知識表現形式 DCKR とその応用

田中 穂積　小山 晴生　奥村 学

[Nilsson 80]は，述語論理式で表現した知識を一度オブジェクト形式に変換し，それを意味ネットワーク形式で表現する方法を述べている．推論は意味ネットワークをたどる操作に還元されるので，推論を行なうためには，意味ネットワークに対するインタープリタを作る必要がある．それに対して本稿で提案するDCKRはホーン節形式の述語論理式で知識を表現する．したがって推論は，Prologに組み込みの機能をそのまま用いることができる．知識継承についてもPrologのユニフィケーション機構をそのまま利用できるという特長がある．DCKRについては第2章で説明する．第3章では，DCKRを用いた自然言語の意味処理を取り上げ，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．第4章では，時間に関する知識をどのようにDCKRで記述するかを述べる．第5章では，(DCKR以上に)高水準の知識表現言語が必要なことを指摘して，SRL/0とよばれる言語を提案する．

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**(以後オブジェクトとよぶ)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オブジェクト形式に変換し，それを意味ネットワーク形式で表現する方法を述べている．それによれば，オブジェクトに関する推論は，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／問題解決を行なうためには，別途，この意味ネットワークに対するインタープリタを作る必要がある．

一方筆者らは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では，オブジェクトを構成する各スロットを，sem述語(後述)をヘッドとするホーン節の集合とみなす．このようなDCKRで記述した知識に対する推論は，Prologに組み込みの機能をそのまま用いることができるので，推論を行なうための特別なプログラム(インタープリタ)を作る必要はない．

DCKRについては第2章で詳しく説明する．第3章では，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によれば，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も，Prologに組み込みの機能に任せることが可能になる．第4章では，時間に関する知識をどのようにDCKRで記述するかを述べる．第5章では，DCKRは機械語のレベルの知識表現形式であり，高水準の知識表現言語が必要なことを指摘して，SRL/0とよばれる言語を提案する．

DCKR——Knowledge Representation in Prolog and its Applications.

Hozumi Tanaka, Manabu Okumura, 東京工業大学工学部情報工学科.

Haruo Koyama, 日本ブーズ・アレン・アンド・ハミルトン株式会社.

コンピュータソフトウェア，Vol. 3, No. 4(1986), pp. 12-20.

[論文] 1986年8月4日受付.

2 DCKRによる知識表現

2.1 オブジェクトの表現

まず初めにDCKRによる簡単な知識記述例を示す.

```
:-op(100, yfx, -),
  op(100, yfx, :),
  op(90, xfy, #).
1.1)  sem(clyde#1, age:6, _).
1.2)  sem(clyde#1, P, S) :-
          isa(elephant, P, [clyde#1|S]).
2.1)  sem(elephant#1, birthYear:1980, _).
2.2)  sem(elephant#1, P, S) :-
          isa(elephant,P, [elephant#1|S]).
3.1)  sem(elephant, color:gray, _).
3.2)  sem(elephant, P, S) :-
          isa(mammal, P, [elephant|S]).
4.1)  sem(mammal, bloodTemp:warm, _).
```

述語isaの定義を与えておく.

```
a.1)  isa(Upper, P, S) :-
          P=isa:Upper;
          sem(Upper, P, S).
```

述語semの第一引数は,オブジェクト名である. 1.1)と1.2), 2.1)と2.2), 3.1)と3.2), 4.1)は, それぞれclyde#1, elephant#1, elephant, mammalという名前のオブジェクトの記述である. オブジェクトには大別して個体とプロトタイプがある. 心理学者は, プロトタイプのことをstereotypeとよぶことがある. オブジェクト名のうち#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1.1)と1.2)のclyde#1は個体名であり3.1)と3.2)のelephantはプロトタイプ名である. 個体名(またはプロトタイプ名)が等しい述語semをヘッドとするホーン節の集合は, 一つの個体(またはプロトタイプ)を表わす. たとえば, 1.1)と1.2)の記述は, clyde#1という個体を表わし, 3.1)と3.2)の記述は, elephantというプロトタイプを表わす.

