TR-147

知識表現形式 DCKR とその応用

田中穂積
(東京工業大学)

December, 1985

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

知識表現形式ＤＣＫＲとその応用

ＤＣＫＲ－－Knowledge Representation in Prolog and its Application

田中穂積

（東京工業大学工学部情報工学科）

１．はじめに

述語論理式で表現した知識と、フレームもしくはStructured Object で表現した知識との間の関係は［Hayes 80］, ［Nilsson 80］, ［Goebel 85］, ［Bowen 85］等により明らかにされている。［Nilsson 80］は、述語論理式で表現した知識を一度Structured Object 形式に変換し、それを意味ネットワーク形式で表現する方法をのべている。それによれば、Structured Object に関する推論は、それと等価な意味ネットワークをたどる操作に還元される。したがって、推論／問題解決をおこなうためには、別途この意味ネットワークに対するインタープリータを作る必要がある。

それに対して筆者らはDCKR(Definite Clause Knowledge Representation)とよぶ知識表現形式を開発している［小山　85］。DCKRはホーン節形式をベースにした知識表現形式であり、意味ネットワーク形式の知識をDCKRで容易に記述することができる。DCKRでは、Structured Object （以後これを単にObjectとよぶ）を構成する各スロットを、sem 述語（後述）をヘッドとする一つのホーン節（一つのProlog文）で表現する。したがって一つの Object は、第一引数が等しいsem 述語をヘッドとするホーン節（スロット）の集合としてみなすことができる。以上のことからDCKRで記述した知識に関する推論を行なうプログラムのほぼ全てを、Prologに組み込みの機能で代用することができることが分かる。推論を行なうための特別なプログラムを作る必要はないのである。

DCKRについては、第２章で詳しく述べる。第３章では、第２章で説明したナイーブなisa 述語の定義では、推論を行なう場合に効率が悪い理由を考察し、その改善策を述べる。第４章では、DCKRの応用として、談話理解で重要な役割を果たすSemantic Matcherのアルゴリズムと、自然言語の意味処理について述べる。まずはじめに、Semantic Matcherのアルゴリズムを示し、DCKRにより、その記述が極めて簡単になることを実例と共に示す。次に自然言語の意味処理を取り上げる。そこでは、辞書項目の記述をどのようにするかが問題になる。DCKRにより、辞書項目の柔軟な意味記述が可能になるとともに、それを利用した意味処理用のプログラムの中核が、Prologに組み込みの機能で代用可能なことを示す。また与えられた文の意味処理結果をDCKR形式にすることにより、質問応答で必要になる推論／問題解決を行なうプログラムを、Prologに組み込みの機能に任せることが可能なことを示す。

2．DCKRによる知識表現

2.1 Objectの表現と推論

第三章以下では、次に示すDCKRによる知識表現例を使う。

```
:-op(100,yfx,^),
  op(100,yfx,:),
  op(90,xfy,#).

01) sem(moore#1,adviser:mcCarthy#1,_).
02) sem(moore#1,P,S) :-
        isa(csStudent,P,[moore#1|S]).
03) sem(csStudent,P,S) :-
        isa(human,P,[csStudent|S]).
04) sem(human,P,S) :-
        isa(mammal,P,[human|S]).
05) sem(elephant,P,S) :-
        isa(mammal,P,[elephant|S]).
06) sem(mammal,bloodTemp:warm,_).
07) sem(mammal,P,S) :-
        isa(animal,P,[mammal|S]).
08) sem(animal,P,S) :-
        isa(creature,P,[animal|S]);
        hasa(body,P,[animal|S]).
09) sem(creature,age:X,S) :-
        bottomof(S,B),
        sem(B,birthYear:Y,_),
        X is 1985 - Y.
10) sem(clyde#1,age:5,_).
11) sem(clyde#1,P,S) :-
        isa(elephant,P,[clyde#1|S]).
12) sem(elephant#1,birthYear:1980,_).
13) sem(elephant#1,P,S) :-
        isa(elephant,P,[elephant#1|S]).
14) sem(elephant#2,birthYear:1982,_).
15) sem(elephant#2,P,S) :-
        isa(elephant,P,[elephant#2|S]).
16) sem(mcCarthy#1,address:stanford,_).
17) sem(mcCarthy#1,nationality:american,_).
18) sem(mcCarthy#1,
              affiliation:stanfordUniversity,_).
19) sem(mcCarthy#1,P,S) :-
        isa(human,P,[mcCarthy#1|S]).
```

```
20) sem(mister5G#1,address:japan,_).
21) sem(mister5G#1,P,S) :-
        isa(human,P,[mister5G#1|S]).
22) sem(misterAI#1,address:america,_).
23) sem(misterAI#1,P,S) :-
        isa(human,P,[misterAI#1|S]).
24) sem(america,capital:newYork,_).
25) sem(america,climate:temperate,_).
26) sem(america,P,S) :-
        T = [america|S],
        isa(coutry,P,T);
        hasa(california,P,T);
        ..............  .
27) sem(california,P,_) :-
        T = [california|S],
        isa(state,P,T);
        hasa(stanford,P,T);
        ..............  .
```

ここでDCKRの記述で重要になる sem述語、isa 述語、hasa述語の意味を、上記したDCKRによる記述例にしたがって説明する。

sem 述語の第一引数は、Object名である。Objectには大別して個体とプロトタイプがある。心理学者は、しばしばプロトタイプのことをstereotypeとよぶことがある。Object名のうち#の付いたものは個体名を、また#の付かないものはプロトタイプ名を表わす。たとえば01),02) に現われるmoore#1 は個体名であり csStudentはプロトタイプ名である。個体名が等しいsem 述語をヘッドとするホーン節の集合は、一つの個体を表わす。 たとえば、01)-02) は、moore#1 という個体を表わす。プロトタイプ名が等しいsem 述語をヘッドとするホーン節の集合は、一つのプロトタイプを表わす。 たとえば、06)-07) はmammalというプロトタイプを表わす。 ここで（ホーン節形式の）DCKRによるObjectの表現は、完全にコンパイルすることができる事に注意しよう。知識のコンパイルは高速化につながるので、この事はDCKRによる知識表現の好ましい性質であるといえる。

sem 述語の第二引数は、スロット名とスロット値の対である。 たとえば01）の記述は、個体moore#1 の adviserが mcCarthy#1である、という事実を表わす。そしてadviser がスロット名で、mcCarthy#1がスロット値である。スロット名とスロット値の対を以下ではＳＶ対とよぶ。したがって01）の

```
adviser:mcCarthy#1
```

