Vol. 28 No. 4 情報処理学会論文誌 Apr. 1987

PAL: 継承階層を扱う拡張 Prolog†

赤間 清††

PAL は，継承階層を扱う拡張 prolog である．それはクラス束縛変数と呼ばれる新しいパターンを採用している．クラス束縛変数は，既存の prolog システムの制約付き変数，例えば，prolog II の「凍結された」ゴールを持つ変数や項記述 (Nakashima 1985) と異なり，推論中に他のクラス束縛変数と単一化されると，その束縛のクラスが変化する．その単一化は，例えば，継承階層のなかの兄弟のクラスの排反性などの，継承階層に課せられた制約を用いて効率的に実行される．クラス束縛変数と高速な単一化の方法によって，ほかの prolog システムでは全数探索によって引き起こされ得る組み合わせ爆発が抑制され，推論の効率が大幅に改善される．

1. まえがき

PAL は，継承階層を扱う特別な機構と，それに基づく拡張ユニフィケーションを導入した拡張 prolog である．ここで継承階層とは，概念の包含関係，部分全体の関係，支配非支配の関係などのなす階層関係の総称である．

継承階層，特に，概念の階層構造を prolog のなかにうまく取り込もうとする試みは，例えば，prolog/kr の多重世界[7] や，項記述[6],[8]，ユニフィケーションの拡張[5]，DCKR[4] による知識表現など，多方面からなされてきた．このうち DCKR は，現在の標準的な prolog の範囲内での表現の試みであり，他は，prolog をなんらかの一般的な枠組みで拡張し，その結果を概念階層の表現に応用するものである．これに対して本研究では，継承階層のための特別な機構を導入して prolog を拡張する．

継承階層に対する特別の工夫は，常識を扱う大規模で本格的なシステムを構築しようとするとき，どうしても必要になる．継承階層に関する推論を行うとき問題となるのは，全数探索による探索量の爆発である．既存の方法では階層構造の持つ性質を十分に利用していないために，本来ならば避けることのできる全数探索を回避できない場合がある．それは扱う対象の数が多ければ多いほど加速度的に無駄な探索を増加させる．全数探索を回避しなければ，本格的なシステムは事実上動かないと言える．

継承階層の情報を十分に利用することによって，PAL は，全数探索を避け推論速度を大幅に向上させる．またそれは，知識の表現の観点から言っても，直感にかなった素直な方法を提供する．我々の方法は，prolog を極めて自然に拡張するものである．継承階層を扱うこと自体は，lisp ベースでも，object 志向でも，その他任意の言語上で可能である．しかしながら本方法は prolog の中でこそ最大の有効性を発揮する．拡張の実現は簡単であり，しかも基礎となる prolog の処理速度を遅くすることなく継承階層の推論機構を付け加えることができる．

継承階層は，クラス束縛変数を通して利用される．クラス束縛とは，変数に，それがユニファイ可能なクラスを指定するものである．例えば，

*xyz^animal

はクラス束縛変数の例であり，変数 *xyz がクラス animal に束縛されている．したがって以後，animal の部分クラスである dog に束縛し直されることはあっても，animal と排反なクラスである plant に束縛し直されることはない．

クラス束縛変数は，制約付き変数の一種と見ることができる．しかしそれは次の意味で，既存の制約付きの変数[3],[9] 以上のものである．既存の制約付き変数は，制約の動的な変化は扱っていない．PAL の継承階層では，制約は互いに相互作用し，動的に変化して行く．またそれらの相互作用はユーザが指定でき，その指定に基づいて極めて小さなコストで実現される．

PAL は lisp や prolog/kr と同様に，S 式のシンタックスを採用している．S 式のシンタックスは他のものより単純で統一的であり，理論的にも扱いが楽である．本論文では S 式のシンタックスを基本にして説明する．それを prolog の別のシンタックスやデータ構造に一般化することは容易であろう．

† PAL: Extended Prolog with Class Bound Variables for Inheritance Hierarchy by KIYOSHI AKAMA (Department of Behavioral Science, Faculty of Letters, Hokkaido University).

