知識工学と人工知能 46－1
（１９８６．５．９）

PAL : Prolog with Extended Unification

for Inheritance Hierarchy

Kiyoshi AKAMA
(Faculty of Letters, Hokkaido University, Sappro-shi, 060, Japan)

Abstract : PAL is an extended prolog with new unification mechanism to deal with inheritance hierarchy(IH). Inferences from knowledge bases including IH may sometimes cause a combinatorial explosion in case of searching from upper classes down to lower classes. PAL adopts a new unification method based on the information of IH and improves the inferential efficiency greatly by avoiding the explosion. It also gives more expressive power. The implementation for the extension is easy and does not affect the efficiency of the inferences in standard prolog.

概念階層を扱う

ＰＲＯＬＯＧ：ＰＡＬ

赤間　清
（北海道大学　文学部　行動科学科）

内容梗概：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．継承階層に関する推論を行うとき，階層の上位から下位の方向に向けてのしらみつぶし探索（＝下方拡大探索）による探索量の爆発が問題となる．継承階層の情報を十分に利用することによって，ＰＡＬは，下方拡大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．この方法は，prologをきわめて自然に拡張するものである．拡張の実現は簡単であり，しかも基礎のprologの推論速度を遅くすることなく継承階層に基づく推論機構を付け加えることができる．

１．　まえがき

自然言語処理などを目指して人間の常識を扱う場合，概念などのなす階層構造による知識の継承が知識表現において大きな役割を果たす．本論文では，継承階層を扱う特別な機構を導入してprologを拡張し，推論の速度や知識の表現力に対してそれがもたらす大きな利点について述べる．

概念の階層構造をprologのなかにうまく取込もうとする試みは，例えば，prolog/krの多重世界［５］や，項記述［４，６］，ユニフィケーションの拡張［３］，ＤＣＫＲ［２］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，階層の上位から下位の方向に向けてのしらみつぶし探索による探索量の爆発である．そのような探索を下方拡大探索と呼ぶことにする．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下方拡大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．下方拡大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いえる．

継承階層の情報を十分に利用することによって，我々は，下方拡大探索を避け推論速度を大幅に向上させる方法を提案する．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をきわ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ものである．拡張の実現は簡単であり，しかも基礎のprologのスピード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．

本論文の継承階層は，言語ＰＡＬとして実現されている．ＰＡＬはprologとlispを結合した言語で，prolog/krと同様のシンタクスを持つ拡張prologである．ＰＡＬで書かれた大きなプログラムとして帰納的学習システムＬＳ／１［１］がある．また現在，ＰＡＬの上で，継承階層を利用した自然言語処理システムの試作が進行中である．

２．　継承階層の記述とその意味

本論文では，概念の包含関係，部分全体の関係，支配非支配の関係など，常識の世界に現われる色々な階層関係の総称として継承階層という言葉を用いる．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２つの概念を区別する．説明の簡単のために，本論文ではインスタンスもクラスも共に有限個のシンボルだけに限定する．

クラスとクラスの基本的な関係は排反和である．例えば，

人間＝男性＋女性

という排反和の例を，ＰＡＬではdefcという組み込み述語を使って

```
(defc human (man woman))
```

と表現する．（defcのcはclassのcである．）

いくつかのインスタンスがあるクラスに含まれているという知識を表現するためには，組み込み述語defiを用いる．（defiのiはinstanceのiである．）例えば，manのクラスがtaro，jiro，saburoの３人を含むという知識は，

```
(defi man (taro jiro saburo))
```

と書ける．

defcとdefiを有限個用いた表現Ｅは，クラスやインスタンスを節(node)とするグラフを与える．その（有向)辺(arc)は，defcやdefiの第１引数の節(node)から第２引数の要素の節(node)に至る辺である．我々は，付随するグラフが根付き木の排反和となるような場合だけを考える．そのとき，そのグラフはあきらかに階層性を持つ．以上が本論文で継承階層と呼ぶ構造の表現方法である．

２つ以上の根付き木の排反和となっているグラフは，新しい節Ｎと，節Ｎからすべての木の根にいたる辺を付け加えることによって，Ｎを根とする１つの根付き木とみなすことができる．この節と辺の追加は，defcを用いた１つの式で表現できる．従って，理論上，１つの根付き木だけを考察の対象にしていると考えても一般性を失わない．以下で一般的な議論をするときは簡単のためにそのようにみなすことにする．