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1.1)の記述は「個体clyde#1のageが6である」という事実を表わす. ageがスロット名で, 6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(性質)とよぶ.

2.2 知識継承と推論

1.2)は,「clyde#1はelephantである」という事実を表わすが, 実は上位から下位への知識継承(inheritance of knowledge)の記述にもなっている. 知識継承の観点からは, 1.2)は「elephantが性質Pをもてば, clyde#1も同じ性質Pをもつ」と読むが, この例のように, 述語isaでプロトタイプに連結された個体は, プロトタイプのインスタンスになる. 例えば1.2)の記述により, 個体clyde#1はプロトタイプelephantのインスタンスである.

知識継承がPrologのユニフィケーション機構によって自動的になされることをみるために,

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

を実行すると, 以下のようにelephant#1に関する知識を次々に(自動的に)得ることができる.

```
P=birthYear:1980;
P=isa:elephant;
P=color:gray;
P=isa:mammal;
P=bloodTemp:warm;
```

知識継承によりelephant#1の上位にある知識がすべて得られていることに注意されたい. 一方,

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

を実行すれば, XとYの対としてすべての知識を出力し始める. また,

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

を実行すれば,

```
X=clyde#1;
X=elephant#1;
X=elephant
```

のように, mammalの下位に位置する個体とプロトタイプとを知ることができる.

さて1.2)の記述は, 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みなせる. 1.2)の記述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すべての知識にアクセスすることができる. 一方個体は, 必要に応じて動的に作られるものであるから, 前もって個体の世界の様子を(プロトタイプの世界から)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ために, 述語instance

が用意されている.

```
b. 1)  instance([Instance|Y], Instance) :-
           (var(Y);atomic(Y)), !, nonvar(Instance).
b. 2)  instance([X|Y], Instance) :-
           instance(Y, Instance).
```

1. 2), 2. 2), 3. 2)の記述から明らかなように, 述語semの第三引数には, 上位に向けて階層をたどった道筋をスタックする. これを述語instanceの第一引数に与えて, プロトタイプの世界を呼び出した個体(インスタンス)が何であるかを知り, その個体が持っている知識を取り出す事ができる. たとえば4. 1)のmammalに対してさらに4. 2)の記述があるものとする.

```
4. 1)  sem(mammal, bloodTemp:warm, _).
4. 2)  sem(mammal, age:X, S) :-
           instance(S, Instance),
           sem(Instance, birthYear:Y, _),
           X is 1986 - Y.
```

4. 2)のボディでは述語instanceを用いて, プロトタイプmammalのInstanceを知り, InstanceのbirthYearからageを計算する. それにより,

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

を実行すると,

```
X = 6
```

を得ることができる. 4. 2)のボディは, ageを計算するためのメソッドであるとみなし得る.

部分全体に関しても推移律が成り立つことがある. そのためにDCKRでは以下に示す述語hasaが定義されている.

```
c. 1)  hasa(Part, X:Y, S) :-
           X==hasa,
           (Y=Part;
            sem(Part, hasa:Y, S)).
```

述語hasaの働きを調べるために次の例を考えよう.

```
5. 1)  sem(america, climate:temperate, _).
5. 2)  sem(america, P, S) :-
           T=[america|S],
           (isa(country, P, T);
            hasa(california, P, T);
            ……).
6. 1)  sem(california, P, S) :-
           T=[california|S],
           (isa(usState, P, T);
            hasa(stanford, P, T);
            ……).
```

以上の定義をしてから,

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

を実行してみると,

```
X=california;
X=stanford;
……
```

のように次々にamericaの部分を得ることができる.