†† 北海道大学文学部行動科学科

2. 継承階層の記述とその意味

概念の包含関係，部分全体の関係，支配非支配の関係など，常識の世界などに現れるいろいろな階層関係は，知識表現において大きな役割を果たす．その中で概念の階層構造は，最も広範で重要な応用を持つが，PAL が扱う対象は概念階層に限定されない．したがって我々は，より一般的な継承階層という単語を用いる．ただし本論文では概念階層に関する例だけを述べ，他の階層構造への応用の方法は別の論文に譲る．

継承階層の表現においては，クラスとインスタンスという二つの概念を区別する．それらはともにシンボルであり，それぞれ有限個しか存在しないと仮定する．シンボルは，クラス名であるか否か，インスタンス名であるか否かにより4種類に分かれる．このうち，クラス名であり，しかも，インスタンス名であるものは，特殊な使い方をするが，本論文ではそれに言及しないので，以下では存在しないとみなして記述する．

クラスとクラスの関係は，組み込み述語 asc（これは assert class の略である）を用いて，

(asc 親クラス名　子クラス名)

のように書く．例えば，人間が男性と女性からなることは，

(asc human man)
(asc human woman)

のように記述される．また，クラスとインスタンスの関係は，組み込み述語 asi（これは assert instance の略である）を用いて，

(asi 親クラス名　子インスタンス名)

のように書く．例えば，太郎，次郎，三郎が男性であることは，

(asi man taro)
(asi man jiro)
(asi man saburo)

と記述される．

継承階層は，asc と asi の式の有限集合 E によって表現する．ただし E は次の2条件（◎と○）を満たしているものとする．

◎ 表現 E に付随するグラフは，ループをもたない．

○ asc と asi の両方の式の第1引数となるクラスは存在しない．

ここで，E に付随するグラフとは，E に出現するクラスやインスタンスを節 (node) とし，E のすべての asc や asi の式に対して，その第1引数（の節）から第2引数（の節）に至る（有向）辺 (arc) を持つグラフである．以後 E に付随するグラフを系図になぞらえて節の関係を表現する．例えば，asc の第2引数（のクラス）は第1引数(のクラス）の子クラスであるという．

集合 E に，◎と○のほかにどのような条件を課すかによって，知識表現の自由度や表現される内容は変化する．ここでは次の2条件（☆と★）を追加する．本論文の目的は，典型的な構造に対する自然な知識表現と，十分な推論速度を達成するための中心的なアイデアを与えることなので，条件は十分きつくしてある．

☆ 付随するグラフは，一つの（根付き）木である．

★ 任意のクラスの異なる子クラスは排反である（と解釈する）．

根付き木[1]（arborescence）とはグラフ理論の用語であり，根のある木を意味する．しかし情報科学の分野では木という用語を根付き木の意味で使うことが多い．したがって以下では，簡単のために根付き木を単に木と略記する．さて，上の☆で付随するグラフを，n 個ではなく，1個の木としたことについて説明をしておく．n 個の木があるとき，それらをまとめた新たなクラスを導入し，n 個の木を1個の木にまとめることができる．もし n 個の根に対応するクラスが排反ならば，上記の変換は★の解釈も含めて等価変換になり，一般性を失うことなく1個の木で考察できる．もし排反でないならば，★の解釈に関して不都合がおこる．その解決の方法は別の論文で述べる．

表記の便宜上，asc や asi の式のうち，第1引数が同一のものをまとめて記述できるように，組み込み述語 defc と defi（それぞれ define class, define instance の略）を導入する．それらの一般形は，

(defc 親クラス名　子クラス名のリスト)
(defi 親クラス名　子インスタンス名のリスト)

である．これを用いれば，例えば，

(defc human (man woman))
(defi man (taro jiro saburo))
(defi woman (hanako miyoko))

などと書くことができる．以下の説明では，簡単のために，defc と defi だけを用いる．

継承階層を表す表現 E において，E（に付随するグラフ）に出現する任意のクラス（節）を s とする．そ

のとき，s の子孫のインスタンス全体の集合 S が一意に定まる．このように，継承階層を仮定すれば，クラス名を表すシンボルを使うことによって，ある種の集合を簡単に指定できる．

