# 2章　知識表現形式DCKRの高速化と機能拡張

田中　穂積（東京工業大学工学部情報工学科）
山田　基司（ソニー株式会社生産技術本部）
奥村　学（東京工業大学大学院理工学研究科）

## 1. 序

田中らが開発した知識表現形式DCKR (Definite Clause Knowledge Representation)は，Prologのユニフィケーション機構をそのまま利用した知識の継承，手続き付加などが容易に行えるが，知識の量が増えるにつれて検索速度が低下するという問題がある[1][4]．この問題を解決するため，われわれはDCKRのもつ知識検索に関する一般性と整合性を保ち，しかも高速に動作可能な知識の新しい内部表現形式を開発したので，その概要を実験結果とともに説明する．この新しい内部表現形式は，Prologの宣言節として実現されており，それをどのように解釈して使うかを決める簡単なインタプリタとともに用いる．しかし，この内部表現形式で知識を直接記述することは，必ずしも容易でない．そこでSRL/Oと呼ばれる高水準の知識表現言語が設計されている[2]．SRL/Oを用いて記述された知識は，トランスレータにより内部表現形式に変換される．また，多数の世界を扱うことができるように機能拡張を施したが，その使用法を実例とともに示す．

## 2. DCKRによる知識表現

### 2・1 オブジェクトの表現

本章では，DCKRの簡略化版について説明する[1]．DCKRでは，オブジェクトを構成する各スロットを，semという単一の述語をヘッドとするホーン節の集合とみなす．たとえば，clyde#1は「象」で，「色は白」，象は「サイズが大」で「哺乳類」，哺乳類は「温血」であるという知識のDCKRによる記述は，(1a)，(1b)，(1c)，(1d)，(1e)のようになる．

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

sem(clyde#1, color:white).              (1a)
sem(clyde#1, P) :-
        isa(elephant, P).               (1b)
sem(elephant, size:big).                (1c)
sem(elephant, P) :-
        isa(mammal, P).                 (1d)
sem(mammal, bloodTemp:warm).            (1e)
```

述語isaの定義は以下のようである．

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

述語semの第一引数はオブジェクト名である．(1a)と(1b)，(1c)と(1d)，(1e)は，それぞれclyde#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と呼ぶことがある．オブジェクト名のうち#のついたものは個体名を，また#のつかないものはプロトタイプ名を表す．たとえば(1a)，(1b)のclyde#1は個体名であり(1c)，(1d)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）を表す．たとえば，(1a)と(1b)の記述は，clyde#1という個体を表し，(1c)，(1d)の記述は，elephantというプロトタイプを表す．

述語 sem 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，(1a) の記述は「個体 clyde#1 の color が white である」という事実を表すが，ここで color がスロット名で，white 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属性 (SV 対) と呼ぶ．

### 2・2　知識継承と推論

(1b) は，「clyde#1 は elephant である」という事実を表すが，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，(1b) は「elephant が性質 P をもてば，'yde#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．たとえば (1b) 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(clyde#1,P).                    (2)
```

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

```
P = color:white;                     (3a)
P = isa:elephant;                    (3b)
P = size:big;                        (3c)
P = isa:mammal                       (3d)
```

知識継承により，clyde#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).                        (4)
```

を実行すれば，X と Y の対としてすべての知識を出力しはじめる．また，Prolog の特徴を生かして，

```
?- sem(X,isa:mammal).                (5)
```

を実行すれば，

```
X = clyde#1;                         (6a)
X = elephant                         (6b)
```

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．

### 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す変数をもつ知識が 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呼ぶ．個体変数は一般に A#B のように表される．第一引数が個体変数の述語 sem をヘッドにもつ DCKR 表現は，その個体変数が全称限量化された知識であるから一般的な知識である．

文献（3）の中の一つの例を DCKR で記述し，その働きを調べてみよう．

```
sem(X#J,worksIn:Y#K) :-              (7a)
        sem(Y#K,isa:department),
        sem(Y#K,manager:X#J).
```

(7a) 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．

