Definite Clause Knowledge Representation

-- Prolog による structured object の表現形式と推論 --

小山 晴生、 田中 穂積
(東京工業大学 工学部)

The knowledge representation is one of the most important problems in Artificial Intelligence. There are two aspects of knowledge representation. One is how to store knowledge in what forms, and the other is how to manipulate it. Many systems have proposed to represent structured knowledge using either frames, semantic nets, or logical forms. [Nilsson 80] has already dicussed that there are close relations among them. He proposed the way to translate structured knowledge(structured objects) in first order logic into equivalent semantic nets, which are used to carry out inferences. Therefore, it is necessary to develop a program for interpreting the semantic nets.

This paper describes the way to represent the knowledge of structured objects by means of Definite Clauses (Prolog programs), and then discusses some advantages of our method over [Nilsson 80]. We call our knowledge representation form DCKR(Definite Clause Knowledge Representation). DCKR provides a natural and simple way to organize various types of knowledge, for example, attribute-value knowledge, if-needed knowledge, and default knowledge. Furthermore, we can express inheritance of knowledge easily through the unification mechanism of Prolog interpreter. Representing all knowledge in DCKR, almost all inferences are carried out through Prolog interpreter. Moreover, the meaning of structured objects becomes more clear, since we can regard it as meaning of Prolog program.

1. はじめに

コンピューターによって知識を扱うことの重要性は、人工知能の研究分野できすきすたかまってきている。ここで問題となるのは、知識をどのような表現形式で蓄えておくかという知識ベースに関すること、及びそれをどのように扱うかという推論に関することである。

前者については、構造を持ったデータをどのような形式で表現するかをめぐってこれまでフレーム、セマンティックネット、述語論理表現などが提案されている。これらの間にはたがいに深い関係があることはすでに指摘されている[Nilsson 80]。Nilssonは、このような構造に重点を置いた表現を structured object と呼び、一階の述語論理で表現した structured object についての知識を、それと等価なセマンティックネットに変換し推論を行う方式を提案している。それによれば推論はこのセマンティックネットをたどる操作に還元されることから、推論を行う interpreter を別途作成する必要がある。また、structured object の意味は interpreter の解釈法に依存するので、structured object自体は単なるデータにすぎないことになる。

本論文では、structured object に関する知識をホーン節の集合(Prolog プログラム)で表現する方法を提案する。これを以下では、DCKR (Definite Clause Knowledge Representation) と呼ぶ。DCKR によれば、従来の知識表現におけるいくつかの問題、例えば知識の継承や、if-needed型の知識の表現、あるいはdefault の問題を Prolog に組込みの機能をそのまま用いて扱うことができる。したがってその実現は極めて自然で容易である。知識が Prologプログラムで表現されているため、それに対する推論を行うための interpreter を別途作成する必要がないだけでなく、効率に関しても知識を直接実行できるため改善が期待されるとともに、structured object の意味が Prologプログラムの意味に帰着できるので論理的意味が明確化する。

It is the last one.
Please copy now!!!

以下ではまず本方法の基本的な考え方を Nilsson の方法と対比させながら説明する。それにより本方法の利点を明らかにする。次に本方法による知識の継承、if-needed型の知識の表現、default の扱いを実際の例を用いて説明し、最後に将来の問題を論じる。

2. DCKR の基本的な考え方

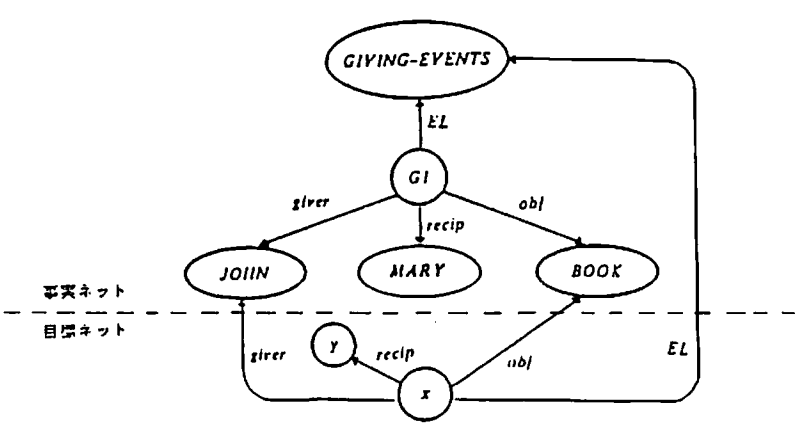
[Nilsson 80] は知識表現の問題を次のように考えている。知識はまず述語論理で表現される。次に相互に関係のある述語論理表現をユニットと呼ばれるより大きな構造にまとめる。この大きな構造はシステムが対象としている領域における object に対応している。ある object に関する情報が必要になった場合には、このユニットをとりだし、それに関連するすべての情報を得る。彼は、表現の構造(structure)に重点を置いたこのような表現形式を structured object と呼んでいる。彼の述語論理表現はモジュラリティを重視して、基本的には二項関係で構成し、それを有向グラフ(セマンティックネット)として表現する。このグラフ上での推論は、事実を表すグラフと目標となるグラフの照合プログラム(インタープリタ)によって実現される。