継承階層を表わす表現Ｅにおいて，Ｅ（に付随する木）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，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．追加される新しい変数は，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というクラスの任意のインスタンス）とだけマッチするクラス束縛変数である．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．従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１つである項記述［４，６］を想起させる．項記述では，

*xyz:woman

を任意のwomanとだけマッチする項とみなすことが可能である．宣言的意味に限れば，これら２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）述語（１引数）によって与えられる．項記述とのより重要な違いの１つは，その手続き的意味，すなわち，推論の方法と速度にある．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

項＝コンス＋通常変数＋クラス束縛変数＋定数アトム

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．従って，以下ではクラス束縛変数と４種の項のユニフィケーションを定義する．

１：コンスとのユニフィケーション

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^classx, *y)　　--> 失敗

２：定数アトム a とのユニフィケーション

a が class に含まれるなら成功．*xは a に束縛し直される．

a が class に含まれないならば失敗．

unify(*x^classx, a)

--> *x = a　　if a ∈ classx

--> 失敗　　if a ∉ classx

３：通常変数 *y とのユニフィケーション

成功し，*y も *x と同じクラス束縛を受ける．

unify(*x^classx, *y)　　--> *x = *y, *x^classx, *y^classx

４：クラス束縛変数 *y^classy とのユニフィケーション

２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．

２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^classx, *y^classy)

--> *x = *y, *x^classx, *y^classx　　if classx ⊂ classy

--> *x = *y, *x^classy, *y^classy　　if classx ⊃ classy

--> 失敗　　if classx ∩ classy = φ

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

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２つのクラスの関係は，包含関係か排反関係かのいづれかになる．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．ユニフィケーションを実行する際に問題となりうる処理はそこだけである．それらを判定するには，根付き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，上位方向に辺（arc)をたどる操作だけで判定できる．従って，下方拡大探索は避けられる．

prologの拡張の試み（項記述を含む）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［2，3］．しかし，それらでは下方拡大探索を回避する上記のユニフィケーションを実現できない．なぜなら，それは継承階層の性質を陽に使ってはじめて可能だからである．クラスを1引数の述語という一般のレベルに還元してしか扱わないならば，それが達成できないことは明かである．

4．　PALによる表現

nというパラメータを持つ次の問題を考える．

「自転車と自動車は共に乗り物である．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．乗り物はタイヤを持ち，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．」

PALではこの問題を次のように表現する．nは9とする．まずクラスやインスタンスは

```
(defc vehicle (bicycle car))
(defi bicycle (bic1 bic2 bic3 bic4 bic5 bic6 bic7 bic8 bic9))
(defi car (car1 car2 car3 car4 car5 car6 car7 car8 car9 mycar))
```

のように書く．これを実行すると概念の階層構造がメモリー上に作られる．defcやdefiで作られたものを確認するためにlistcとlistiのコマンドがある．

```
PI(listc)                          ;defc で定義されたクラス名は？
(vehicle)
(listc)
PI(listc vehicle)                  ;vehicleの定義は？
(defc vehicle (bicycle car))
(listc vehicle)
PI(listi)                          ;defi で定義されたクラス名は？
(car bicycle)
(listi)
PI(listi car bicycle)              ;carとbicycleの定義は？
(defi car (car1 car2 car3 car4 car5 car6 car7 car8 car9 mycar))
(defi bicycle (bic1 bic2 bic3 bic4 bic5 bic6 bic7 bic8 bic9))
(listi car bicycle)
```

vehicleの下位概念を求めるには，述語classを使う．

```
PI(class * vehicle)                ;vehicleの下位概念は？
(class vehicle vehicle)
```

それ自身も下位概念だと定義しているので，最初の解はvehicleがでてくる．すべての下位概念を次々に求めるためにトップレベルをmore onモードにする．

```
PIm                                ;more onモードにせよ
----------> more on
```

システムからのプロンプトmore:に対してyと答えるかぎり次の解が探索される．もし更なる解がなければ応答はfailureである．

```
PI(class * vehicle)                ;vehicleの下位概念は？
(class vehicle vehicle)
more:y
(class bicycle vehicle)
more:y
(class car vehicle)
more:y
failure
```

同様に上位概念を求めることができる．（紙面の制約で more:yは省略している．）

```
PI(class car *)                    ;carの上位概念は？
(class car car)
(class car vehicle)
```

```
failure
PI(class bicycle *)                    ;bicycleの上位概念は？
(class bicycle bicycle)
(class bicycle vehicle)
failure
```