```
sem(joeSmith#1,worksIn:pd#1).        (7b)
sem(pd#1,manager:joeJones#1).        (7c)
sem(pd#1,P) :-                       (7d)
          isa(department,P).
```

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．(7c) 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，(7a) により「joeJones#1 が pd#1 で働いている」という事実が導出されていることに注意されたい．

### 2・4　DCKR の問題点

これまで説明してきた DCKR による知識の記述は，Prolog 組込みの機能を極力そのまま利用するという方針で設計されているため，知識のきわめて簡潔な記述が可能となっている．しかし，それゆえに検索時の効率が悪い．すなわち，すべての知識が sem という単一

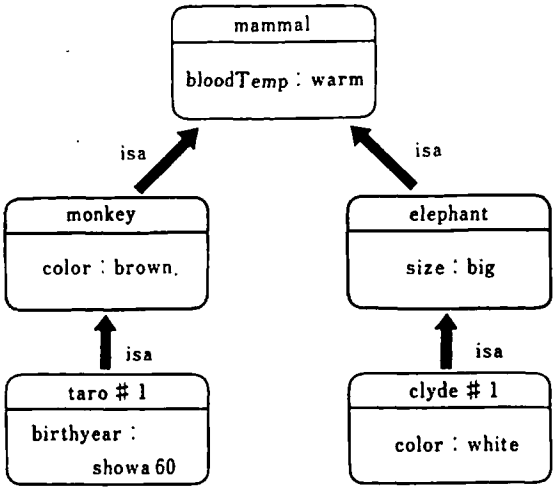


図 2・1　継承階層概念図

の述語の引数として記述されているため，知識の検索の際にはすべての知識とマッチングを行わなくてはならない．前述の（5）のようにオブジェクト名が変数となっている問合せの場合には，特に効率の低下が著しい．以下にその理由を述べる．いま知識として（1a）～(1e）があり，さらにその前に（0a)～(0d）の知識が存在するとしよう（図2·1）．

```
sem(taro#1, birthYear:showa60).          (0a)
sem(taro#1, P) :-
        isa(monkey, P).                  (0b)
sem(monkey, color:brown).                (0c)
sem(monkey, P) :-
        isa(mammal, P).                  (0d)
```

ここで（6）を実行する．