3. クラス束縛変数とユニフィケーション

継承階層を prolog での推論に利用するために，クラス名で束縛された変数の概念を導入して，prolog の変数の概念を拡張する．新しく追加される変数は，プログラム上では，

変数名^クラス名

の形で記述される．これは「変数名」で示される変数が（マッチングの全対象を，変数を含まない S 式の範囲に限定して考えた場合）「クラス名」に対応する集合に含まれるインスタンスとしかマッチしないという制約を持つことを示している．我々はこのような変数またはこのような記述全体をクラス束縛変数と呼ぶことにする．例えば，

*xyz^woman

は任意の woman（すなわち，woman というクラスに属する任意のインスタンス）とだけマッチするクラス束縛変数（PAL では * で始まるシンボルが変数を表す）である．標準的な prolog の変数は，任意の S 式とマッチしうる変数だけであり，マッチする対象に対する制限を変数自体が持つことはない．したがってクラス束縛変数の導入は prolog の拡張を引き起こす．

上の例は，prolog の拡張の試みの一つである項記述[6),8)]を想起させる．項記述は，例えば，

*xyz：woman

を任意の woman とだけマッチする項とみなすことを可能にする．宣言的意味に限れば，これら二つの表現は同じと考えてよい．ただし woman の定義は，PAL では継承階層によって与えられるクラスであり，項記述では woman という一般の1引数述語によって与えられる．項記述とのより重要な違いの一つは，その手続き的意味，すなわち，推論の方法と速度（5.3節参照）にある．それはユニフィケーションの方法の違いによってもたらされる．

継承階層を利用したユニフィケーションは次のように定義される．ユニフィケーションの対象となる項は，コンス，定数アトム，通常変数，クラス束縛変数の4種類である．このうちクラス束縛変数だけが新しく追加された項であり，それ以外の項どうしのユニフィケーションは通常の prolog と同様である．したがって，以下ではクラス束縛変数 $*x$^class$x$ と4種の項のユニフィケーションを定義する．

1：コンス（car . cdr）との場合．
インスタンスにはアトムだけを許しているので失敗する．
unify($*x$^class$x$, (car . cdr)) ⟶失敗

2：定数アトム a との場合．
a が class$x$ に含まれるなら成功．$*x$ は a に束縛し直される．
a が class$x$ に含まれないならば失敗．
unify($*x$^class$x$, a)
⟶$*x=a$ …if $a\in$class$x$
⟶失敗 …if $a\notin$class$x$

3：通常変数$*y$ との場合．
成功し，$*y$ も $*x$ と同じクラス束縛を受ける．
unify($*x$^class$x$, $*y$)
⟶$*x=*y$, $*x$^class$x$, $*y$^class$x$

4：クラス束縛変数 $*y$^class$y$ との場合．
二つのクラスに包含関係が成り立つとき成功．
狭い方のクラス束縛になる．
二つのクラスに包含関係が成り立たないとき失敗．（実際には排反の場合しかない）
unify($*x$^class$x$, $*y$^class$y$)
⟶$*x=*y$, $*x$^class$x$, $*y$^class$x$
…if class$x\subset$class$y$
⟶$*x=*y$, $*x$^class$y$, $*y$^class$y$
…if class$x\supset$class$y$
⟶失敗…if class$x\cap$class$y=\phi$

常識的な継承階層を仮定して，ユニフィケーションの簡単な例を挙げる．

unify($*p$^human, (1 2))⟶失敗
unify($*p$^human, adam)⟶成功：$*p$=adam
unify($*p$^woman, adam)⟶失敗
unify($*p$^human, $*q$)
⟶成功：$*p=*q$, $*p$^human, $*q$^human
unify($*p$^woman, $*q$^human)
⟶成功：$*p=*q$, $*p$^woman, $*q$^woman
unify($*p$^human, $*q$^woman)
⟶成功：$*p=*q$, $*p$^woman, $*q$^woman
unify($*p$^woman, $*q$^man)⟶失敗