最後に[菅原 85]の例題をDCKRで記述してみる. それは「ある場所の気候が分らなければ, その場所の上位部分を探して, そこの気候を利用する」という知識の記述である. これは7. 1)のように記述できる.

```
7. 1)  sem(usState, climate:X, S) :-
           instance(S, Instance),
           sem(Superpart, hasa:Instance, _),
           sem(Superpart, climate:X, _).
```

いま,

```
?-sem(california, climate:X, _).
```

を実行すると, 6. 1)でcaliforniaの上位にあるusStateに至り, 7. 1)の記述から, californiaの上位部分americaのclimateの値temperateを取り出すことができる.

```
X = temperate
```

最後に, a. 1)の定義を拡張することにより, 階層をたどる範囲(知識継承の範囲)を制限することもできることを指摘しておく. これは述語isaとsemの第三引数を利用する.

2.3 一般的な知識の表現と推論

2. 1節のDCKRによるオブジェクトの記述例では, オブジェクトは(第一引数をオブジェクト名とする)述語semをヘッドとするホーン節の集合として表現されていた. そしてオブジェクト名はいずれも, 個体かプロトタイプを表わす定数であった. これに対して述語semの第一引数が(個体を表わす)変数であるような知識がDCKRではしばしば重要な役割を果たす. このような変数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. 第一引数が個体変数の述語semをヘッドにもつDCKR表現は, (Prologの個体変数は全称限量化されているから)一般的な知識である.

[Nilsson 80]の中の一つの例をDCKRで記述し，その働きを調べてみよう．

8. 1) sem(X#J, worksIn:Y#K, _) :-
sem(Y#K, isa:department, _),
sem(Y#K, manager:X#J, _).

8. 1)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)」という知識である．ここでさらに次の事実があるものとしよう．

9. 1) sem(joeSmith#1, worksIn:pd#1, _).

10. 1) sem(pd#1, manager:joeJones#1, _).

10. 2) sem(pd#1, P, S) :-
isa(department, P, [pd#1|S]).

ここで「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．10. 1)を用いて，pd#1のマネージャがjoeJones#1であるということから，8. 1)により「joeJones#1がpd#1で働いている」という事実が導き出されている．

最後にDCKRの有効性を確認するために，読者は[Nilsson 80]の9. 4. 6節のプロダクション規則の表現と8. 1)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で記述された知識に対する推論がPrologに組み込みの機能で実現されることの意味がより良く分かると思う．

3 自然言語の意味処理への応用

3. 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．辞書項目の記述でよく用いられるものとして，フレーム表現(オブジェクト表現)がある．DCKRでは，一つのオブジェクトは幾つかのスロットの集合から構成され，一つのスロットには，述語semをヘッドとする一つのホーン節が対応していた．ただし，この述語semの第一引数にはオブジェクト(フレーム)名がくる．

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，スロットに書いてある制約条件を満たさなければならない．スロットに書かれている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. 2節で述べた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．

11. 1) sem(open, subj:Filler-In-Out, _) :-
sem(Filler, isa:human, _),

```
        addProp(agent:Filler-In-Out);
        (sem(Filler, isa:event, _);
         sem(Filler, isa:thingOpen, _)),
        addProp(object:Filler-In-Out);
        sem(Filler, isa:instrument, _),
        addProp(instrument:Filler-In-Out);
        sem(Filler, isa:wind, _),
        addProp(reason:Filler-In-Out).
11.2)   sem(open, obj:Filler-In-Out, _):-
        (sem(Filler, isa:event, _);
         sem(Filler, isa:thingOpen, _)),
        addProp(object:Filler-In-Out).
11.3)   sem(open, with:Filler-In-Out, _):-
        sem(Filler, isa:instrument, _),
        addProp(instrument:Filler-In-Out).
11.4)   sem(open, P, S):-
        T=[open|S],
        (isa(action, P, T);
         isa(event, P, T)).
```