```
?-sem (X, size：big).                    (6)
```

述語名はすべてsemであるから，（6）の述語semは（0a)～(0d)，(1a)～(1e）のすべてのヘッドとマッチングを行う．まず，(0a）とマッチするが，taro＃1の属性はsize：bigとは異なるので，これは失敗する．しかし，taro＃1自身がsize：bigという属性をもっていなくても，taro＃1の親のクラスからその属性を継承する可能性がある．そこで，taro＃1の親のプロトタイプがこの属性をもつかどうかを調べるために，(0b）の述語isaによって継承階層を遡る．taro＃1は親としてmonkeyをもつことがわかるので，今度はmonkeyの属性を調べる．(0c）より，monkeyの性質としてcolor：brownが得られるが，これはsize：bigとは異なるので失敗する．しかし，monkeyは（0d）より，mammalの下位でもあるから，mammalの属性も調べなくてはならない．(1e）より，mammalの属性はbloodTemp：warmであるからこれも失敗する．ここで注意すべきことは，これまでの検索でmonkeyやmammalの属性を調べてはいるが，それらはあくまでもtaro＃1の属性を調べるために継承階層を遡ったもので，後にmonkeyやmammalに対して，それ自身の属性としてsize：bigがあるかどうかを調べるために再度検索が行われることがある．これがオブジェクト名が変数のときにDCKRが遅くなる主要な原因である．今の例でさらに検索を続けると，オブジェクトとしてtaro＃1を考えた場合はすべて失敗したので，次は，オブジェクトとして（0c）によりmonkeyを考えることになる．monkeyは属性としてsize：bigをもたず，またmonkeyの親のプロトタイプのmammalもそのような属性はもたないので，これもすべて失敗する．いいかえると，ここまでの実行はすべてむだであったということになる．

さらに実行を続けると，ここで初めてオブジェクトclyde＃1を選ぶことになる（(1a）より）．clyde＃1は属性としてcolor：whiteしかもっていないのでこの段階ではfailするが，clyde＃1の親のプロトタイプのelephantからclyde＃1はsize：bigを継承しyesという答えを得る．

このように，DCKRの実行では不必要なフレームとのマッチングを行い，そのたびにisa階層を遡るので速度の低下が起こる．後述するように知識の量が多くなればこれはきわめて大きな速度低下になる[4]．

## 3. 高速に動作可能な知識の内部表現形式

そこでわれわれは，DCKRの簡潔性を受け継ぎ，高速で動作する知識の内部表現形式を考案した．この形式の知識の検索はPrologで記述したコンパクトなインタプリタによって行う．このように，Prolog組込み

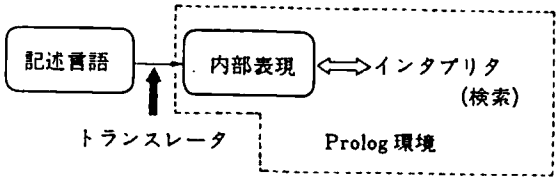


図 2・2　システム概要

の機能を利用しつつ，検索制御用のプログラムを用いる．また，内部表現形式をそのまま記述するのではなく，高水準の知識表現言語SRL/Oを用いる．システムの概要を図2·2に示す．それぞれの内容について以下の節で詳しく説明する．

### 3·1　知識の内部表現とインタプリタ

知識の内部表現例を示す．DCKRの(1a)に対応する知識は(8a1)～(8a3)，(1b)は(8b1)～(8b3)，(1c)は(8c1)～(8c3)のように表す．ここで，(8b1)～(8b3)は知識継承階層を表す内部表現である．

```
clyde(1, color,   white, p).          (8a1)
 olor(0, clyde#1, white, s).          (8a2)
hite(0, clyde#1, color, v).           (8a3)

elephant(0, =>, clyde#1, p).          (8b1)
clyde(1, <=, elephant, p).            (8b2)
isa(clyde#1, elephant, s).            (8b3)

elephant(0, size, big, p).            (8c1)
size(0, elephant, big, s).            (8c2)
big(0, elephant, size, v).            (8c3)
```

第一引数の数字は個体を識別するためのもので，述語名がプロトタイプのときには数字の0が入る．また，最後の引数のp，s，vは，述語名がそれぞれオブジェクト名，スロット名，スロット値であるということを表す．後述するように，これは知識検索の道筋に冗長性をもたせたことに等しい．たとえば，(8a1)，(8a2)，(8a3)では，オブジェクト名が既知の場合には(8a1)の知識を，スロット名が既知の場合には(8a2)の知識，値が既知の場合には(8a3)の知識を用いて検索を行う．(8b1)，(8b2)では，上位のオブジェクトelephantが既知で，その下位にどのようなオブジェクトが存在するかを検索したい場合には，(8b1)の知識を用いる．逆に，下位のオブジェクトclyde#1の上位にどのようなオブジェクトがあるかを知りたいときには(8b2)の知識を用いる．

このように，どの知識を用いて検索を行うべきかは，検索にあたり何が既知で何が未知であるかを調べることにより容易に決定することができる．それは，Prologで記述された小さな検索制御用プログラム（後述する述語con）を介して行う．一つの知識に対して，オブジェクト名，スロット名，スロット値のそれぞれを述語名にもつ3つの節として記述しているので，そのいずれを未知にして検索する場合でも，既知のものを述語名にもつ節を選んで，検索対象をすばやく取り出すことができる．それにより，検索の高速化を図ることができる．

具体的に（6）の実行では，スロット名が既知でオブジェクト名が未知（変数）であるから，スロット名sizeを述語名にもつ節（8c2）を用いて，対応するオブジェクト名を取り出して，(8b1)により（親から子への）知識継承階層の知識を用いて継承を下りながら，そのスロット値を継承するオブジェクト名を求めていくことができる．検索制御を行うためのトップレベルの述語はconと呼ばれ，オブジェクト名，スロット名，スロット値のそれぞれを既知のものとして検索するために，三つの述語con1，con2，con3を呼び出す．