例えば 'John gave Mary the book' という知識は GIVE(JOHN, MARY, BOOK) と表せる。これを二項述語化すると、(∃x)[EL(x,GIVING-EVENTS)∧GIVER(x,JOHN)∧RECIP(x,MARY)∧OBJ(x,BOOK)] となる。述語 EL は、'集合とその要素' の関係にあることを示すために使われる。ここで、存在限量詞で限量された変数をスコレム化したとき、ある特定の '物のやりとりに関する事象' に G1 という名前を与えると、EL(G1,GIVING-EVENTS)∧GIVER(G1,JOHN)∧RECIP(G1,MARY)∧OBJ(G1,BOOK) と表せる。これは集合 GIVING-EVENTS 上に定義された関数を使った表現 EL(G1,GIVING-EVENTS)∧EQ[giver(G1),JOHN]∧EQ[recip(G1),MARY]∧EQ[obj(G1),BOOK] と等価である。これを次のようなユニット表現で表す。

```
G1
    self : (element-of GIVING-EVENTS)
    giver: JOHN
    recip: MARY
    obj  : BOOK
```

さらにこれを、セマンティックネットへと変換する。 figure 2.1 中の事実ネットの部分が上記したユニット G1 に対応している。

[figure 2.1]
([Nilsson 80])



セマンティックネット上での推論は次のように行う。

'To whom did John give the book?'

という質問に対して次のユニット構造を作る。

```
x
    self : (element-of GIVING-EVENTS)
    giver: JOHN
    recip: y
    obj  : BOOK
```

これは figure 2.1 中の目標ネットの部分に対応している。この二つのネットを、別に作成した照合プログラム(セマンティックネットのインタープリタ)で照合することによって代入 {G1/x,MARY/y} を得る。

ところで Prolog は、(一階述語論理のサブセットである)ホーン論理におけるホーン節の集合をプログラムとみなした場合のインタープリタと考えられる。したがって知識をホーン節の集合(Prolog プログラム)で表現できれば、Prolog インタープリタは上記の知識インタープリタとして機能しうる。ここで、上記した Nilsson の方法と対比させるために、本稿で提案する DCKR で知識表現を行ってみる。DCKR の詳細な検討と記法の説明は3章以降で行う。figure 2.1 の事実ネットは (b)〜(e) のように表現される。

```
(a)  :- op( 100, xfx, ':').

(b)        sem( g1, giver: john).
(c)        sem( g1, recip: mary).
(d)        sem( g1, obj  : book).
(e)        sem( g1, PROPERTY) :- sem( giving_events, PROPERTY).

(f)        sem(  X, is_a : X).
```

(b) は、g1 の giver が john であることを表現している。(e) は知識の inheritance (継承)と関係しているが、詳細は3章で述べる。(f) は停止節であるが、これも3章で説明する。目標ネットは次の (g) のように表される。事実ネットと目標ネットとの照合は、(g) のゴールを実行することで置き換えることができる。

```
(g)  ?- sem(X, is_a:giving_events),
        sem(X,giver:john),
        sem(X,obj:book),
        sem(X,recip:Y).
```