4. 継承階層を用いた知識表現

継承階層以外の任意の知識は，述語 asp, defp などを用いて表現する．その時，defc や defi などを用い

て定義した継承階層（のクラス）を利用することができる．DCKR による表現と比較して説明する．

DCKR による次のような表現を考える．

(a) sem(clyde#1, *P*) :- sem(elephant, *P*).
(b) sem(clyde#2, *P*) :- sem(elephant, *P*).
(c) sem(clyde#3, *P*) :- sem(elephant, *P*).
(d) sem(elephant, *P*) :- sem(mammal, *P*).
(e) sem(dog, *P*) :- sem(mammal, *P*).
(f) sem(cat, *P*) :- sem(mammal, *P*).
(g) sem(mammal, *P*) :- sem(living_thing, *P*).
(h) sem(plant, *P*) :- sem(living_thing, *P*).
(i) sem(*X*, is_*a*: *X*).
(j) sem(clyde#2, birth_year: 1962).
(k) sem(elephant, color: gray).
(l) sem(mammal, blood_temp: warm).
(m) sem(time#current, year: 1986).

これは，継承階層を用いれば次のように書ける．

(1) (defc living_thing (mammal plant))
(2) (defc mammal (elephant dog cat))
(3) (defi elephant (clyde1 clyde2 clyde3))
(4) (defp birth_year ((clyde2 1962)))
(5) (defp color ((*^elephant gray)))
(6) (defp blood_temp ((*^mammal warm)))
(7) (defp year ((time_current 1986)))

(a)，(b)，(c)は，elephant の具体例を記述しており，(3)にあたる．DCKR ではインスタンスは#というファンクタで示されるが，PAL の継承階層では defi の第2引数の要素ということで識別される．(d)，(e)，(f)は mammal を上位クラスとする継承関係の記述であり，(2)にあたる．(g)，(h)は living_thing を上位クラスとする継承関係の記述であり，(1)にあたる．(1)，(2)，(3)の順序は任意でよい．ここでは，上位クラスから順に定義した．defc や defi のシンタックスにはそれが自然であろう．DCKR では排反性（例えば，mammal と plant の排反性）は表現されていないのに対して，PAL の継承階層では排反性を前提として表現している．

(i)の述語 is_*a* は継承階層の下位-上位関係を記述する．それを PAL では，subclass_of, instance_of という二つの組み込み述語で実現している．述語 subclass_of は defc で宣言された知識を基礎としてクラス間関係を，述語 instance_of は defi と defc で宣言された知識を基礎としてインスタンス-クラス関係を記述する．例えば，

(subclass_of dog living_thing)
(instance_of clyde3 mammal)

が成り立つ．また

(subclass_of *class mammal)
(subclass_of cat *class)
(instance_of clyde2 *class)
(instance_of *instance living_thing)

などとして，変数部分にあてはまるクラスやインスタンスを後戻り（backtrack）によって次々に求めることもできる．subclass_of や instance_of それ自身には階層関係を新しく宣言する機能はない．

(j)から(m)までは，それぞれ(4)から(7)までに意味的に対応した述語を定義している．defp は prolog/kr の define とほぼ同じである．PAL は define と defp の両方を備えている．両者の違いは，*^mammal のような継承階層を用いたクラス束縛変数を，define では利用できないのに対して，defp では利用できることである．クラス束縛変数を含む述語を定義するためには，クラス束縛変数を含まないことが分かっている場合に比べて，処理時間が多くかかる．define はクラス束縛変数を扱わないことを前提として定義実行の高速化を図っている．(4)はクラス束縛変数を含まないので，述語 define を用いて，

(4)′ (define birth_year ((clyde2 1962)))

と書くこともできる．(5)は，すべての elephant の色が gray であることを，(6)は，すべての mammal が温血動物であることを述べている．(7)は，現在が 1986 年であることを示している．

次の例は，生物の年齢を計算するには，今年からその生物の生まれた年を引けばよいことを素直に表現している．

```
(defp age
    ((*^living_thing *age)
     (year time_current *year)
     (birth_year *^living_thing *birth_year)
     (— *year *birth_year *age)))
```

