問題を単純化するため DCGの構文ルール[2,3]を合成する問題を、次の様に設定する。

(1) DCGの構文ルールを合成するために与える正の観測文には、単語辞書に登録されていない単語を含めない。もし、辞書にない単語を含めたいときは、辞書に登録後、使用する。

(2) 単語辞書に登録されている単語を、以下の単語に限定する。各単語は、構文ルールの合成に使われるとする。

```
n( [hermia | X],X).
n( [lysander | X],X).
vi( [walks | X],X).
vt( [loves | X],X).
```

ただし、n, vi, vt は、各々、名詞、自動詞、他動詞であり、すでに、知識ベース "dcg" に獲得されている。

(3) 構文ルールs(X,[]) は、 CFG (Context Free Grammar) を対象とし、単語辞書が格納されてる知識ベース "dcg" 内に獲得される。

(4) 知識ベース "dcg" に対するICs は、次の形をしており、すでに、知識ベース "ic" に獲得されているとする。

```
inconsistent ( [insert], s( _ ,_)) :-
                    s(X,[]), exist_variable(X).
```

このメタルールは、次の様な意味をもつ。

> (※) 構文ルールを " dcg" に挿入(insert)後、構文 s(X,[]) から演繹される文章 Xの中に変数を要素とする単語が存在するならば、その動作(insert)は、矛盾する。

また、このメタルールは、後の実行結果でもわかるが、DCG の構文ルールを合成する際に、非常に有効なルールであり、観測事実を与える回数を、極めて少なくするのに役立っている。さらに、このメタルールは、一般の文構成要素 Y(_ ,_) に容易に拡張が可能であり、全ての文法獲得時に適用できる。ここで、述語 " exist_variable" は、次の様に定義されている。

```
exist_variable( [X | Y]):- var(X),!.
exist_variable( [X | Y]):- !, exist_variable(Y).
exist_variable( []):- !, fail.
```

(5) DCG の構文ルールを合成する際に使用する観測事実、ICs 、組込み述語には、誤りがないと仮定する。また観測事実とICs の間には、矛盾がないものと仮定する。

3．実行結果

DCG の構文ルールを合成する命令は、

```
create(Knowledge-base-name, ICs-name, Concept)
```

で与えられる。ここでは、DCG の構文を、知識ベース内に新たに作成する意味をもたせるために、その命令を、"create" と命名した*。この命令を使い、次の構文ルールが合成された。

```
s(X,Y):-n(X,Z),vi(Z,Y).
s(X,Y):-n(X,Z),vt(Z,W),n(W,Y).
```

この合成問題は、次の２方式の、どちらでも解ける。

〈方式１〉 create(dcg,[],s(_, _)).

ここでは、ICs を使用せずに DCGの構文ルールを、合成するために、ICs-nameとして [] を、与える。この方式では、観測文として、以下の７つの事実を与えなければならないことが分った。

```
(1) s([hermia,walks],[]),true
(2) s([x,y],[]),false
(3) s(X,walks],[]),false
(4) s([hermia,loves,lysander],[]),true
(5) s([x,y,z],[]),false
(6) s([x,y,lysander],[]),false
(7) s([x,loves,lysander],[]),false
```

〈方式２〉 create(dcg,ic,s(_, _)).

この方式によると、観測文として、以下の２つの事実（正の事実）を与えるだけで、DCG の構文ルールを合成できることが分った。

```
(1) s([hermia,walks],[]),true
(2) s([hermia,loves,lysander],[]),true
```

本システムでは、合成された構文ルールを検証するために、 retrieve 命令が用意されているが、その命令形式は、次のようになる。

```
retrieve(Knowledge-base-name,Goals,Return-list).
```

この命令の実行トレースとして、次の結果が得られた。

```
X?- retrieve(dcg,s(X,[]),[X]).

[[hermia,loves,hermia]]
[[hermia,loves,lysander]]
[[hermia,walks]]
[[lysander,loves,lysander]]
[[lysander,loves,hermia]]
[[lysander,walks]]

yes
```

4．おわりに

知識獲得システムを使って、DCG の構文ルールについて、簡単な合成を行ってみたところ、ICs が観測事実をあたえる回数を低減するのに、非常に強力に働く事を確認した。また、ICs が、直感的にわかりやすい形式であたえられる事も示した。本研究の意義は、ICs と一般的な仮説生成スキーマを利用して、DCG の構文ルールを合成した事にある[4]。

5．今後の課題

本報告では、DCG の構文ルールを、Bottom Up に合成するには、良い見通しをあたえるが、Top Downの合成に関しては、何の解決策もあたえていない。ユーザ定義の述語が、不完全な形をしているときの合成アプローチは、今後の課題である。また、単なる CFGの合成ではなく、引数を含むより実際的な文法**の合成も、今後の課題である。

6．謝辞

本研究の機会をあたえて下さった ICOT の研究所長・渕一博氏、および有益な討論に参加していただいた ICOT 第２研究室・国藤進、宮崎敏彦、両氏に深謝致します。

7．参考文献

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

［２］長尾真：言語工学，昭晃堂　人工知能シリーズ２ (1983).

［３］中島秀之：Ｐｒｏｌｏｇ，産業図書　コンピュータサイエンス・ライブラリー (1983).

［４］E.Y.Shapiro : Algorithmic Program Debugging, An ACM Distinguished Dissertation 1982, The MIT Press.

---

* 合成対象の知識が、知識ベース内に一つでも存在するときは、命令 "accommodate " を利用する。この命令は、既存知識の修正に、利用される。また、"create" および、"accommodate " の命令は、共に、帰納推論および演繹推論などを、主要メカニズムとして実現されている。

** 多種の引き数があるが、その中で、構文の解析木を取得する機能をもつ引き数がある[3]。この機能は、メタ述語 "demo" [1] に、解析木（証明木）の取得機能をもたせることによっても実現できる。"demo" 述語に解析木の取得機能をもたせると、DCG の構文ルールにその機能をもつ引き数が不要になるので、DCG の構文ルールが簡単になる。