はＳＶ対である。

02）の記述は、moore#1 がcsStudent （計算機科学科の学生）であると読む。ここで02）は上位からの知識の継承（inheritance of knowledge）をそのまま記述したものであることに注意されたい。 02)は、csStudent が性質P を持てばmoore#1 もその性質P を持つと読むのである。03)-04) はcsStudent が human であり、human がmammalであるという事実を記述している。知識の継承

はPrologに組み込みのユニフィケーションによって、自動的になされることにも注意されたい。　さて、07)-08) はmammalはanimalで、animalはcreatureでbodyを持っている、という事実を記述している。以上の説明から知識の継承で用いる述語isa は、Objectの階層をたどるための述語であることが理解できよう。

isa の定義をhasaとともに以下に示す。

```
28) isa(Upper,P,S) :-
        P = isa:Upper;
        sem(Upper,P,S).
29) hasa(Part,X:Y,S) :-
        X == hasa,
        (Y = Part;
         sem(Part,hasa:Y,S)).
```

再び02) の記述に戻って、これを別の観点から眺めると、02) は、個体の世界からプロトタイプの世界を呼び出し、プロトタイプが持っている情報を引き出すための記述であると見なすことができる。DCKRでは、isa 述語により一旦プロトタイプの世界に入れば、プロトタイプの世界に存在する全てのプロトタイプにアクセスすることができる。しかし、個体は動的に作られるものであるから、あらかじめ個体の世界の様子を知ることはできない。DCKRでは、プロトタイプの世界から個体の世界の様子を知る一つの手段として、09) のボディーで使われている述語bottomofが用意されている。

```
09.1) bottomof([Result|Y],Result) :-
              (var(Y);atomic(Y)),!,nonvar(Result).
09.2) bottomof([X|Y],Result) :-
              bottomof(Y,Result).
```

sem 述語の第三引数には、階層をたどった道筋がスタックされる。これをbottomof述語の第一引数に与えて、プロトタイプの呼び出し元（プロトタイプの世界を呼び出した個体）が何であるかを知ることができる。そして、プロトタイプの世界から、その個体が持っている知識を引き出す事ができる。たとえば　09) は、bottomof述語を用いて呼び出し元　B を知り、B のbirthYear を用いてage を計算する知識である。それにより、

```
?-sem(elephant#1,age:X,_).
```

を実行すると、isa 述語の働きによって09) に至り

```
X = 5
```

を得ることができる。また

```
?-sem(elephant#1,P,_).
```

を実行すると

```
P = birthYear:1980;
P = isa:elephant;
P = isa:mammal;
```

```
P = bloodTemp:warm;
P = isa:animal;
P = isa:creature;
P = age:5;
```

のようにelephant#1に関する知識を次々に得ることができる。知識継承によりelephant#1の上位にある知識（ＳＶ対；性質）が全て得られていることに注意されたい。

賢明な読者はお気付きのことと思うが、

```
?-sem(X,Y,_).
```

を実行すれば、システムは自分が知っているあらん限りの知識を、X と Yの対として語り始める。

Prologの特徴を生かして、

```
?- sem(X,isa:mammal,_).
```

を実行すれば、下位の個体またはプロトタイプを上位のmammalから、アクセスすることができる。だだし、これは必ずしも能率的に実行されない。なぜならこのゴールは、02）のような知識継承を行なうホーン節の全てのヘッドとユニファイ可能になるからである。その多くは最終的に失敗することになるから、知識継承を行なうホーン節の数が増えれば増えるほど計算コストがかかる。この問題は第三章で再び取り上げ、一解決法を与える。

最後に述語hasaの機能を調べるために、読者は

```
?- sem(america,hasa:X,_).
```

を実行して見るとよい。

これまでの説明から10）以降の記述がどのようなものであるか理解できるだろう。また推論を行なうプログラムが全く不要であることが分かるだろう。知識をDCKRで記述しておきさえすれば、推論はPrologに組み込まれたインタープリータによって自動的になされる。DCKRで記述した知識は読み易いと思われるがどうであろうか。このことは、知識の記述し易さにも通じる。

2.2 一般的な知識の表現と推論

2.1 節のDCKRによるObjectの記述例では、Objectは（第一引数をObject名とする）sem 述語をヘッドとするホーン節の集合として表現されていた。そしてそのObject名はいずれも（個体かプロトタイプを表わす）constantであった。これに対してsem 述語の第一引数が、（個体を表わす）variableである知識がDCKRではしばしば重要な役割を果たす。このようなvariableを以下では個体変数とよぶ。個体変数は一般にA#B のように表わされる。個体変数を第一引数として持つsem 述語をヘッドとするDCKR表現は、主として既存の知識から新たな知識を作り出す推論規則として働く。[Nilsson 80]の中から三つの例を取り上げ、DCKRでそれを記述し、その働きを調べてみよう。まず初めに、 "Everyone who lives at Maple Street is a programmer"という文に対応したDCKRの記述は次のようになる。

```
30) sem(X#J,profession:programmer,_) :-
            sem(X#J,isa:human,_),
            sem(X#J,loc:mapleStreet,_).
```

ここで X#Jは個体変数を表わす。さて、個体moore#1 には 01),02)の他に次の記述があるものとしよう。

```
31) sem(moore#1,loc:mapleStreet,_).
```

01),02),31) には、moore#1 のprofessionに関する記述はない。しかし、30)の推論規則によって、"moore#1のprofessionは何ですか" という質問文に対応する次のゴール

```
?-sem(moore#1,profession:A,_).
```

を実行することにより

```
A = programmer
```

を得ることができる。

[Nilsson 80] のFig.9.22は、 "csStudent (計算機科学科の学生)のadviser はcsFaculty である" 、という知識を表わしている。これはDCKRによって次のように表わすことができる。

```
32) sem(X#J,isa:csFaculty,_) :-
            sem(Y#K,isa:csStudent,_),
            sem(Y#K,adviser:X#J,_),
            X#J ¥== Y#K.
```

01)-32) の知識により次のゴール

```
?-sem(A#B,isa:csFaculty,_).
```

を実行することにより、

```
A = mcCarthy
B = 1
```

を得ることができる（読者はその理由を考えてみよ）。

[Nilsson 80]の9.4.6 節では、プロダクション規則の例が取り上げられている。それをDCKRで記述してみると次の33),34) のようになる。

```
33) sem(X#J,worksIn:Y#K,_) :-
            sem(Y#K,isa:department,_),
            sem(Y#K,manager:X#J,_).
34) sem(X#J,boss:Y#K,_) :-
            sem(X#J,worksIn:Z#L,_),
            sem(Z#L,manager:Y#K,_),
            X#J ¥== Y#K.
```