11.1), 11.2), 11.3)は，オブジェクトopenのsubj, obj, withという名前のスロットである．変数Filler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の統語的制約条件を表わす．すなわち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わなければならないことを表わす．各スロットのボディには，フィラーの意味的制約条件とアクションの対(CA対)を記述する．たとえば11.1)のボディには，四つのCA対の記述があり，その各々が選言記号「;」で結合されている．

例えば最初のCA対:

```
sem(Filler, isa:human, _),
addProp(agent:Filler-In-Out)
```

は，Fillerが人間(human)なら，Fillerの深層格を動作主(agent)にするアクションaddPropを起動し，深層格構造(agent:Filler)を変数Inに付加した結果をOutに返す．ここで，sem(Filler, isa:human, _)はFillerに対する意味的制約条件である．

第二のCA対:

```
(sem(Filler, isa:event, _);
 sem(Filler, isa:thingOpen, _)),
addProp(object:Filler-In-Out)
```

は，Fillerが，催し事(event)または開く物(thingOpen)なら，Fillerの深層格を対象格(object)にすることを記述している．一般にFillerの意味的制約条件として，連言と選言の任意の組合せが記述可能であるが，その解釈はPrologインタープリタが代行してくれる．

以下同様にして第三，第四のCA対の意味は明らかだろう．また以上の説明からスロット11.2), 11.3)の意味も明らかだろう．前置詞句に対応するスロットはwithの他多数あるが本例では省略してある．

11.4)は，Fillerがスロット11.1), 11.2), 11.3)を満たせなかった時，最後に実行され，2.1節で述べた述語isaを介してopenの上位のactionとeventの持つスロットにアクセスすることができる．これは，知識の多重継承(multiple inheritance)の例にもなっている．

11.1)～11.4)のopenの記述は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，コンパイルできなかったのと良い対照をなす．

3.2 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．平叙文を解析する簡単な文法規則の例を次に示す．

```
sdec(CogVp, SemSdec)-->
  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は主語と動詞の呼応を調べるものである．述語seminterpは，SemVpに含まれる主動詞(例えば前節のopen)がオブジェクト名(第一引数)で，SemSubj中の主名詞がフィラーで，スロット名がsubjの辞書項目を呼び出す(例えば11.1)．得られた意味処理結果は，SemSdecに返される．seminterpは5行程の小さなプログラムで，意味処理を行なうことができる．

3.3 実験結果

3.1節と3.2節で説明した考え方を用いて，意味処理をした結果について述べる．意味処理に用いた文は"H

opens the door with a key." である.

```
Input Sentences
|: 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, with a key が door にかかる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 82]の形式で表現すると, 以下のようである.

```
[ open#5
    prototype = open
    agent = [ he#4
              [ prototype = he ] ]
    instrument = [ key#7
                   [ prototype = key
                     det = a ] ]
    object = [ door#6
               [ prototype = door
                 det = the ] ] ]
```

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形式にしてあるということである. DCKR形式の意味処理結果を得ておけば, "With what does he open the door?" という質問文から,

sem(open#J, instrument:X, _)

を得て, 単にそれを実行することにより,

J=5

X=key#7

という答を得ることができる.

3.4 DCKRと自然言語理解システム

これまで述べてきたことから, 自然言語理解システムの一般的な構成図として図1が考えられる.

図1の二重枠の斜線を施した部分は, Prologの基本計算機構に任せることが可能な部分を示す. 前節までの説明から, 一般知識の一部と辞書とがDCKRで記述されていれば, 意味処理のほとんどをPrologの基本計算機構に任せることができることが分かる. 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, いわゆる構文処理は, DCGを変換したPrologプログラムを実行することに帰着される. しかし[Pereira 80]の方法は, トップダウンパージングであるため, 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. それにより, 構文処理はほぼ完全にPrologインタープリタに任せることができるので, 構文処理用のパーザを作る必要はない[Matsumoto 83] [田中 86].

以上の事実と, Prologインタープリタが推論エンジンであるとすれば, 「Prologマシン+α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αが何かと問われれば, それは知識ベースマシンである. 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.