また，質問も

```
(age clyde2 *age)
```

と簡単である．これらは完全に宣言的で，直感にかなっている．

例外がある知識の記述について考える．例えば，鳥は普通飛ぶが，ペンギンは飛ばない．これに関連する知識は，例えば，

```
(defc animal (bird fish mammal))
```

```
(defc bird (penguin hawk eagle))
(defp can_fly ((*^bird) (not (can_not_fly *))))
(defp can_not_fly ((*^penguin)))
(defp has_a ((*^bird wing)))
(defp move ((*^animal)))
```

と書けばよい（ただし利用方法には制限がある）．第3式は，述語 can_not_fly を満たさない鳥は can_fly であると書いてある．この can_not_fly は can_fly の例外を示す知識である．したがって第3式は，例外が書かれていない限り鳥は can_fly である，と読める．例外の記述は第4式にある．そこでは，penguin はすべて例外だ，と述べている．

5. 推論速度

5.1 DCKR との比較

PAL の継承階層とクラス束縛変数を利用すれば，推論の速度を大幅に改善できる場合がある．例えば次の問題[2]を考えてみる．

「自転車と自動車はともに乗りものである．bic の1から n までは自転車である．car の1から n までと mycar は自動車である．乗りものはタイヤを持ち，自動車はドアを持ち，私は mycar を所有している．このとき，タイヤを持ち，ドアを持ち，私が所有しているものを求めよ．」

いくつかの条件を満たすものを求めるという意味で，これは prolog の典型的な問題の一つである．この問題を DCKR と PAL の方法で表現し（以下の表現は $n=3$ の場合を示す），それぞれの場合に，n をいろいろ変えて解を得る所要時間を比較する．ただし，表現と実行はすべて PAL の上で行う．

［1］ DCKR の場合の表現

前提となる知識は次のように書ける．PAL ではファンクタを扱わないので，インスタンスをアトムのリストで表現して，クラスと区別した．同様に，sem の第2引数も長さ2のリストで表現している．

```
(as (sem bicycle *) (sem vehicle *))
(as (sem (bic1) *) (sem bicycle *))
(as (sem (bic2) *) (sem bicycle *))
(as (sem (bic3) *) (sem bicycle *))
(as (sem (car *) (sem vehicle *))
(as (sem (car1) *) (sem car *))
(as (sem (car2) *) (sem car *))
(as (sem (car3) *) (sem car *))
(as (sem (mycar) *) (sem car *))
(as (sem vehicle (has tires)))
(as (sem car (has doors)))
(as (sem (mycar) (own I )))
```

ここでは概念階層の記述を先に，他の性質を後にして記述した（これが最もよい順と言うわけではない）．最後に，問題は次のように書ける．

```
(as (test *) (sem * (has tires))
        (sem * (has doors)) (sem * (own I )))
```

［2］ PAL の継承階層を用いた表現

継承階層を使えば，この問題は次のように表現できる．

```
(defc vehicle (bicycle car))
(defi bicycle (bic1 bic2 bic3))
(defi car (car1 car2 car3 mycar))
(defp has ((*1^vehicle tires)) ((*2^car doors)))
(defp own ((I mycar)))
(defp test ((*) (has * tires)
        (has * doors) (own I *)))
```

これは人間の直観にあっており，読みやすく，書きやすい．

問題のパラメータ n と表現方法を変えて，所要時間（単位：秒）を比較すると**表1**が得られる．PAL の優位は歴然としている．n が大きくなると，DCKR は加速度的に所要時間が増加するのに対し，PAL は，まったく変化しない．問題の規模が大きいほどそれらの差は（急激に）開いていく．ただし，節のインデキシングを行うシステムでは，増加の速度は表1よりも低く抑えることができる．しかしここでの比較は所要時間のオーダが n 以上か否かだけを問題にしているので，結論は変化しない．n の数は知識の量を表していることに注意せよ．DCKR は，獲得知識の増加とともに，ますます推論の所要時間がかかるという矛盾を示している．