33) の知識（プロダクション規則）は、" Y#K が部門（department）で、そのマネージャ（manager ）がX#J なら、X#J はY#K で働いている（worksIn ）" ということを表わしている。34) の知識は、" X#J がZ#L で働いており、Z#L のマネージャがY#K で、X#J とY#K が同一人物でなければ、 X#J のボス(boss)はY#K である" ということを表わしている。33),34) の他に、更に次の

知識があるものとしよう。

```
35) sem(joeSmith#1,worksIn:pd#1,_).
36) sem(pd#1,manager:joeJones#1,_).
37) sem(pd#1,P,S) :-
            isa(department,P,[pd#1|S]).
```

この時、”pd#1で働いている人は誰ですか”という質問に対応する次のゴール

```
?-sem(A#B,worksIn:pd#1,_).
```

を実行することにより

```
A = joeSmith
B = 1;
A = joeJones
B = 1
```

を得ることができる。同様に、”joeSmith#1のボスは誰ですか”という質問に対応する次のゴール

```
?-sem(joeSmith#1,boss:A#B,_).
```

を実行することにより、

```
A = joeJones
B = 1
```

を得ることができる。この様にして、pd#1のマネージャがjoeJones#1であるという36）の知識から、joeJones#1がpd#1で働いており、彼はjoeSmith#1のボスである、という事実を引き出すことができる。

ここで、本節で述べたDCKRによる知識表現形式（ホーン節のヘッドにあるsem 述語の第一引数が個体変数）のために、知識の継承を行なう推論が非能率的になる理由を考察しておく。たとえば

```
?-sem(< 個体名>, P, _).
```

なるゴールは、個体の性質（P）を知るためにしばしば実行される。特に、（知識の継承のために階層をたどる）isa 述語の中でこのゴールがよく使われる。ところがこれは 30), 32), 33), 34) の全てのヘッドとユニファイ可能である。したがって、本節で述べたような推論規則の数が増えるにしたがって、階層をたどるたびに、無効な推論規則の適用回数が増大し、計算コストがかかることになる。この問題の解決は今後の課題である。

最後にDCKRの有効性を確認するために、読者は［Nilsson 80］の Fig.9.9と本節の 30)の記述、Fig.9.22と本節の 32)の記述、9.4.6 節のプロダクション規則と本節の 33),34)の記述とを見比べてみてほしい。それにより、DCKRの記述の分かり易さが理解できるとともに、DCKRに対する推論がPrologに組み込みの機能で代用されることの意味が良く分かると思われる。

３．DCKRによる推論の高速化

DCKRによる推論の高速化をはかる必要がなければ、02）のボディーに書かれたファンクターのisa を単にsem に変えても、知識継承を行なうことができる［小山　85］。この知識継承表現での問題は、下位から（isa を介さず）直

接上位のsem が呼ばれるため、バックトラックにより、知識継承に関する同一計算の繰り返しが阻止できないことである。このような再計算を阻止したり、一レベルだけ階層を遡る（4.1.2 参照）、などといった木目細かな制御を行なうためには、階層の上下を監視する機構が必要になる。isa 述語は、そのために設けられた。再計算の阻止は、Prologによるボトムアップ統語処理システムBUP ではじめて採用された方式が利用できる[Matsumoto 83]。以下では、それに基づいた述語 isa の定義を与える。 isa 述語では、成功パターンと失敗パターンを、それぞれwfisa,failisa としてassertする。assertした情報は、28.1) で持ち込まれるが、カット記号の存在により、28.2) の実行が回避され、再計算が防止される。その効果は、知識ベースが小さい場合より、知識ベースが大きい場合の方が大きく、推論が複雑になるにつれて高速化の効果が著しいことが観察される。数倍の高速化がはかれる。

```
28.1) isa(_,Prop,[HI_]) :-
              anchored(Prop),
              (wfisa(H,Prop),!,true;
               failisa(H,Prop),!,fail).
28.2) isa(Upper,Prop,[HIT]) :-
              (Prop == isa:Upper,!,true;
               Prop = isa:Upper;
               sem(Upper,Prop,[HIT])),
               wfassert(H,Prop).
28.3) isa(_,Prop,[HIT]) :-
              (var(T);atomic(T)),
              not(wfisa(H,Prop)),
              asserta(failisa(H,Prop)),!,fail.
28.4) wfassert(H,Prop) :-
              (wfisa(H,Prop);
               asserta(wfisa(H,Prop))),!.
28.5) anchored(X:Y#Z) :-
              !,atom(X),atom(Y),atomic(Z).
28.6) anchored(X:Y) :-
              !,atom(X),atomic(Y).
```

4．DCKRの自然言語処理への応用

DCKRの応用として、意味を考慮したパターンマッチのアルゴリズムと、自然言語の意味処理について説明する。またDCKRの有効性についても論じる。

4.1 Semantic Matcherの実現

4.1.1 Semantic Matching

人工知能の分野では様々な Object を扱わねばならない。この時 Object 相互間のパターンマッチを行なう必要がしばしば生じる。この方向を更におしすすめて、Object相互間のユニフィケーション（同一化）を基本計算機構に取り入れた言語 CILが開発されている［Mukai 85］。

Object 相互間の同一化を行なう場合の一つの問題は、Objectを構成するスロットの並びに関する順序的な制約がないため、スロットの並びの順番に依存しない同一化が必要になることである。そのため、Objectを一つの大きなデータ構造（たとえばリスト構造）として表現した時には、計算コストの問題が生じる。ところがDCKRを用いれば、スロットが一つのホーン節として表現されるため、同一化時に必要になるスロットの選択を、Prologに組み込みのバックトラック機能に任せることができるので、問題は若干軽減する。 それでも、一般にObject相互間の同一化を基本計算機構に組み入れた言語を設計する場合には、CIL のように同一化は、意味ではなくシンタクスのレベルに留めざるをえないだろう。

一方、意味を考慮したObject相互間の同一性の判定及び同一化を行なうことの重要性は、［Bobrow 77］は Forced Matching として、［Nilsson 80］は Semantic Matching として、またColmerauerはPrologのユニフィケーションを拡張する試みとして論じている。以下では、２章で述べた比較的単純な言語的な知識を背景にして、Object相互の同一化を、DCKRをベースにして行なう方法を述べる。この時、DCKRの形式で記述された知識の体系がどのようなものであるかを知る必要がある。そのため本節で説明するアルゴリズムでは、次のようなメタ知識を使う。

```
38) metakb(mammal,ntype:exclusive).
39) metakb(age,ltype:exclusive).
40) metakb(address,ltype:exclusive).
```