(g) の実行により次の答えを得る。

```
X = g1
Y = mary
```

以上のことから、DCKR による知識表現を用いれば推論を行うためのインタープリタの中核を Prolog のインタープリタで代行することが可能なことが読み取れる。また DCKR の表現形式自体も自然で可読性もよいことが理解できよう。結論を先き取りすれば、DCKR により、従来の知識表現形式で試みられたいくつかの問題、知識の遺伝、部分-全体関係、手続き付加、default の扱いについても Prolog に組み込みの機能を用いて部分的に扱うことが可能である(3章、4章)。また DCKR の応用として、自然言語処理の意味解析用辞書 DCD(Definite Clause Dictionary)が検討されその有効性が確認されている[田中他 1985]。推論の効率に関しても知識を直接実行できるため大幅な改善が期待されるとともに、structured object の意味が Prolog プログラムの意味に帰着できるため論理的基盤が明確化する。

3. DCKR による知識表現

本章ではまず DCKR で重要な役割を果たす述語 sem について説明する。次にこれを用いてどのように階層構造を構成するか説明する。基本的な階層関係として is_a 関係と has_a関係を考える。また if-needed型の知識を DCKR でどのように表現するかを示す。最後にいくつかの論理的推論の例について述べる。

3.1 DCKR による知識表現のための基本述語 sem

DCKR では sem と呼ぶ基本述語を使用する。sem述語の典型的な形式は以下のようである。

```
sem( UnitName, SlotName:Filler )
```

具体的には、

```
(a)     sem( clyde#1, age:5 )
(b)     sem( elephant, color:gray )
(c)     sem( mammal, blood_temp:warm )
```

のように記述する。ここで第一引数の UnitName で '#' を持つもの (例えば (a) の clyde#1)は '個体(individual)' を表す。'#' を持たないもの(例えば (b) の elephant) は 'プロトタイプ(prototype)' を表す。プロトタイプ elephant は 集合 ELEPHANTS(figure 3.1)に属す個体に共通した性質を持つ典型的な(抽象的な)elephant を表すものである。

第二引数は、第一引数にある個体またはプロトタイプの '性質(property)' を表している。性質は、区切り記号 ':' によって分割される二つの部分からなる。左の部分、すなわち SlotName はこの性質の名前を表しており、右の部分、すなわち Filler は性質の値を表している。例えば (a) では個体 clyde#1 の年齢(age)が 5歳であることを表している。(b) では典型的な象 elephant は色(color)が灰色(gray)であることを表している。

3.2 is_a関係

3.1 の例で、個体 clyde#1 のプロトタイプが elephant であるならば clyde#1 はプロトタイプ elephant の性質も持つ。すなわち elephant が持つ、色が灰色であるという性質は clyde#1 が例外的に白い象であるなどの事実がわかっていない限り、clyde#1 にも継承されなければならない。また elephant のプロトタイプは mammal(哺乳類)であるから mammal の性質、温血動物であることも clyde#1 に継承されなければならない。これらの知識の継承(inheritance of knowledge)に関する問題をどのように扱ったらよいであろうか。

Nilsson は、知識の継承の問題を集合を用いて扱っている。例として Nilsson による figure 3.1 を考える。

[figure 3.1]

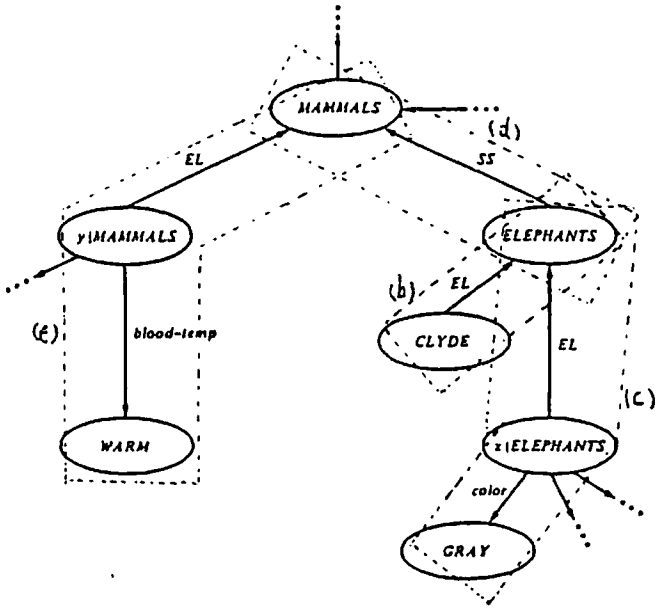


figure 3.1 では、個体 CLYDE が集合 ELEPHANTS に属すという事実を EL(element-of) リンクを用いて (b) のように表現している。 また集合 MAMMALS の部分集合として集合 ELEPHANTS があることを SS(subset-of)リンクを用いて (d) のように表現している。階層構造はこの二種類のリンクによって表現される。集合 ELEPHANTS に属すどのような個体も、色(color)が灰色(GRAY)であるという事実を x|ELEPHANTS(delineation と呼ぶ)という記法を用いて (c) のように表す。同様に(e) は MAMMALS に属すような任意の個体(y|MAMMALS)は、血液の温度(blood-temp)が暖かい(warm)ことを示している。