DCKR の方法において，解の候補となるクラスやインスタンスは（節の順序を上のようにするとき）次のように変化する．

vehicle,
bicycle, bic1, bic2, bic3,

表1 各表現方法ごとの求解の所要時間（単位：秒）
Table 1 Execution time to obtain the first solution in each method (in sec.).

n		3	6	9	18	27
(1)	DCKR	2.40	6.10	11.00	63.00	210.00
(2)	PAL	0.04	0.04	0.04	0.04	0.04

car, car1, car2, car3, mycar

これらが与えられた条件を満たすかを順に全数チェックするのが，DCKR における解の探索の実態である．この順序は，上位クラスへの継承の知識やその他の知識を表現する sem 節の順序によって決まっている．したがって，sem の節の順序を変えると探索の順番をある程度自由に変えることができる．もし

(as (sem (mycar) *) (sem car *))

を一番前にするなら，mycar という解を発見するまでの所要時間はずっと少なくてすむ．しかし，すべての問題に対してそのようにうまくいく順番は存在しない．また，全解を求める場合の全所要時間は順番を変えても短縮できない．

DCKR が上記のような問題において多くの推論時間を要するのは，盲目的な全数探索が原因である．その探索順は，チェックされるべき条件によらず一定であり，柔軟性を欠いている．それらの全数探索は，扱う対象の数が多ければ多いほど加速度的に無駄な探索を増加させる．

5.2 PAL の推論方法

PAL はこの無駄を，クラス束縛を用いて解消する．具体的に上記の述語 test を逐次実行してみれば，それは明らかになる．

```
P¦ (test *)
⟶ call (test *)
      match rule=((test *) (has * tires)
                   (has * doors) (own I *))
⟶ call (has * tires)
      match rule=((has *1^vehicle tires))
⟵ succ (has *1^vehicle tires)
⟶ call (has *1^vehicle doors)
      match rule=((has *2^car doors))
⟵ succ (has *2^car doors)
⟶ call (own I *2^car)
      match rule=((own I mycar))
⟵ succ (own I mycar)
(test mycar)
```

これが解探索の全実行プロセスである．これらの過程を観察すれば，

* ⟶ *1^vehicle ⟶ *2^car ⟶ mycar

という推移が読み取れる．これは解の集合の段階的限定の過程であり，チェックすべき条件に対して最小限必要な処理だけが行われている．n を増加させてインスタンスやクラスに対する知識を追加しても，それらは問題の解に係わりがないので，解探索過程にまったく影響を与えない．つまり，知識の増加が推論の所要時間に悪影響を与えるという矛盾は回避されている．全解をチェックするために追加されるべき時間はほんのわずかなので，全解探索に要する所要時間について考えても，この利点について同様のことが言える．

PAL の方法のキーポイントは

vehicle∩car=car

car∩{mycar}={mycar}

の関係を利用するところにある．しかも PAL の継承階層では，それらの関係を高速に利用できる．それを理解するためには，次の2点に注意すればよい．

（1） 継承階層を構成する基礎であるクラスの関係が排反和であることより，継承階層の任意の二つのクラスの関係は，包含関係か排反関係かのいずれかになる．また，インスタンスとクラスの包含関係の有無も，インスタンスを単元集合（唯一の元からなる集合）と見れば，包含関係か排反関係かのいずれかになる．

（2） 二つの集合に対する包含，排反関係を判定するには，木の中で一方から他方へ行く有向道（path）があるか否かを見ればよい．これは，下位から上位方向に辺をたどる操作だけで判定できる．

5.3 他の方法との比較

prolog の拡張の試みの多くは，ユニフィケーションの拡張に基づいてなされてきた[5),6),8)]．しかし，それらの拡張ユニフィケーションでは，前述した項記述を含めて，（なんらかの意味の）全数探索を回避することはできない．なぜなら，その回避は，継承階層を陽に宣言し，その排反和などの性質をうまく生かすことによってはじめて可能なのに対して，それらの方法では，クラスの知識を一般の1引数述語から明確に分離して扱わないからである．それらの方法では，程度の差こそあれ，DCKR の方法と類似の計算時間を必要とし，同じように問題規模の壁に突き当たる．