38）は、プロトタイプ mammal の直下にあるプロトタイプ相互（たとえば human とelephant）が両立しないことを表している。39）は ageのスロット値が、また40）はaddress のスロット値が異なれば、そのようなスロット相互は両立しないことを表している。 例をあげれば 38）は human である個体とelephantである個体とは同一化できないということ、また39）は、 ageが44才の個体と55才の個体とは同一化できないことを表している。

ここでSemantic Matching の例を幾つか挙げる。我々は、２章のDCKRの記述から容易に次のi)-iv)のような推論を行なうことができる。

i) mcCarthy#1のaddress はstanfordで、misterAI#1のaddress はamerica であることが分かる。我々は、stanfordがamerica のなかにあるという事実を知っているから、misterAI#1とmcCarthy#1とを同一視し、同一化してもなんら矛盾はないと推論することができる。
ii) 同様にして、america とjapan は異なる国であるから、mister5G#1とmcCarthy#1とを同一化できないと推論することができる。
iii)現在の年号から生まれた年の年号を引いたものが年齢であるとすれば、年齢が5 才のclyde#1 と生まれた年が1980年の elephant#1 とを同一化しても良いと推論することができる（現在が1985年だと仮定する）。
iv) 同様な推論を行なって、clyde#1 とelephant#2とを同一化することはできないと結論することができる。
v) mcCarthy#1とclyde#1 とを同一化できない。なぜなら、前者は人間(human)であり後者は象(elephant)だからである。

このように、意味を知り二つのObjectの同一化を行なう操作を Semantic Matching (Forced Matching)とよび、Semantic Matching を行なうプログラムをSemantic Matcherとよぶ。

4.1.2 Semantic Matcherのアルゴリズム

これまで、このようなSemantic Matcherの必要性はしばしば論じられてきたが、Semantic Matcherを作成したという試みはさほど多くない。それは、上記した程度のSemantic Matcherでさえ、相当複雑なものになると考えられるからである。しかし、DCKRによる知識表現を用いれば、その作成は比較的容易である。完全なものとはいえないが、我々は次の[A] から [F]に示すアルゴリズムによって、i)からv)に示した程度の同一化能力を持つSemantic Matcherを作ることができる。

[A] : 同一化の対象になる二つの個体 o#1, o#2 に共通する上位の Object (Oi) があればそれを取り出し[B] へ、さもなければ[D] へ。
[B] : metakb(Oi,ntype:exclusive)が成り立つ（Oiの直接１レベル下位のObjectが互いにexclusive な関係にある）ならば[C] へ、さもなければ[A] へ。
[C] : Oiの直接１レベル下位に相異なる二つの Object Oj,Ok があり、個体o#1とo#2 の上位にそれぞれOjとOkが位置しているなら、同一化は失敗であるとしてリターン、さもなければ[A] へ。
[D] : o#1 にＳＶ対 Ax:Bx が、 o#2にＳＶ対 Ax:By があれば、以下（次ページ）に示す集合Ｓを作り[E] へ（スロット名が共通なことに注意）、さもなければ o#1と o#2とを同一化してはならないという積極的な理由がないので、同一化が可能であるとして [F]へ。

```
S = {(Ax:Bx, Ax:By) | metakb(Ax,ltype:exclusive) &
                      (Bx == By  or
                       sem(Bx,isa:By,_)  or
                       sem(By,isa:Bx,_)  or
                       sem(Bx,hasa:By,_)  or
                       sem(By,hasa:Bx,_))  }
```

[E] : Sが空集合でなければ同一化が可能であるとして[F] へ、空集合なら同一化は失敗であるとしてリターン。

[F] : o#1 とo#2 とを等しいとおくために（同一化するために）、次の事実をassertする。

```
sem(o#1,P,S) :-
    isa(o#2,P,[o#1|S]).
sem(o#2,O,S) :-
    isa(o#1,O,[o#2|S]).
```

以上によりo#1 にはo#2 の、またo#2 にはo#1 の性質が自動的にうけつがれo#1 とo#2 とは同一化される。

[A] から[F] のアルゴリズムは、以下に示すmkeq述語として与えられる。ただしmkeq述語の第三引数が "?"マークなら、同一化可能性のみを調べ、[F] のassertは行なわない。

```
mkeq(X,Y) :- mkeq(X,Y,_),!,abolish(unknownisa,2).
mkeq(X,Y) :- abolish(unknownisa,2),!,fail.

mkeq(X,X,_) :- !.
mkeq(X,Y,_) :-
        sem(X,isa:Y,_),
        sem(Y,isa:X,_),!.
mkeq(X,Y,_) :-
        % checks exclusive relations in isa relations.
        indiv(X),indiv(Y),
        % get a common node Z which is marked as 'exclusive.'
        sem(X,isa:Z,_),
        metakb(Z,ntype:exclusive),
        sem(Y,isa:Z,_),
        % get a node A and B that is one level lower than Z.
        sem(A,isa:Z,1),
        not(indiv(A)),
        sem(X,isa:A,_),
        sem(B,isa:Z,1),
        not(indiv(B)),
        A ¥== B,
```

```
        sem(Y,isa:B,_),
        !,fail.
mkeq(X,Y,_) :-
        indiv(X),indiv(Y),
        sem(X,Ax:Bx,_),
        Ax ¥== isa, %isa relations have already checked above.
        eqsem(Ax:Bx,Y,State),
        ((State==fail,!,fail);
        fail).
mkeq(X,Y,Mk) :-
        indiv(X),indiv(Y),
        mklink(X,Y,Mk),
        mklink(Y,X,Mk).
eqsem(Ax:Bx,Y,State) :-
        metakb(Ax,ltype:exclusive),
        sem(Y,Ax:By,_),
        (Bx == By;
          % (Bx==dog amd By==pluto)
         sem(Bx,isa:By,_);
         sem(By,isa:Bx,_);
          % (By==tokyo and By==Japan)
         sem(Bx,hasa:By,_);
         sem(By,hasa:Bx,_);
         State=fail),!.

mklink(_,_,MK) :-
        MK == ?,!
mkink(X,Y,_) :-
        assertz((sem(X,Prop,S) :-
                        isa(Y,Prop,[X|S]))).

indiv(X) :-
        nonvar(X),Y#_ = X,nonvar(Y).
```