CLYDE の血液が暖かいことを証明するためには、次のような機能を持つプログラムが必要である。CLYDE の血液が暖かいことは CLYDE の性質として直接与えられていないので、まず CLYDE の ELリンクを矢印の向きにたどって ELEPHANTS のノードへ行く(b)。ELリンク(c)を矢印とは逆にたどってその先に blood-temp というこの性質が delineation により与えられていないことを確認したのち、SSリンク(d)を矢印の向きに、MAMMALS へ行く。さらにここから ELリンクを矢印とは逆向きにたどり、delineation y|MAMMALS の blood-temp リンクの先の値を取り出すことにより y に CLYDE が代入され成功する。リンクを上下にたどる以上の操作は必ずしも容易ではなく、そのためのインタープリタを別途作成する必要がある。

figure 3.1 と等価なものは、我々の方法では figure 3.2 のように記述される。

[figure 3.2]

```
:- op( 100,xfx,:),op( 90,xfx,#).

(a)   sem(X,is_a:X).

(b)   sem(clyde#1,PROPERTY) :- sem(elephant,PROPERTY).
(c)   sem(elephant,color:gray).
(d)   sem(elephant,PROPERTY) :- sem(mammal,PROPERTY).
(e)   sem(mammal,blood_temp:warm).
```

節(b) は個体 clyde#1 がプロトタイプ elephant であることを表しており、これは EL リンクに相当する。これは、elephant が性質 PROPERTY を持てば、clyde#1 もその PROPERTY を持つということを表しており、それにより elephant の性質がすべて clyde#1 に継承される。いいかえると上位からの知識の継承を素直に表現したものになっている。

節(d) も同様に elephant が mammal であることを表しており、これは SSリンクに相当している。この節によってプロトタイプ mammal の性質がプロトタイプ elephant に継承され、上記した例の場合にはそれがさらに clyde#1 にも継承される。

節(c),(e) はそれぞれ各プロトタイプがもつ性質を示している。後述するように節(a) は is_a関係を調べる場合に使用される停止条件節で、これにより上位概念の名前を第二引数に得ることが可能になる。

先程の CLYDE が温血動物であることは、次のゴールを実行することにより調べることができる。

```
?- sem(clyde#1,blood_temp:warm).
```

このゴール呼び出しとユニフィケーション可能なヘッドを持つ節は (b) のみであるから、PROPERTY に blood_temp:warm が代入されて、

```
sem(elephant,blood_temp:warm)
```

が新たなゴールとなる。これは CLYDE に与えられた知識だけでは問題が解決できないことから is_aリンクをたどって elephant の知識を使おうとしていることを意味している。

さらに、上のゴールとユニフィケーション可能な節は (d) であるから、

```
sem(mammal,blood_temp:warm)
```

が新たなゴールとなる。これは最終的に (e) によって成功する。以上の Prolog プログラム実行のトレースから節 (b)、(d) により、is_a関係による性質の継承が自然に表現できていることが理解できよう。

また次の形式の質問によってある個体の性質をすべて調べることができる。

```
?- sem(clyde#1,PROPERTY).
```

まず clyde#1 は clyde#1自身であるから (a) により、

```
PROPERTY = is_a:clyde#1 ;
```

が得られる。さらに (b) によって elephant へ行き、(a) によって、

```
PROPERTY = is_a:elephant ;
```

が得られる。この (a) の機能により、(a) を停止条件節と呼んでいる。以下同様にして、

```
PROPERTY = color:gray ;
PROPERTY = is_a:mammal ;
PROPERTY = blood_temp:warm ;
```