```
con(X, S:V) :-
        is_object(X) -> con1(X, S:V);          (9a)
        nonvar(S)    -> con2(X, S:V);          (9b)
        nonvar(V)    -> con3(X, S:V).          (9c)
```

con1はX，すなわちオブジェクト名が変数でないときに呼び出され，以下を行う．

(i)　オブジェクト名を述語名にもつ節（内部表現）を探してマッチすれば，そのスロット名とスロット値の対を返して(ii)にいく．

(ii)　知識継承階層を表す内部表現があれば，継承を遡って親のオブジェクト名を取り出して(i)にいく．なければfailでリターン．

con2はXが変数で，Sが変数でないとき，以下を行う．

(iii)　スロット名を述語名にもつ節（内部表現）を探して，マッチすればオブジェクト名と，スロット値を返して(iv)にいく．

(iv)　オブジェクト名に対し知識継承階層を表す内部表現があれば，継承を下って（スロット値を引き継ぐ）子のオブジェクト名を得て(iv)にいく．なければfailでリターン．

con3はX，Sが変数で，Vが変数でないとき，以下を行う．

(v)　スロット値を述語名にもつ節（内部表現）から，当該スロット値をもつスロット名を求めて，それぞれのスロット名，スロット値の対に対して(iii)を実行する．

実行方法はトップレベルで，

```
?- con(clyde#1, X).
```

を実行すると (3a)～(3d) が，また，

```
?- con(X,isa:mammal).
```

を実行すると (6a)，(6b) が得られる．

### 3・2　内部表現形式を用いた知識検索の検討

2．で，DCKRは，オブジェクト名が変数の場合に特に効率が悪いことを述べたが，それでは本内部表現形式を用いると，それがどのように改善されるかを具体例で示す．2．で説明した知識を本内部表現形式で表すと，(8a1)～(8c3) の知識のほか，以下のようになる．

```
taro(1, birthYear, showa60, p).
birthYear(0, taro#1, showa60, s).
showa60(0, taro#1, birthYear, v).

.........

mammal(0, bloodTemp, warm, p).
bloodTemp(0, mammal, warm, s).
warm(0, mammal, bloodTemp, v).
```

実行結果は次のようになる．

```
?- con(X, size:big).     (10a)
X = elephant;            (10b)
X = clyde#1;             (10c)
```

Prologのトップレベルで (10a) を実行すると，オブジェクト名が変数 (X) であるので，述語conの本体ではスロット名のsizeに関する知識を探しにいく．前述のように内部表現形式はオブジェクト名，スロット名，スロット値のそれぞれを述語名とした知識をもっているので，sizeを述語名とした知識も存在するはずである．その結果 (8c2) の記述とマッチする．(8c2) の第二引数より，sizeというスロット名をもつオブジェクトとしてelephantがあることがわかり，そのスロット値bigは問合せ (10a) のスロット値と一致するので，これより (10b) の結果が得られる．elephantがsize：bigという属性をもつことがわかれば，elephantの下位もその属性を引き継ぐので，あとはisa階層を順に下ってelephantの子のプロトタイプ名（もしくは個体名）を得ればよい．すなわち，(8b1) よりelephantの下にclyde#1があることがわかるので，即座に (10c) を得る．clyde#1の下にはもうプロトタイプも個体もないのでfailする．さらにsizeを述語とする知識の記述はほかにはないので実行を終了する．このように，内部表現を用いた今回の方法では，size：bigという属性をもつオブジェクトを即座に得ることができ，あとはisa階層を下るだけで知識を次々に取り出すことができる．そのため，大量の知識が記述されていても必要な知識のみを参照し，しかもDCKRのように同じ知識を何度も重複して参照することがないため，高速化を図ることができる．本章で述べた内部表現形式を用いた場合に，どの程度の検索速度の向上が見込まれるかを知るために実験を行った．それについては4．で述べる．

### 3・3　本方式による一般的な知識の記述

2・3節で述べたような一般的な知識の表現は，本方式ではDCKRとまったく同じように記述する．すなわち，DCKRのsemをconに変えるだけでよい．

### 3・4　高水準知識表現言語SRL/O

(8a1)～(8a3)，(8b1)～(8b3) のような内部表現形式は，直接書くには煩雑であろう．より高水準の知識表現言語が必要である．われわれは，奥村が設計したSRL/Oを知識表現言語として用いる[2]．SRL/Oでは，(8a1)～(8a3)，(8b1)～(8b3) の知識を (11) のように書く．SRL/Oの形式で書かれた知識 (11) は，トランスレータにより (8a1)～(8a3)，(8b1)～(8b3) の内部表現形式に変換される．