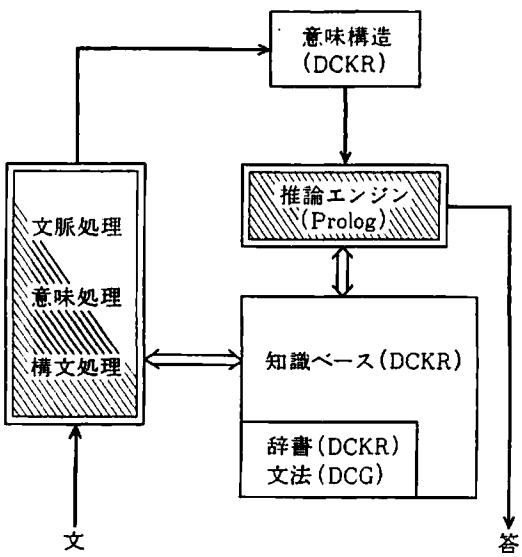

図 1 自然言語理解システム

4 時間に関する知識の表現

次に時間に関する知識をDCKRでどのように表現するかについて説明する．まず離散的な時刻(整数で表わす)で世界の状態が変化する場合を考える．時刻Time1からTime2までの時間帯をTime1-Time2で表わす．以下では常に時間帯を扱うことにするから，時刻Time1は，Time1-Time1で表わされることになる．時間帯Time1-Time2で成立している知識を一般に次のように表わす．

```
sem(⟨objname⟩, ⟨property⟩,
      T1-T2, TT1-TT2) :-
   time(Time1-Time2, T1-T2, TT1-TT2).
```

ここで，述語timeは，第一引数の時間帯Time1-Time2と，第二引数の時間帯T1-T2との重複時間帯をとり，結果を第三引数の時間帯TT1-TT2とする．重複時間帯がなければ失敗する．ただしT1あるいはTime1が変数なら時刻0を仮定し，T2あるいはTime2が変数なら現在以降任意の時刻を仮定する．以下に簡単な例を挙げる．

```
sem(clyde#1, location:india, T, TT) :-
  time(1-3, T, TT).
sem(clyde#1, location:japan, T, TT) :-
  time(4-10, T, TT).
sem(clyde#1, location:india, T, TT) :-
  time(11-12, T, TT).
```

以上に対して次を実行してみる．

```
?-sem(clyde#1, location:Location, 3-6, TT).
```

これは「時間帯3-6でclyde#1はどこにいたか？」という意味の質問である．実行結果は，

```
Location=india
T=3-3;
Location=japan
T=4-6
```

となる．また次の質問に対しては，

```
?-sem(clyde#1, location:india, T, TT).
TT=1-3;
TT=11-12
```

の結果が得られる．

[米崎84]は，時間の概念を様相として取り入れている．それによれば，次の五つが基本概念として取り上げられている．

p：現在においてpが成り立つ．

sometimes p：現在以降pが成り立つ時刻が存在する．

always p：現在以降常にpが成り立つ．

next p：現在の次の時刻でpが成り立つ．

p until q：現在以降qが成り立つ時刻までは常にpが成り立つ．

DCKRでは，sometimes, always, next, untilは，それぞれオペレータとして扱われる．以下では，DCKRによるsometimesの定義を与えておくが，他の時間概念の定義と詳細な説明は[小山86]を参照されたい．

```
sometimes sem(X, Y, T-T, TT) :-
  sem(X, Y, T-T1, TT).
```

現在時刻は質問時にTに与えるが，ボディのT1は変数のまま残されるから，時刻T以降の任意の時刻で一度でもボディのsem(X, Y, …)が成り立てば，sometimes sem(…)が成功することになる．