PROLOG-II の freeze を用いれば，上記の問題を PAL と同じように高速に解くことができる．それは，条件チェックを変数がアトムに束縛されるまで凍結しておくことによって，求めるものが mycar であると分かった時点で，それが tire や door を持つことをチェックできるからである．しかし上記の例題：「タイヤを持ち，ドアを持ち，私が所有するものは何か？」を少し変えて，「タイヤを持ち，ドアを持つものは何か？」で考えれば，freeze は役にはたたない．それは答えがインスタンスの場合にのみ，たまたま効

果を発揮するものであり，継承階層を一般的に扱う能力はない．

ここに提起された点は非常に重要である．前者の問題に対して「mycar」と答えるのとおなじやり方で，後者の問題には，「自動車」と答え得ることは，（継承階層を扱う）知識表現システムにとって最も基本的な要求事項の一つであろう．これは DCKR や PAL では可能であるが，prolog Ⅱ や項記述を含む多くの（拡張）prolog（の推論機構）では扱うことができない．それらのシステムでは，「自動車」などの「概念」を，システムの答えになり得る「対象」として認めることが，そもそも考慮されていないのである．

クラス束縛変数は，「概念」を「対象」として扱うという意味で，単なる制約付きの変数以上のものである．さらに次の点も重要である．上記の freeze を含む既存の制約付き変数[3),9)]は，制約があるか否かの，単に1段の制約であり，制約の動的な変化は扱っていない．一方 PAL の継承階層では，継承階層に表現された n 段の制約があり，制約は互いに相互作用し，動的に変化して行く．それが上に見られるような解集合の段階的限定を可能としている．またそれらの相互作用はユーザが指定でき，その指定に基づいて極めて小さなコストで実現される．以上の意味で，PAL の継承階層機構は prolog のもう一つの新たな方向を示唆するものである．

6. 実　　現

継承階層機構の実現は，構造コピー方式でも構造共有方式でも簡単である．PAL では，構造共有方式の基本構造をそのまま用いて，通常の prolog の変数束縛に継承階層のための変数のクラス束縛を追加している．それを説明する．内部表現のレベルでは，変数に対するクラス束縛と通常の S 式束縛が同時に起こらないように表現できることに注意せよ．例えば，

$*x=*y$, $*x$^human, $*y$^human

は，変数 $*x$ の環境を env_x，変数 $*y$ の環境を env_y とするとき，

($*x$ env_x)⇒($*y$ env_y)
($*y$ env_y)⇒human

のように内部表現すればよい．したがって，S 式とその環境のペアを変数とその環境のペアに対応させる通常の変数束縛にならって，クラス束縛ではクラス名を変数とその環境のペアに対応させればよい．両者は，ペアか否かで区別される．

ユニフィケーションによってクラス束縛がどのように変化するか，例をあげて説明する．$*x$^human と $*y$^woman はユニファイ可能である．それぞれの環境を env_x, env_y とすれば，束縛は，

from : ($*x$ env_x)⇒human
　　($*y$ env_y)⇒woman

から

to : ($*x$ env_x)⇒($*y$ env_y)
　　($*y$ env_y)⇒woman

へと変化する．同様に，$*x$^woman と $*y$^human の場合もユニファイ可能で，次のように from から to へ変わる．

from : ($*x$ env_x)⇒woman
　　($*y$ env_y)⇒human
to : ($*x$ env_x)⇒woman
　　($*y$ env_y)⇒($*x$ env_x)

また，$*x$^human と $*y$ もユニファイ可能である．

from : ($*x$ env_x)⇒human
to : ($*x$ env_x)⇒human
　　($*y$ env_y)⇒($*x$ env_x)

$*x$^human と adam もユニファイ可能である．

from : ($*x$ env_x)⇒human
to : ($*x$ env_x)⇒(adam env_a)

$*x$^animal と $*y$^vehicle はユニファイ可能ではない．

もちろん上記のユニフィケーションは，woman が human のサブクラスであり，animal と vehicle が排反であるなどの常識的な継承階層の知識を用いた．