```
clyde#1 ::
        [color:white]                    (11)
        [isa:elephant].
```

一般的な知識，すなわち，"Everyone who lives at Maple Street is a programmer"，といった知識は，DCKRでは (12) のように記述するが，SRL/Oでは (13) のように記述する．

```
sem(X#J, profession:programmer) :-
                sem(X#J, isa:human),
                sem(X#J, loc:mapleStreet).      (12)
```

```
X#J ::
        [profession:programmer
                if X#J isa:human,
                   X#J loc:mapleStreet].          (13)
```

## 4. 検索速度の比較

DCKRと今回作成したインタプリタの速度比較を図2·3～2·5に示す．縦軸は検索に要したCPU時間，横軸は知識の個数である．点線がDCKRを，実線が本内部表現形式を用いた場合の実測値である．むろん，継承の数やデータの種類によって速度は変化するが，定性的な傾向はおおむねこのようになる．図2·3～2·5から明らかなように，DCKRでは，知識の量が増えると検索時間が指数関数的に増えるのに対し，本方式では比例関係にある．したがって，知識の規模が大きいとき，もしくは（5）のように変数を含む知識検索を行う場合にはきわめて有効であることがわかる．また，2．でオブジェクト名が変数の場合，DCKRでは検索に時間がかかることを述べたが，実際この場合（図2·3）には他の場合と比べて検索のオーダが1桁上がる．

## 5. 機能の拡張——多重世界の導入

3．で説明した知識の内部表現形式では，知識の検索を行うときに，con1，con2，con3というインタプリタがつねに介在する．そのためのオーバヘッドが考えるが，4．で説明したように，知識の量が増えるにつれて検索速度の向上が著しい．知識の検索につねに介在するインタプリタがあることは，そこでさまざまな知識検索の制御を行う機会を与える．たとえば，その時機をとらえて，知識継承の際のオーバライドの機能を容易に実現することができる[5]．また，ワールドを導入することもできる[7]．これについて説明する．われわれのSRL/Oを拡張したワールド記述を（14）に示す．

```
in  ワールド名  {
    when {
        <precondition>
          }                                      (14)
    SRL/Oによる知識の記述  または
    （サブ）ワールドの定義
}
```

すべてのワールドは，ルートワールドの下に作られる．むろん，ワールドは何重にもネストできる．事実を記述するときに，ワールドを指定しなければ，ルートで成り立つものとみなされる．ワールド間の上下関係は，定義の際に明示的に与えなければならない．検索の際には，トップレベルで，