4.1.2 isa 述語の定義の修正

前節で述べた同一化のための操作 [F]のために、３章で述べたisa 述語の定義を修正する必要がある。なぜなら前節の操作[F] により、階層をたどる道筋がループするため、そのチェックを行う必要があるからである。さらに、ループが発見されfailでリターンする場合には、後で別の道筋をたどり、証明しようとする事柄が証明可能となることが考えられるから、真のfailではないことになる。したがって、isa 述語の定義のボディで実行されるsem 述語に対して、それが真に証明不可能でfailしてリターンする場合と、ループのチェックによりfailしてリターンする場合とを明確に分けて考える必要がある。

たとえば、オブジェクト｛a｝と｛c｝が同一化により、階層を双方向に

たどることが可能になっているとする（前節[F]）。今｛a｝から｛b｝の持っている性質を証明しようとして最初に｛c｝に入り、｛c｝から｛a｝に入ろうとしてループチェックのためfailすると、3章で述べた28.3）のisa 述語の定義により、失敗パターンがassertされてしまう。その副作用のためバックトラックにより改めて｛b｝の持つ性質を｛a｝から引き出す時に、assertされた失敗パターンが発見されて、｛b｝の持つ性質を引き出すことができない。以上の理由から、このようなループチェックによる失敗の場合には、unknownisaとして真のfailとは異なるassertを行ない、真の失敗と区別する。このようにして失敗パターンのassertを確実なものだけに限定する。

以上を考慮して28.1）から28.3）を、次に示す28.7）から28.14)で、また29）を29.1）で置き換える。ここで28.8）は、階層を1レベル上下する時に使うものであることに注意したい。

```
28.7)   isa(Upper,Prop,S) :-
              (notmemb(Upper,S);
               bottomof(S,B),
               not(unknownisa(B,Prop)),
               asserta(unknownisa(B,Prop)),!,fail),
              !,is(Upper,Prop,S).
28.8)   is(Upper,Prop,[_|T]) :-
              T == 1,!,
              Prop = isa:Upper.
28.9)   is(_,Prop,[H|_]) :-
              anchored(Prop),
              (wfisa(H,Prop),!,true;
               failisa(H,Prop),!,fail).
28.10)  is(Upper,Prop,[H|T]) :-
              (Prop == isa:Upper,!;
               Prop = isa:Upper;
               sem(Upper,Prop,[H|T])),
              wfassert(H,Prop).
28.11)  is(_,Prop,[H|T]) :-
              (var(T);atomic(T)),
              not(wfisa(H,Prop)),
              not(unknownisa(H,Prop)),
              asserta(failisa(H,Prop)),!,fail.
28.12)  notmemb(H,[H|_]) :- !,fail.
28.13)  notmemb(H,[_|T]) :- (var(T);atomic(T)),!.
28.14)  notmemb(H,[_|T]) :- notmemb(H,T).
29.1)   hasa(Part,X:Y,S) :-
              X == hasa,
              notmemb(Part,S),
              (Y = Part;
               sem(Part,hasa:Y,S)).
```

4.1.3 実験結果

4.1.1 節で与えたmkeqの定義と、4.1.2 節の28.7)-28.14)と29.1) の定義を用いて Semantic Matcher の簡単な実験を行なった。

```
(a) ?- mkeq(x#1,y#1).
(b) yes
(c) ?- mkeq(x#1,misterAI#1).
(d) yes
(e) ?- sem(y#1,P,_).
(f) P = isa:x#1;
(g) P = isa:misterAI#1;
(h) P = address:america;
(i) P = isa:human;
(j) P = isa:mammal;
(k) P = bloodTemp:warm;
(l) P = isa:animal;
(m) P = isa:creature;
(n) no
(o) ?- mkeq(mcCarthy#1,mister5G#1).
(p) no
(q) ?- mkeq(mcCarthy#1,misterAI#1).
(r) yes
(s) ?- mkeq(mcCarthy#1,clyde#1).
(t) no
(u) ?- mkeq(elephant#1,clyde#1).
(v) yes
(w) ?- mkeq(elephant#2,clyde#1).
(x) no
```

(a)では、x#1 とy#1 という２つのObjectを作り出し、それらを等しいと置いている[D] 。(c) ではx#1 と misterAI#1 とを等しいと置いている[D] 。したがって(e) でy#1 のもつ性質を聞くと(f)-(m) の応答に見るように、y#1 はmisterAI#1の性質を受け継いでいることが分かる[F] 。(o),(q),(u),(w) に対する応答は [E]によるが、それぞれi),ii),iii),iv) で説明したSemantic Matchingの例になっている。(s) に対する応答は [C]によるが、これは(v) で説明したSemantic Matching の例になっている。ここで、(u)と(w) では elephant#1とelephant#2のいずれにも、age についての記述がないにもかかわらず、正しい応答を示していることに注意したい。その理由は 2.1節で既に説明した。

4.2 自然言語の意味処理

4.2.1 DCKRによる辞書項目の記述

意味処理の基本となるのは、辞書項目の記述である。これまで、辞書項目の記述形式として最も良く用いられてきたものは、フレーム（Object）形式で

あろう。DCKRでは、一つのObjectは幾つかのスロットの集合から構成される。そして一つのスロットには、sem 述語をヘッドとする一つのホーン節が対応する。ただし、このsem 述語の第一引数にはObject名がくる。

意味処理で用いるスロットのスロット値は、最初未定であるが、意味処理が進むにつれて、スロット値が確定する。このことをスロットがフィラーで満たされるという。フィラーが、あるスロットのスロット値になり得るためには、フィラーはそのスロットに書かれた制約条件を満たさなければならない。スロットに書かれた制約条件をフィラーが満たすと、アクションが起動され、意味構造を抽出したり、より深い推論を行なう。

スロットに書かれる制約条件には、大別して統語的制約条件と意味的制約条件とがある。統語的制約条件は、フィラーが文中で果たす統語的役割を表わす。意味的制約条件は、フィラーの持つべき意味についての制約条件である。