が次々に得られる。

またこの例とは逆に、ある性質を持つ個体をすべて選びだすこともできる。

```
?- sem(X#Y,is_a:mammal).
```

これを実行すると、

```
X = clyde
Y = 1
```

が得られる。もちろん、他にも条件を満たすものがあれば次々と得ることができる。
yes/no 型の質問も可能である。

```
?- sem(elephants,blood_temp:warm).

yes
```

またある個体の性質の FILLER について質問することができる。

```
?- sem(clyde#1,color:COLOR).

COLOR = gray
```

このようにいろいろな形の質問がすべて、双方向のユニフィケーションを行う Prolog インタープリターによって自然に実現されることがわかる。

ここで多重継承(multiple inheritance)の問題を考えてみよう。例えば jewel(宝石)は stone(石)であり、また同時に accessory(アクセサリ)であり、さらに fortune(財産)の性質も持っている。これらは、

```
sem(jewel,PROPERTY) :- sem(stone,PROPERTY).
sem(jewel,PROPERTY) :- sem(accessory,PROPERTY).
sem(jewel,PROPERTY) :- sem(fortune,PROPERTY).
```

と表現できる。これによって、jewel は三つの上位概念の性質をすべて継承することになる。ただし異なった上位のプロトタイプに矛盾した情報が書いてある場合についてはどのようにすべきかが問題である。しかしながら例えば、ookamiotoko は満月であれば wolf であり、そうでなければ man であるという事実を、継承する性質に条件をつけることで次のように表現することは可能である。

```
sem(ookamiotoko,PROPERTY) :-
        sem(moon#current,state:fullmoon),
        sem(wolf,PROPERTY).
sem(ookamiotoko,PROPERTY) :-
        not( sem(moon#current,state:fullmoon) ),
        sem(man,PROPERTY).
```

3．3 has-a関係

次に DCKR により has-a関係がどのように表現されるかについて述べる。例えば、

(mammals) -is_a-> (animals) -has_a-> (eyes) -has_a-> (pupil(瞳))

を考える。これは DCKR では figure 3.3 のように表現される。

[figure 3.3]

```
sem(X,is_a:X).

(a)   sem(mammal,PROPERTY) :- sem(animal,PROPERTY).
(b1)  sem(animal,has_a:eyes).
(b2)  sem(animal,has_a:X) :- sem(eyes,has_a:X).
(c1)  sem(eyes,has_a:pupil).
(c2)  sem(eyes,has_a:X) :- sem(pupil,has_a:X).
```

一つの has-a関係は二つの節で表現される。一つは (b1),(c1) のようなユニット節で性質名 has_a の値を直接かえしてくる。もう一つは (b2),(c2) のような節でこれにより has_a関係の継承を行っている。この has_a関係で継承すべきものは has_aの性質だけに限られるため、第二引数の性質名は has_a に固定される。これに (a) のような is-a関係も同時に考えると、上位のプロトタイプの持つ has_a関係も is_a関係を介して継承できることになる。例えば、

```
?- sem(mammal,PROPERTY).
```

を実行すると、

```
PROPERTY = is_a:mammal ;
PROPERTY = is_a:animal ;
PROPERTY = has_a:eyes ;
PROPERTY = has_a:pupil ;
```

が次々に得られる。

has_a関係の検索は、全体から(数量的に多い)部分への関係をたどるものであるから、探索空間が広がる。このことから使用にあたっては注意が必要であるが、ここでは has_a関係を sem述語で表現することが可能であることを示した。

3．4　if-needed型知識の表現

フレーム型の知識表現では手続き的な知識は普通 if-needed あるいは if-added のデモンを仮定して実現することが多い。これに対して Prolog系の論理的表現は静的な論理式であるとみることが可能であると同時に、プログラムとして考えると手続きであるとみることも可能である。このことからいわゆる手続き的知識の中で if-needed 型のものは非常に簡単に実現される。ここでは DCKR により if-needed 型の知識をどのように表現するかについて考える。(if-added 型については知識の更新などのメタなレベルの話題とも関係が深いことから今後の課題とする。)