```
?- in(ワールド名).
```

として，あるワールドにはいると，そのワールドと，そのワールドの親のワールドの内容だけがみえるようになる．検索はサブワールド優先で行われる．また，ワールドを選ぶために，"when" 文中に〈precondition〉が存在する．〈precondition〉は，そのワールドが満たさなければならない前提条件ではなく，あるワールドにはいるための条件，いいかえるとガードとしての役

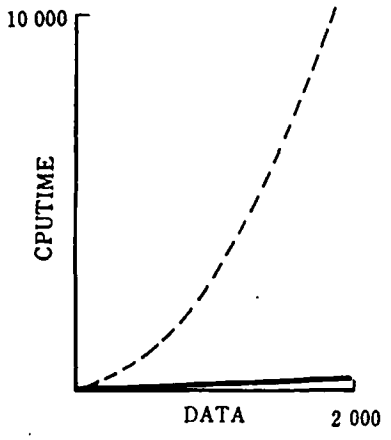

図 2·3　プロトタイプ名が変数の場合

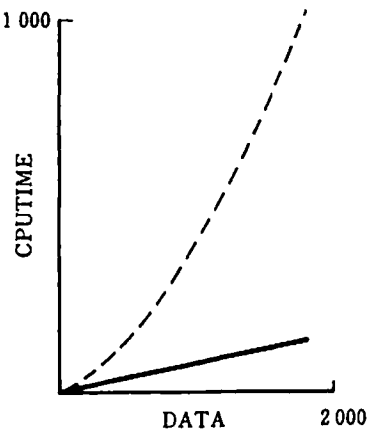

図 2·4　スロット名が変数の場合

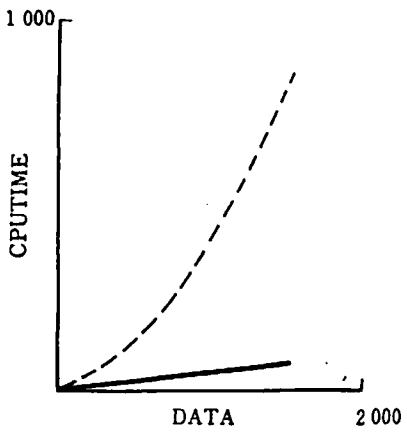

図 2·5　スロット値が変数の場合

割を果たす．すなわち，トップレベルで，

```
?- when(<precondition>).
```

とすると，<precondition> を満たすワールドに入る．

以下に記述例を示す．

```
champ ::        [points:10].
second ::       [points:6].
third ::        [points:3].

hoshino#1 ::    [team:leytonHouse].
matsumoto#1 ::  [team:cabin].
takahashi#1 ::  [team:advan].
takahashi#2 ::  [team:sagawakyubin].
nakajima#1 ::   [team:epson].
hagiwara#1 ::   [team:leytonHouse].

in race#1 {
        when {  time:: [7~7].
                place:: [circuit:suzuka].
                place:: [prefecture:aichi].
        }
        hoshino#1::     [isa:champ].
        matsumoto#1::   [isa:second].
        takahashi#2::   [isa:third].
}

in race#2 {
        when {  time:: [8~8].
                place:: [circuit:fuji].
                place:: [prefecture:shizuoka].
        }
        nakajima#1::    [isa:champ].
        hagiwara#1::    [isa:second].
        matsumoto#1::   [isa:third].
}

in race#3 {
        when {  time:: [9~9].
                place:: [circuit:sugo].
                place:: [prefecture:sendai].
        }
        takahashi#1::   [isa:champ].
        hoshino#1::     [isa:second].
        nakajima#1::    [isa:third].
}
```

上の記述はレースの結果をワールドを用いて表したものである．1位，2位，3位のポイントやエントリしているレーサとそのチーム名はどのレースでも同じであるが，レースごとに順位は異なる．そこで，どのレースにも共通な事項をルートのワールドに記述し，その下に各レースのワールドを作り，そこに結果（順位）を記述している．次に実行例を示す．

```
| ?- when(place, circuit:fuji),
     con(X, isa:champ).

W = g¥race#2,
X = nakajima#1 ;

no
| ?- when(time, 9, W),
     in(W),
     con(hoshino#1, P).