典型的な意味処理は、およそ次のようにして進行する。

i) 選択したスロットの統語的ならびに意味的制約条件をフィラーが満たせば、アクションを起動して終了、さもなければii) へ。

ii) 次に選択すべきスロットがあれば、それを選択してi)へ、さもなければiii)へ。

iii)上位のプロトタイプがあれば、そのスロットを取り出しi)へ、さもなければ意味処理失敗として終了。

以上のi)からiii)の意味処理の手順を観察すると、次のことが分かる。

a) i)の意味的制約条件は論理式で表現することが多い。後述するように、これはDCKRで容易に表わすことができる。

b) ii) のスロットの選択には、Prologに組み込みのバックトラック機構を利用することができる。DCKRでは、スロットが一つのホーン節として表わされているからである。

c) iii)は、2.1 で述べたDCKRのもつ知識継承機構によって、容易に実現することができる。

このように、辞書項目記述形式をDCKRにすれば、意味処理の中核をなすプログラムをPrologに組み込みの基本計算機構で代用可能なことが読み取れる。以下ではそれを実例により示す。はじめにDCKRによる辞書項目openの記述例を示す。

```
41) sem(open,subj:F^In^Out,_) :-
            sem(F,isa:human,_),
            addProp(agent:F^In^Out);
            (sem(F,isa:eventOpen,_); sem(F,isa:thingOpen,_)),
            addProp(object:F^In^Out);
            sem(F,isa:instrument,_),
            addProp(instrument:F^In^Out);
            sem(F,isa:wind,_),
            addProp(reason:F^In^Out).
```

```
42) sem(open,obj:F^In^Out,_) :-
            (sem(F,isa:eventOpen,_);sem(F,isa:thingOpen,_)),
            addProp(object:F^In^Out).
43) sem(open,with:F^In^Out,_) :-
            sem(F,isa:instrument,_),
            addProp(instrument:F^In^Out).
44) sem(open,P,S) :-
            T = [open|S],
            isa(action,P,T);
            isa(event,P,T).
```

41),42),43) は、openを構成する subj, obj, withという名前のスロットである。変数Fは、これらのスロットのフィラーである。スロット名はフィラーFが満たすべき統語的制約条件を表わす。subj, obj, with は、フィラーが文中でそれぞれ主語、目的語、withのついた前置詞句の役割を担っていなければならないことを表わしている。一方これらスロットに対応するホーン節のボディーには、意味的制約とアクションの対（以後これをＣＡ対とよぶ）が記述されている。たとえば41）のボディーには、四つのＣＡ対が記述され、その各々がor(";") で結合されている。

最初のＣＡ対：

```
sem(F,isa:human,_),
addProp(agent:F^In^Out);
```

は、フィラーFが人間（human ）なら、フィラーFの深層格を動作主(agent)にするアクションaddProp(agent:F^In^Out) を起動し、深層格構造を抽出する。ここで、フィラーFが人間であるかどうかを調べるsem(F,isa:human,_)はフィラーFに対する意味的制約条件を表わしている。述語addProp は、抽出した深層格構造をInに付加した結果をOut にしてリターンする。

上記したように、意味的制約条件の検査は、Prologプログラムを直接実行することに置き換えることができる。したがって、たとえば「血液型 AまたはB の人」などという比較的複雑な意味的制約条件も、次に示すように容易に記述することができる：

```
sem(F,isa:human,_),
        (sem(F,bloodType:a,_);
         sem(F,bloodType:b,_))
```

第二のＣＡ対：

```
(sem(F,isa:eventOpen,_); sem(F,isa:thingOpen,_)),
 addProp(object:F^In^Out);
```

は、フィラーが、開催する出来事（eventOpen)または開くもの（thingOpen ）なら、フィラーの深層格を対象格（object）にすることを記述している。

第三のＣＡ対：

```
sem(F,isa:instrument,_),
addProp(instrument:F^In^Out);
```

は、フィラーが道具ならフィラーの深層格を道具格（instrument）にすることを記述している。

第四のＣＡ対：

```
sem(F,isa:wind,_),
addProp(reason:F^in^Out).
```

は、フィラーが風（wind）ならフィラーの深層格を理由格（reason）にすることを記述している。

以上の説明から42)，43)のスロットの意味は明らかだろう。前置詞句に対応するスロットはwithの他に多数あるが、説明を単純化するために省略してある。

44）は、フィラーが41)，42)，43）のスロットを満たすことができない時に（バックトラックにより）実行し、openの上位のactionとevent にあるスロットにアクセスするための記述である。これは、2.1 節では知識の継承として詳しく説明したが、知識の多重継承（multiple inheritance）の例にもなっている。

41)-44）のopenの記述は完全にコンパイルすることができるので、高速化を図ることができる。これまでの辞書項目の記述形式では、辞書項目を一つの大きなデータ構造として表現していたため、それをコンパイルできないのと良い対照をなす。

4.2.2 文法規則の記述

文法規則はDCG [Pereira 80]の記法を用いる。意味処理はDCG の補強項で行なう。Declarative な文を解析する簡単な文法規則の例を次に示す。

```
sdec(CogVp,SemSdcc) -->
        np(CogSubj,SemSubj),
        vp(CogVp,SemVp),
        {concord(CogSubj,CogVp),
         seminterp(SemVP,subj:SemSubj,SemSdec)}.
```

{ , } で囲まれた部分が補強項である。述語concord は主語と動詞の一致を調べるものである。述語seminterp は、単に形式を整えてsem 述語を呼び出すもので、およそ５行の小さなプログラムである。この文法規則の例では、SemVp の中の主動詞のフレーム（たとえば41)-44）の open フレーム）のsubj スロットを、SemSubj の中の主名詞（フィラー）が満たし得るかどうかを調べ、意味処理結果をSemSdec に返す。したがって我々は意味処理用のプログラムをほとんど作る必要がないことが分かる。

意味処理はDCG に付加された補強項で行なっている。そのため統語処理と意味処理とが融合し同時に進行することになる。これは心理学的にも妥当な言語処理のモデルであると主張されていたものである。

4.2.3 実験結果

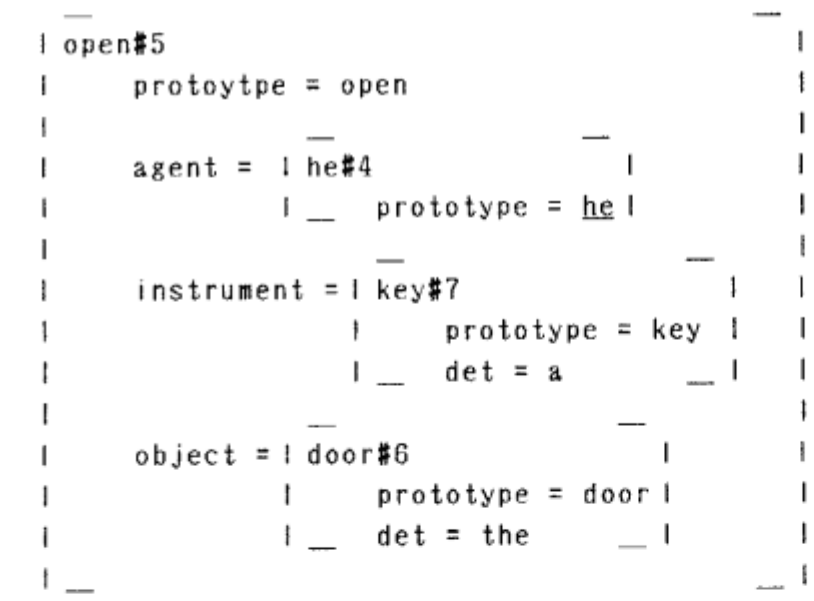
4.2.1 と4.2.2 で説明した考え方を用いて、意味処理を行なった結果について述べる。意味処理に用いた文は"He opens the door with a key"である。

```
Input Sentences
1: He opens the door with a key.

Semantic Structure is:

sem(open#5,P,S) :-isa(open,P,[open#5|S]).
sem(open#5,agent:he#4,_).
sem(open#5,instrument:key#7,_).
sem(open#5,object:door#6,_).
sem(he#4,P,S) :- isa(he,P,[he#4|S]).
sem(door#6,P,S) :- isa(door,P,[door#6|S]).
sem(door#6,det:the,_).
sem(key#7,P,S) :- isa(key,P,[key#7|S]).
sem(key#7,det:a,_).
```