if-needed 型の知識の例として、カモメのジョナサン(gull#jonathan)の年齢(age)を求めることを考える。gull#jonathan の age が直接知識ベースに与えられていれば、それを取り出せばよいので問題はない。ところが、gull#jonathan の age が直接存在しなければ、我々はどのように考えるであろうか。例えば gull#jonathan の生年月日から、それを計算するかもしれない。この推論を行うための知識は、この場合すべての生物に共通していると考えられるから、個々の個体ではなく上位のプロトタイプのところに与えておきたい。しかしながら 推論の実行時に必要となる gull#Jonathan の生年月日は gull#jonathan のところに書いておくべきものである。このようにすると、上位のプロトタイプ生物のところに付加された年齢計算プログラムから、gull#jonathan の年齢を知る機構が必要になる。すなわち、上位のプロトタイプに移行する際に呼び出しもとについての情報を持って行く必要があることがわかる。そこで我々の sem述語の形式を拡張し、第三引数に呼び出しもとの情報を保持しておくことを考える。

```
(a)  sem( time#current, year:1925, _).
(b)  sem( gull#jonathan, birthYear:1962, _).
(c)  sem( gull#jonathan, PROPERTY, X) :- sem( bird, PROPERTY, X).
(d)  sem( bird, PROPERTY, X) :- sem( livingThing, PROPERTY, X).
(e)  sem( livingThing, age:AGE, X) :-
            sem( X, birthYear:BIRTHYEAR, X),
            sem( time#current, year:YEAR, time#current),
            AGE is YEAR - BIRTHYEAR.


        ?- sem( gull#jonathan, age:AGE, gull#jonathan).
```

ageスロットは、直接与えられていないことからニニファイ可能な節は (c) のみである。さらに (d) に続いて 最終的には初めの第三引数 gull#jonathan が (e) の第三引数 X に束縛され、sem(X, birthYear:BIRTHYEAR, X) により、gull#jonathan の BIRTHYEAR を取り出すことができ、計算が行われ AGE に 23 が代入されて結果が得られる。

4. DCKR による不完全な知識の表現

ここではいわゆる不完全な知識について考える。矛盾した情報と default に関する問題及び、二つのレベルの否定に関する問題である。これらは相互に関係がある点である。

4.1 矛盾した情報と default

ここで述べる default による推論は、Moore によれば Autoepistemic Reasoning に近いと考えられる。すなわち例外についてもすべて記述されているデータベースの上で確実な推論を行うことを考える。

ある個体について、直接ある決まった情報が与えられていない場合には、すでに述べた is_a関係などによる上位のプロトタイプからの性質の継承により答えが与えられる。このようにして得られる情報は、default な情報であると考えられる。しかし、上位のプロトタイプから得られる性質が、すでに下位で知られている性質と矛盾している可能性がある。例えば、

```
sem(penguin.can_fly:no).
sem(penguin,PROPERTY) :- sem(bird,PROPERTY).
sem(bird.can_fly:yes).
sem(bird.has_a:wing).
sem(bird,has_a:X) :- sem(wing,has_a:X).
```

を考えると、

```
?- sem(penguin.can_fly:X).
```

という質問に対して、penguin についての知識から、

```
X = no
```

が得られるが、同時に bird に関する知識によって、

```
X = yes
```

も得られてしまう。この矛盾を解消するために次のようなカットオペレータを用いた表現を考える。

```
sem(penguin,can_fly:no) :- !.
sem(penguin,PROPERTY) :- sem(bird,PROPERTY).
sem(bird,can_fly:yes).
sem(bird,has_a:wing).
sem(bird,has_a:X) :- sem(wing,has_a:X).
```

この場合、先程の質問に対しては正しい答え(X = no)をかえすが、penguin の持つすべての性質を、

```
?- sem(penguin,PROPERTY).
```

という質問で引き出すことはできない(例えば has_a:wing)。また、

```
?- sem(penguin,can_fly:yes).
```

に対しては誤った答え(yes)をかえす。そこで質問の形式によらない一般的な方法を次にしめす。

```
sem(penguin,can_fly:no).
sem(penguin,PROPERTY) :- sem(bird,PROPERTY),not( PROPERTY = can_fly:X ).
sem(bird,can_fly:yes).
sem(bird,has_a:wing).
sem(bird,has_a:X) :- sem(wing,has_a:X).
```

この方法ではカットオペレーターは使用しない。基本形に対して変更点は not(PROPERTY = can_fly:X) の部分である。これは、相続できる PROPERTY に制限を加えているフィルターであり、上位のプロトタイプからの can_fly に関する性質の継承を防いでいる。この解決策は、Prolog ではホーン節のボディが左から右へ逐次的に実行されるという性質を利用している点に注意しなければならないが、このようにして Prolog インタープリタで default 知識が表現できることは意味があるであろう。