5 高水準知識表現言語SRL/0

第2章で説明したDCKRによる知識の記述は，さほど困難ではないと思われるが，第3章の意味処理用の辞書項目の記述は必ずしも容易ではない．DCKRによる表現は，いわば機械語のレベルの表現である．そこで我我はさらに高水準の知識表現言語を開発すべきである．本章では，そのための一つの試みとして高水準知識表現言語SRL/0を提案する[奥村86]．SRL/0で表現したものはトランスレータによって最終的にDCKRの表現に変換される．第3章までに説明したDCKR表現と等価な知識を，SRL/0で表現したものを以下に示す．

```
clyde#1 ::
        [age:6]
        [isa:elephant].
elephant#1 ::
        [birthYear:1980]
        [isa:elephant].
   elephant ::
        [color:gray]
        [isa:mammal].
mammal ::
        [bloodTemp:warm]
```

```
        [age:X
            where { X is 1986 - birthYear! in-
                    stance } ].
```

ここで，where…はスロットに対する制約条件を，また birthYear!instance は，mammal のインスタンスの birthYear の値を意味していることに注意[Mukai 85].

```
    america ::
        [climate:temperate]
        [isa:country]
        [hasa:california]
        .....
    california ::
        [isa:usState]
        [hasa:stanford]
        .....
    usState ::
        [climate:X
            where
                Superpart hasa:instance,
                Superpart climate:X].
```

一般的な知識の記述:

```
    X#J ::
        [worksIn:Y#K
            where Y#K isa:department,
                  Y#K manager:X#J].
```

以下の意味処理用の辞書項目記述は，11.1)～11.4)に対応しているが，より精密な記述になっている．11.1)～11.4)の記述では，"The conference opens with a key." 等といった意味的に異常な文を受け入れてしまうのが問題である．11.1)～11.4)のスロット記述では，スロット相互の共起関係が記述できないからである．

以下の記述では，スロット相互間の共起関係に関する条件は，when…で表わされている．また，スロット名の直後の $ 記号は，満たされていないスロット(unsatisfied slot)を表わす．$ 以下は CA 対を表わす．ただし CA 対には，共起関係についての条件が付加されることもある．[ ]で囲まれたスロットは必須格を，( )で囲まれたスロットは任意格を表わす．それらがどのような DCKR 表現に対応するか，そしてそれらがどのように動作するかは[奥村 86]を参照されたい．

意味処理用辞書項目記述例:

```
    open ::
        [subj $ isa:human => agent]
        [obj $ isa:event;
              isa:thingOpen => object]
        (with $ isa:instrument
           when object!instance isa:thingOpen
                => instrument)
        (with $ isa:animal => coagent)
    ::
        [subj $ isa:event;isa:thingOpen
              =>object]
    ::
        [subj $ isa:instrument=>instrument;
              isa:wind => reason]
        [obj $ isa:thingOpen => object]
    ::
        ((at;in) $ isa:place => location).
```

ここで，when object!instance isa:thingOpen という記述は，open のインスタンスの object スロットの値が thingOpen でなければならないということを記述している．when を用いた記述では，where 以下の意味制約条件が満たされない場合には直ちに失敗するのに対し，when を用いた記述では，意味処理が十分に進んでいないため，インスタンスの object スロットにまだ値がない場合には，値が満たされるまでデモンを作って待つ．その後，値が満たされた時点でデモンが起動され，意味制約条件を調べて成功か失敗かを決める．

6 おわりに

フレーム形式の知識表現の利点として，chunking of knowledge が挙げられていた．知識の chunking(フレーム化)によって，一つのフレームにアクセスするだけで，それに関連するすべての知識(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い．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述語 sem の第一引数(オブジェクト名)をハッシュ化しておくことにより，関連知識

にのみ高速にアクセスすることができるので，フレーム的な考え方は容易にシミュレートすることができる．幸いなことにPrologには，関連知識をすべ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，それらを利用することもできる．