W = g¥race#3,
P = team:leytonHouse ;

W = g¥race#3,
P = isa:second ;

W = g¥race#3,
P = points:6 ;

no
| ?- in(g¥race#N),                    (8)
     con(X, isa:champ).


N = _1,
X = hoshino#1 ;

N = _2,
X = nakajima#1 ;

N = _3,
X = takahashi#1 ;
```

ワールド名にも，個体と同じようにナンバをつけることができる．（8）は，（レースの）ワールドのナンバを変数にして検索を行っているが，こうすれば，バックトラックにより自動的にすべてのレース結果を得ることができる．

## 6. おわりに

DCKRのもつ知識検索に対する一般性と整合性を保ち，かつ高速な知識表現方式を提案した．知識の量が大きくなるほど，検索速度の差は大きくなる．現在，

多重知識継承の実現について考察中である．また，大規模なシステムへの実際の適用を検討している[6]．

## 文　献


（1）田中穂積，小山晴生，奥村学："知識表現形式DCKRとその応用"，コンピュータソフトウェア，3，4 (Oct. 1986)

（2）奥村学（他）："意味記述用言語SRL/Oの設計とDCKR"，情報処理学会情報学基礎研資，1-2 (1986)

（3）Nilsson, N. J.："Principles of Artificial Intelligence", Palo Alto. Calif, Tioga (1980)

（4）赤間清："継承階層PROLOGと多重継承"，日本ソフトウェア科学会第3回大会論文集（1986）

（5）Stefik, M. and Bobrow, D, G.："Object-Oriented Programming: Themes and Variations", The AI Magazine, 6, 4, pp. 40-62 (1986)

（6）Lenat, D., Prakash, M. and Shepherd, M.："CYC：Using Common Sense Knowledge to Overcome Briteness and Knowledge Acquisition Bottlenecks", The AI Magazine, 6, 4, 65-85 (1986)

（7）中島秀之："多重世界機構の論理的意味"，日本ソフトウェア科学会第3回大会論文集（1986）

# 付録　知識検索制御用プログラム

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

/** interpreter **/

con(X, S:V) :-
        is_instance(X)  -> con1(X, S:V);
        nonvar(S)       -> con2(X, S:V);
        nonvar(V)       -> con3(X, S:V).

con1(XX, S:V) :-
        (add0(XX, X#N), match(X, N, S, V, n));
        con11(XX, S:V).
con11(XX, SV) :-
        add0(XX, X#N),
        up_down(X, N, <=, Upper),
        (SV = isa:Upper;
         con1(Upper, SV)).

con2(X, isa:V) :-
        match(isa, 0, X, V, s),
        con1(X, isa:X).
con2(XX, S:V) :-
        match(S, 0, X, V, s),
        (XX = X;
         con22(XX, X, S:V)).
con22(XX, X, S:V) :-
        add0(X, X1#N),
        up_down(X1, N, =>, Lower),
        add0(Lower, Lower1#NN),
        (match(Lower1, NN, S, _, p) -> fail;
                                       (XX = Lower;
                                        con22(XX, Lower, S:V))).

con3(X, S:V) :-
        setof(S, match(V, 0, _, S, v); P),
        con33(P, X, S:V).

con33([], X, S:V) :- !, fail.
con33([SH|SR], X, S:V) :-
        (con2(X, SH:V), S = SH);
         con33(SR, X, S:V).

/* match */
match(Arg1, Num, Arg2, Arg3, Type) :-
        C =.. [Arg1, Num, Arg2, Arg3, g, Type],
        call(C).

/* up_down */
up_down(Arg1, Num, Direction, Arg2) :-
        C =.. [Arg1, Num, Direction, Arg2, g],
        call(C).

/* add0 */
add0(X, XX#N) :-
        atom(X) -> XX#N = X#0; XX#N = X.

/* is_instance */
is_instance(X#Y) :-
        !, nonvar(X).
is_instance(X) :-
        nonvar(X).

/* end */
```