以上に説明した方法は closed world assumption のもとで、ホーン節を基本とするPrologインタープリター上に 'not (否定 : negation as failure)' と '= (ユニフィケーション)'を用いて実現されている。McDermott と Doyle による Nonmonotonic Logic では Mオペレーターを使用して定式化が行なわれた。Mオペレーターによる Mp は ¬ (p) であることが証明できないという意味をもっている。先程の例は、

$$\mathrm{bird}(X) \wedge M[\mathrm{can\_fly}(X)] \rightarrow \mathrm{can\_fly}(X)$$
$$\mathrm{penguin}(X) \rightarrow \neg(\mathrm{can\_fly}(X))$$

のように表現される。Nonmonotonic Logic においては、証明の順序が重要な意味を持つことはすでに指摘されている。そのため Nonmonotonic Logic の定理の定義は fixed point を用いて与えられた。

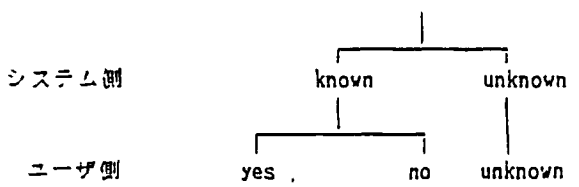
これに対して我々の場合には一般的な公理系について議論している場合と異なり、証明の順序は予想されている。フィルターを使用する場所はその性質が得られた直後である。したがって、上の例ではフィルターの存在自体が ¬ (can_fly(X)) がすでに証明されていることを意味している。いいかえればフィルターは、Mオペレーターが証明の過程において偽になることがあらかじめ予想される場所に埋めこまれることにより、Mオペレーターの働きを結果的にシミュレートするものである。Reiter による default logic においてはMは推論規則を適用する際の前提条件として導入されるが、これについても同様の解釈が有りうる。しかしながらフィルターと非単調論理との関係についてはさらに検討する必要がある。

4．2　二つのレベルの否定

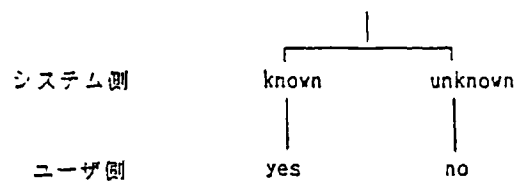
否定の知識の扱いには二種類ある。一つは肯定の知識と否定の知識の両方を与えておく方法である。Prolog は基本的には closed world assumption を採用していることから状態としては、 yes(known)と no(unknown) の二種類しか存在しない。しかしながら known の場合の値の解釈は利用者に解放されている。すなわち、これまで例として使用していた can_fly:yes(肯定) と can_fly:no(否定) もこの形式によっており、システム側では can_fly:no が否定であるとして認識しているわけではないが、利用者がこれを否定と解釈することは可能である。これは figure 4.1 のように表せるであろう。sem述語でこれが表現できることは 4.1 の例(can_fly:yes, can_fly:no)でもすでに明らかである。

もう一つの表現は全面的に closed world assumption を採用し、肯定の知識だけを持つ方法である。(figure 4.2)

[figure 4.1]



[figure 4.2]



sem述語ではこのような知識を次のように表すことができる。

```
sem(penguin,PROPERTY) :- sem(bird,PROPERTY),not( PROPERTY = can_fly:X ).
sem(bird,can_fly:yes).
sem(bird,has_a:wing).
sem(bird,has_a:X) :- sem(wing,has_a:X).
```

前と比較して変更された点は sem(penguin,can_fly:no) がなくなったところである。肯定の知識だけがゆるされ、明示的な否定の知識は消される。フィルターだけが残されることにより否定の知識が暗に表現される。この記法では penguin が飛ばないことは、

```
?- not( sem( penguin,can_fly:yes) ).
```

が成功することにより示される。

5．　sem述語と他の表現との関連

ここでさらに理解の便宜のために他の表現との関連も示しておく。sem述語は、自然に考えられる述語論理表現とは異なっている。例えば、個体 gull#jonathan が gull#jonathanJunior の父親(father)であることは、二項述語 father を用いて、

father(gull#jonathanJunior, gull#jonathan)

のように表現できる。しかしながら我々の方法では、

sem(gull#jonathanJunior, father:gull#jonathan)