第3章では，DCKRの自然言語処理への応用について論じた．これまで意味処理は困難な技術であるとされてきた．しかしDCKRによる辞書項目記述を行なえば，従来考えられていた困難さは減少する．意味処理用のプログラム作成労力が大幅に軽減されるからである．したがって，DCKRにより本質的に困難な問題に取り組む足掛りが得られた，と筆者は考えている．たとえば文脈処理は困難な問題の一つである．今後DCKRを武器にしてより複雑な文脈の問題に取り組みたいと考えている．

これまでDCKRによる知識表現の利点について述べて来た．DCKRの問題は，すべての知識が，述語semをヘッドとするホーン節であることにある．このことは知識が大規模になるにつれて，知識検索の効率が悪くなることを意味している．プロトタイプの記述については，コンパイルすることにより(述語semの第一引数である)プロトタイプ名をハッシュ化し，高速な検索が期待できる．しかし個体の記述については，オペレータ#が述語semの第一引数(個体名)に現われるので，コンパイルによる高速化は期待できない．この問題は，述語semの実行を知識ベースマシン起動のタイミングとしてとらえ，検索を知識ベースマシンに任せることで解決すべきだと考えている．第五世代コンピュータ計画で設計が進んでいる知識ベースマシンの実現が期待される．

知識表現にはまだ困難な問題が山積みされている．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か，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．これらの問題を解決する過程で，DCKRの真価が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，知識表現に関する最新の研究成果について教えていただいた．筆者の所属する田中研究室の佐藤直人君は第2章で，池田光生君，上脇正君，沼崎浩明君は3.1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．

参考文献

[Bobrow 77] Bobrow, D.G. *et al.*: An Overview of KRL-0, *Cognitive Science*, Vol.1, No.1(1977), pp.3-46.

[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 Grammar, in Bolc, L.(ed.): *Natural Language Communication with Computers*, Springer-Verlag, 1978, pp.133-190.

[Goebel 85] Goebel, R.: Interpreting Descriptions in a Prolog-Based Knowledge Representation System, *Proc. of IJCAI '85*, 1985, pp.711-716.

[Hayes 79] Hayes, P.J.: The Logic of Frames, in Metzing, D.(ed.): *Frame Conceptions and Text Understanding*, de Gruyter, 1979, pp.46-61.

[小山 85] 小山晴生(他): Definite Clause Knowledge Representation, *Proc. of LPC '85*, ICOT, 1985, pp.95-106.

[小山 86] 小山晴生: Prologによるstructured objectの表現形式と推論，東京工業大学理工学研究科修士論文，1986.

[Matsumoto 83] Matsumoto, Y. *et al.*: BUP——A Bottom-up Parser Embedded in Prolog, *New Generation Computing*, Vol.1, No.2(1983), pp.145-158.

[Mukai 85] Mukai, K.: Unification over Complex Indeterminates in Prolog, *Proc. of LPC '85*, ICOT, 1985, pp.271-278.

[Nilsson 80] Nilsson, N.J.: *Principles of Artificial Intelligence*, Tioga, 1980.

[奥村 86] 奥村学(他): 意味記述用言語SRL/0の設計とDCKR，情報処理学会情報学基礎研究会資料，1986, pp.1-2.

[Pereira 80] Pereira, F. *et al.*: Definite Clause Grammar for Language Analysis——A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artif. Intell.*, Vol.13(1980), pp.231-278.

[菅原 85] 菅原俊治: フレームシステムにおける継承機能の拡張，知識情報処理シンポジウム論文集，文部省科研費特定研究「多元知識情報」総括班，1985, pp.97-106.

[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*, ICOT, 1985, pp.317-328.

[田中 86] 田中穂積(他): 自然言語処理のためのソフトウェアシステムLangLAB, *Proc. of LPC '86*, ICOT, 1986, pp.5-12.

[米崎 84] 米崎直樹(他): 時間論理プログラミング言語Templog，日本ソフトウェア科学会第一回大会論文集，1E-4, 1984.