この他にも、"the door with a key" の意味処理結果が得られるが、説明を省略する。上記した意味処理結果を、LFG [Bresnan 81]で用いられている形式で表現すると、以下のようになろう。



ここで次のことに注意したい。それは、意味処理結果もまたDCKRの形式である、ということである。DCKR形式の意味処理結果を得ておくことにより、たとえば
"With what does he open the door?"という質問文から
sem(open#J,instrument:X,_)
を得て、それを実行することにより、
J = 5
X = key#7
という答えを得ることができる。

4.2.4 DCKRと自然言語理解システム

ここでDCKRと自然言語理解システムとの関連を述べておく。これまで述べてきたことから、自然言語理解システムのアーキテクチャとしてFig.1 のような図式が考えられる。

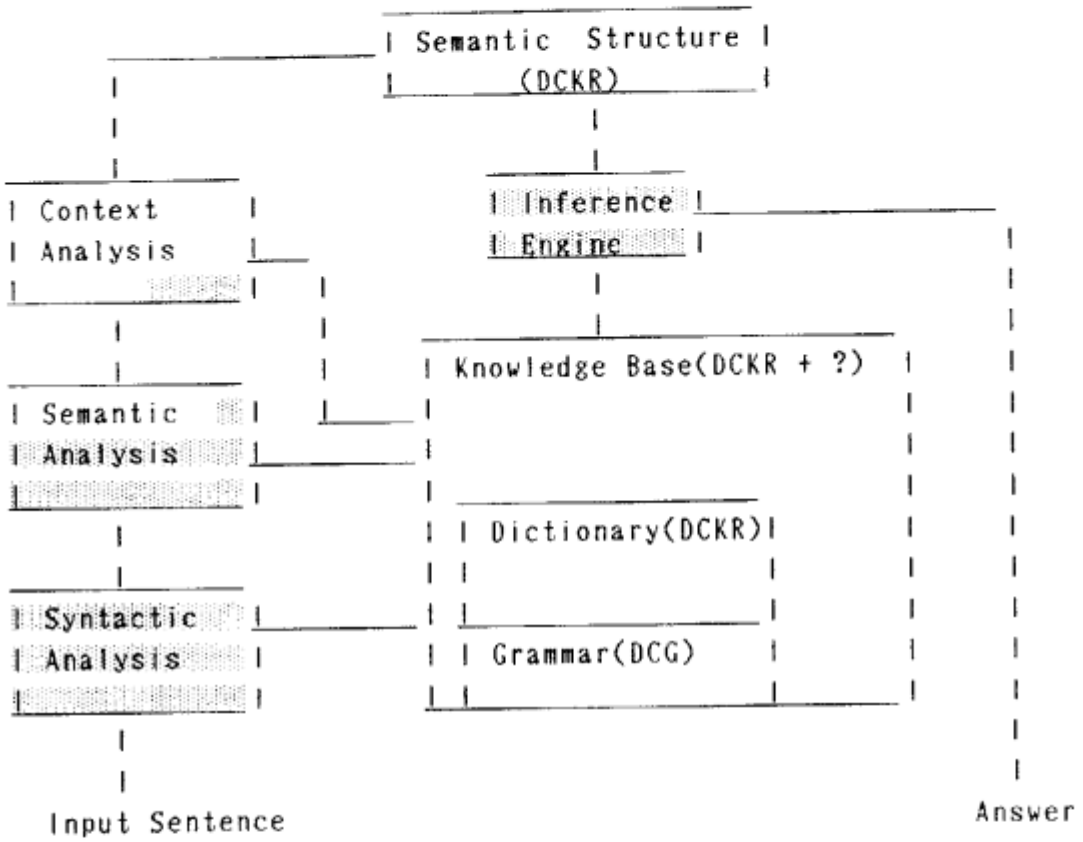


Fig.1 DCKRと自然言語理解システム

Fig.1 の [ ] は、Prologに組み込みのインタープリタによって実現される部分を示す。前章までの説明から、一般常識の一部と辞書とがDCKRで記述されていれば、文脈処理の一部と意味処理のほとんどは、Prologに組み込みの機能に任せることができる。統語処理は、文法をDCG [Pereira 80]で記述すれば、それをPrologプログラムに自動的に変換することが可能だから、いわゆるパージングは、変換されたPrologプログラムを実行することに置き換えられる。実際には [Pereira 80] の方法では、トップダウンパージングが実現されるため、左再帰型の文法規則が扱えない。そこで我々はDCG で記述した文法規則を、ボトムアップパージングを行なうPrologプログラム（BUP 節）に変換する手法を用いている [Matumoto 83]。いずれにしても、Fig.1 に示したように、統語処理はほぼ完全にPrologインタープリタに任せることができる。パージングを行なうパーザを作る必要はないのである [田中 84]。

以上の事実と、推論エンジンがPrologインタープリタであるとすれば、Prologマシン＋alpha が自然言語処理マシンである、と結論してもよいだろう。

ここでalpha が何かと問われれば、それは知識ベースマシンであるということになるだろう。いずれにせよこのような考え方は、我が国の第五世代コンピュータ計画の目指す方向と一致している。

5．おわりに

複数の文の連鎖からなる談話を理解するためには、前文を受けながら談話が進行する過程で様々なObjectが発生し、それらのうちどれとどれが同一であるかを推論することが必要になる。たとえば、言語学でいう前方照応がその典型である。たとえば、”..... 酸素が発生した。その気体を....”という文の系列（談話）に表われる”酸素”と”気体”とが同一であるということは、4.1 で説明したSemantic Matcherによって知ることができるだろう。