PAL では継承階層の知識を，任意のクラスやインスタンスに対して，そのすぐ上やすぐ下のクラスやインスタンスを記述して表現している．そしてその階層構造をたどることによって，クラスやインスタンスの包含，排反関係をチェックし，クラス束縛変数に関するユニフィケーションの結果を求めている．クラスの排反和と完全な木構造を継承階層に要請しているために，それらのチェックはたかだか，葉から根までの距離を2回たどる程度の手間で容易に求めることができる．この手間をさらに改善する方法（継承階層コンパイラ）は別の論文で与える．

7. む　す　び

継承階層とクラス束縛変数を結びつけて prolog を拡張した．その結果，概念の階層構造に関する知識を直感的で自然に記述でき，大規模な知識の処理には致

命的な全数探索を回避して推論速度を大幅に改善できる言語に拡張できた．これは prolog の基本的な構造を十分に生かしたうえでの継承階層の取込である．しかも実現は簡単であり，既存の prolog の部分を使った処理の速度には悪影響を与えない．

いろいろな知識を表現するとき，階層的な構造に依存する部分は非常に大きい．十分実用的な速度で本格的な規模の知識処理を行うためには，階層性を積極的に生かす技術を開発する必要がある．ここで提案された方法は，そのための基礎技術の一つと考えられる．

本論文では，プログラム言語としての立場に立って，高速性を保証できる範囲内に限定して，継承階層を prolog に取り込むための基本的な骨組みを述べた．今後，継承階層を扱うコンパイラなどによるさらなる高速化手法の確立，継承階層を利用する組み込み述語の追加による言語の拡張など，プログラム言語としてのさらなる充実化の課題がある．一方，PAL を知識表現言語の立場から見ることもできる．その方向を追求するためには，現在の PAL の推論速度を極力落とさないで，インスタンスへの制限条件の緩和，多重継承などの表現の導入などの知識表現力を高める方法が検討されなければならない．さらに，自然言語処理などにおける継承階層機構の有効な利用方法の開発なども必要である．

参考文献

1) Berge, C.: *Graphs and Hypergraphs*, p. 528, North-Holland, Amsterdam (1973).
2) Frisch, A. M.: An Investigation into Inference with Restricted Quantification and a Taxonomic Representation, *SIGART Newsl.*, No. 91, pp. 28-31 (1985).
3) Giannesini, F. and Kanoui, H. et al.: *Prolog*, (in French), p. 318, InterEditions, Paris (1985).
4) 小山春生，田中穂積：Definite Clause Knowledge Representation, *Proc. of the Logic Programming Conference '85*, pp. 95-106 (1985).
5) 柴山悦哉：論理型言語におけるユニフィケーションの拡張とその応用，情報処理学会ソフトウェア基礎論研究会資料，10-4, pp. 31-40 (1984).
6) 戸村 哲：TDProlog：項記述可能な Prolog 処理系，*Proc. of the Logic Programming Conference '85*, pp. 237-245 (1985).
7) 中島秀之：知識表現と prolog/kr, p. 179, 産業図書，東京 (1984).
8) Nakashima, H.: Term Description: A Simple Powerful Extension to Prolog Data Structures, *Proc. of IJCAI-IX*, pp. 708-710 (1985).
9) 本吉文男：制約付き変数と関数評価，日本ソフトウェア科学会第1回大会論文集，3E-2, pp. 315-318 (1984).

(昭和61年5月29日受付)
(昭和61年11月5日採録)

赤間 清（正会員）

昭和24年生．昭和48年東京工業大学制御工学科卒業．昭和50年同大大学院修士課程修了．昭和54年同博士課程退学．同年東京工業大学制御工学科助手．昭和56年北海道大学文学部行動科学科講師．現在に至る．人工知能に興味を持つ．とくに昭和50年より一貫して帰納的学習システムを研究し，知能を把える新たな枠組みの構築を目指している．現在，人工知能学会，日本ソフトウェア科学会，日本認知科学会などの会員．