のように表現される。ここで最大の相違点は、関係の名前(father)が述語名の部分に存在するか、引数の位置に有るかという点である。これによる利点は、関係の名前を変数にすることができる点である。もとの形で述語名を変数にすると高階の述語論理式になってしまうが、我々の方法では一階の述語論理式のままである。したがって例えば、gull#jonathan が gull#jonathanJunior にとってどのような関係にあるかを質問できるようになる。

```
?- sem( gull#jonathanJunior, X:gull#jonathan ).
```

この質問に対して、

```
X = father
```

という答えが得られる。このように、関係の名前を引数にとりこむことにより、質問の形式に柔軟性が得られる。sem述語は基本的には二項述語論理式に上記の変形を行ったものであり述語論理式との対応は明確である。

これまでの説明から明らかと思われるが、sem述語による知識表現はフレーム形式とも関係がつけられる。フレームにおける一つのスロットが、一つの sem述語に対応しているのである。第一引数はユニットの名前を表し、第二引数がスロット本体を表す。このように考えるとフレーム的な表現やセマンティックネット的表現を、DCKR の枠組で素直に実現できることがわかる。

6. まとめ

structured object の階層構造、性質の継承の表現が Prolog インタープリターの組み込みの機能で基本的には実現可能であることを示した。if-needed 型知識が自然に表現できることを示すとともに、それを用いた論理的推論の例を示した。また default の扱いも Prolog インタープリターの組み込みの機能で自然に実現可能であることを示した。

我々は現在 C Prolog interpreter 上で実験を行っており、したがって実行例も縦型探索によっている。現在様々な Prolog 処理系が提案されているが、DCKR と並列型 Prolog との関連も今後研究する必要がある。

また個体とプロトタイプを導入したが、集合をどのように扱うかを検討する必要がある。例えば、色が灰色であることは clyde のような elephant に属する個体が持つべき性質であるが、絶滅する可能性があるのは clyde のような個体ではなく、象という種(または集合)である。このような知識をどのように扱うべきかは、今後の課題である。

時間に関係する知識の扱いについても考える必要がある。さらに、動的にデータベース上の知識を変化させ、if-added型の知識に関する問題を考えたり、学習について考えることも興味ある課題である。

現在 DCKR の応用として自然言語の辞書項目記述形式 DCD を用いた自然言語処理の研究が行なわれている。DCD によれば、意味処理用のプログラムの中核を、Prolog に組み込みの機構で代用することが可能となる。さらにこれを応用した機械翻訳システムを検討中である。

謝辞

有益な助言をしていただいた ICOT の淵一博氏に感謝いたします。またいろいろ討論していただいた AIUEO の皆様にも感謝いたします。また日頃いろいろな面で協力していただいた東工大田中研究室の皆様にも感謝いたします。

参考文献

[Tanaka 82] : Tanaka,H.,
Semantic Representation Language, in Kitagawa, T.(ed.):Japan Annual Reviews In Electronics, Computers and Telecommunications, Computer Science and Technologies.
Ohm/North-Holland(1982),71-88.

[Nilsson 80] : Nilsson,N.J.,
Principles of Artificial Intelligence,
Palo Alto, Calif, Tioga(1980).

[Hayes 80] : Hayes,P.J.,
The Logic of Frame, in Metzing,D. ed:
Frame Conceptions and Text Understanding,
Walter de Gruyter, Berlin, New York,(1980),46-61

[Clocksin 81] : Clocksin,W.F., Mellish,C.S.,
Programming in Prolog, Springer-Verlag,(1981).

[McDermott 80]: McDermott,D., Doyle,J.,
Non-Monotonic Logic I,
Artificial Intelligence, Vol.13,
Special Issue on Non-Monotonic Logic, pp.41-72,
North-Holland, Apr. 1980.

[Reiter 80] : Reiter,R.,
A Logic for Default Reasoning,
Artificial Intelligence, Vol.13,
Special Issue on Non-Monotonic Logic, pp.81-132,
North-Holland, Apr. 1980.

[中島 85] : 中島秀之,
知識表現と Prolog/KR,
産業図書,(1985).

[田中 85] : 田中穂積, 池田光生, 奥村 学,
Definite Clause Dictionary,
Prolog による辞書項目記述と意味処理,
Proc.of ロジックプログラミングコンファレンス '85
(1985).