等位接続した名詞のどれとどれとが並立しているかを認識することは、自然言語処理では困難な問題の一つである。言語学者はそれを主として統語論の立場から論じている。しかしそれが不十分なものであることも良く知られている。並立する名詞の同定には、意味的な情報が必要になることが多い。たとえば、”日本の動物と英国の植物”という名詞句に表われる”と”でつながれた名詞のうち、”動物”と”英国”とを並立するものとして認識してはならない。ところが、先と全く同一の品詞の並びを持つ、”日本の動物と植物の歴史”という名詞句に表われる”動物”と”植物”とは、並立するものとして認識しなければならない。以上の認識に意味的な情報が用いられていることは明らかだろう。

しかし、並立する名詞がどれとどれであるかを認識するためには、4.1 で説明したオブジェクト間の同一性を判定するアルゴリズムでは、不十分である。4.1 のアルゴリズムによれば、”動物”と”植物”とは異なるオブジェクトとして判定されてしまうからである。むしろ必要なものは、二つのオブジェクト間の相似性を判定するアルゴリズムである。筆者は今後、Semantic Matcherを拡張して、こうした問題に応用することを考えている。 このようなSemantic Matcher の実現それ自身、長期を要する研究課題であるが、それは、人工知能の研究分野で今後重要になると思われるAnalogicalReasoning や、それをさらに一歩進めた学習の問題などにも応用可能だと思われる。オブジェクト相互の同一性、相似性を認識することが、上記した困難な問題を解決するための第一歩であると考えられるからである。

フレーム形式の知識表現の利点として、chunkning of knowledgeということが挙げられていた。Chunkingによって、ある一つのフレームにアクセスするだけで、それに関連する全ての知識（スロット）を一度に得ることができるので、連想に都合が良いと考えられたからである。それはまた、心理学的にも妥当な記憶モデルだと主張された。ところが、これまでのDCKRによる知識表現では、フレームのように、関連するスロットを囲い他と区別する枠がなく、世界に存在する全てのスロットは全て対等だと見做される。これは一見フレーム的な考え方に反するようにみえる。しかし、スロットに対応するホーン節のヘッドにあるsem 述語の第一引数（Object名）をハッシュ化しておくことにより、関連知識を素早く持ち込むことができるので、フレーム的な考え方は容易にシ

ミュレーションすることができる。幸いなことにPrologには、関連知識を全てリストにして引き出すための述語setof，bagofが用意されているので、それを利用することもできる。

2.1 の最後で、DCKRで書かれた知識の読み易さと書き易さについて簡単に触れた。しかし、我々は更に高水準の知識表現言語を開発すべきである。たとえば02）の記述中の第三引数は、そのような高水準の知識表現言語をコンパイルする過程で自動的に付加してほしい。また、41)-43）の記述中に現われる変数InとOut も自動的に付加してほしい。このように考えると、DCKRの表現は、いわば機械語表現であることが分かる。DCKRを機械語とみなした、より高水準の知識表現言語を開発する必要がある。

本稿では、DCKRの自然言語処理への応用について論じてきた。しかし、DCKRは知識表現の形式である。現在筆者の研究室では法律の知識をDCKRで記述する研究を進めている。法律の専門家システムを実現することを目指しているのであるが、自然言語を扱っていたのでは気付かなかった問題が抽出できるのではないかと考えている。

最後に、知識表現には困難な問題が山積している。高階の知識をどのように表現するか、否定の知識をどのように表現する、集合をどのように表現するか、デフォールト推論をどのように実現するか、などである。このような困難な問題を射程に置きながら、自然言語理解システムの研究に本格的に取り組みたいと考えている。その過程で、恐らく様々な予想もしなかった問題に遭遇すると思われる。DCKRの真価がそこで新たに問われることになるだろう。

謝辞

本研究の契機は、ICOTの淵　一博所長とのディスカッションにある。筆者の良き指導者であり助言者である同氏に感謝する。ICOTの古川康一第１研究室長には、知識表現にかんする最新の研究成果について教えていただいた。ICOTの横井俊夫第５研究室長には、本稿をICOT-TR にしていただくために、お骨折りいただいた。筆者の所属する田中研究室の小山晴生君は第二章で、佐藤直人君は第三章で、奥村学君、池田光生君、上脇　正君、沼崎浩明君は第4.2 節で協力していただいた。斎藤佐知江さんには、本稿の作成で御苦労をいただいた。以上の諸氏に感謝する。

６．参考文献

[Bobrow 77] Bobrow,D.G. et.al.: An Overview of KRL-0, Cognitive Science, 1, 1, 3-46(1977).

[Bowen 85] Bowen,K.A.: Meta-Level Programming and Knowledge Representation, Syracuse Univ.,(1985).

[Bresnan 82] Bresnan,J.(ed.): The Mental Representation of Grammatical Relations, MIT Press,(1982).

[Colmeraure 78] Colmeraure,A.: Metamorphosis Grammer, in Bolc (ed): Natural Language Communication with Computers, Springer-Verlag

133-190(1978).
[Goebel 85] Goebel, R.: Interpreting Descriptions in a Prolog-Based Knowledge Representation System, Proc. of IJCAI'85,711-716 (1985).
[Hayes 80] Hayes,P,J.: The Logic of Frame, in Frame Conceptions and Text Understanding, Walter de Gruyer, Berlin, 46-61(1980).
[小山 85]小山晴生（他）: Definite Clause Knowledge Representation, Proc.of LPC'85, 95-106(1985).
[Matsumoto 83] Matsumoto,Y. et.al.:BUP--A Bottom-up Parser Embedded in Prolog, New Generation Computing, 1, 2, 145-158(1983).
[Mukai 85] Mukai,K.: Unification over Complex Indeterminates in Prolog, Proc.of LPC'85, 271-278(1985).
[Nilsson 80] Nilsson,N.J.: Principles of Artificial Intelligence, Tioga, (1980).
[Pereira 80] Pereira,F. et.al.: Definite Clause Grammer for Language Analysis --A Survey of the Formalism and a Comparison with Augmented Transition Networks, Artificial Intelligence, 13, 231-278(1980).
[田中 84]田中穂積（他）：自然言語処理におけるProlog，情報処理，25, 12, 1396-1403 (1984).
[田中 85]田中穂積（他）: Definite Clause Dictionary--Prologによる辞書項目記述と意味処理，Proc.of LPC'85, 317-328(1985).