次のような問い合わせはサポートしていない．

```
PI(class * **)                         ;上位下位の関係にあるものは？
can not answer------> (class * **)
```

片方がインスタンスである場合には述語instanceを用いる．ここではvehicleのすべてのインスタンスを求める例を示す．（紙面の制約で more:yは省略している．）

```
PI(instance * vehicle)                 ;vehicleの例は？
(instance bic1 vehicle)
(instance bic2 vehicle)
   …省略…
(instance bic9 vehicle)
(instance car1 vehicle)
(instance car2 vehicle)
   …省略…
(instance car9 vehicle)
(instance mycar vehicle)
failure
```

階層構造以外の述語はdefineかdefpで定義する．defineはprolog/krと同じシンタックスである．

```
(define test ((*) (has * tires) (has * doors) (own I *)))
(define own ((I mycar)))
```

defpは階層の情報を用いて述語を表現できる．例えばhasは

```
(defp has
  ((*1^vehicle tires))
  ((*2^car doors)))
```

とかける．これらは人間の直観にあっており，読みやすく，書きやすい．これらを実行した結果は組み込み述語listを用いて見ることができる．引数をなしにすれば述語（それに関数）のリストが得られる．

```
PI(list)                               ;述語または関数の定義が見たい
pred1(1) pred2(2), or func(3)  1       ;何番のタイプが見たいか？   1番です
(test own)
(list)
PI(list)                               ;述語または関数の定義が見たい
pred1(1) pred2(2), or func(3)  2       ;何番のタイプが見たいか？   2番です
(has)
(list)
```

defineとdefpで作った述語はpred1とpred2のタイプ名で区別される．それらを述語listで見ると，

```
PI(list test own has)                  ;test own has の定義は？
(as (test *) (has * tires) (has * doors) (own I *))
(as (own I mycar))
(as (has *1^vehicle tires))
(as (has *2^car doors))
(list own has)
```

述語をdefineやdefpの形式で書かせるか，又は，asの形式で書かせるかは指定できるが，ここでは後者の例を示した．

5．　既存の方法との比較

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．例えば前節の問題を考えてみる．これは求解過程から見て，prologの典型的な問題の１つである．この問題をいろいろな方法で，また，ｎをいろいろに変えて，解を得る所要時間を比較する．（以下の表現はｎ＝９の場合を示す．）

［１］ＤＣＫＲの場合の表現

ＤＣＫＲを用いた表現の例を挙げる．ＰＡＬではファンクターを扱わないので，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のリストで表現する．

```
(as (sem bicycle *) (sem vehicle *))
(as (sem (bic1) *) (sem bicycle *))
(as (sem (bic2) *) (sem bicycle *))
    …省略…
(as (sem (bic9) *) (sem bicycle *))
(as (sem car *) (sem vehicle *))
(as (sem (car1) *) (sem car *))
(as (sem (car2) *) (sem car *))
    …省略…
(as (sem (car9) *) (sem car *))
(as (sem (mycar) *) (sem car *))
(as (sem vehicle (has tires)))
(as (sem car (has doors)))
(as (sem (mycar) (own I)))
(as (test *) (sem * (has tires)) (sem * (has doors)) (sem * (own I)))
```

［２］述語ＡＫＯを用いた表現

述語superで，概念の階層で隣接する上下関係を記述する．

```
(as (super bicycle vehicle))
(as (super bic1 bicycle))
(as (super bic2 bicycle))
    …省略…
(as (super bic9 bicycle))
(as (super car vehicle))
(as (super car1 car))
(as (super car2 car))
    …省略…
(as (super car9 car))
(as (super mycar car))
```

述語superを基礎として，概念の包含関係akoを定義する．述語akoでは，与えられる引数が変数か定数かによって，上方向または下方向の探索が引き起こされる．

```
(as (ako * *))
(as (ako *x *z) (atom *x) (cut) (super *x *y) (ako *y *z))
(as (ako *x *z) (var *x) (atom *z) (cut) (super *y *z) (ako *x *y))
```

akoを使えば，残りの記述は容易である．

```
(as (own I mycar))
(as (has * tires) (ako * vehicle))
(as (has * doors) (ako * car))
(as (test *) (has * tires) (has * doors) (own I *))
```

［３］継承階層を用いた表現

前節に述べたように，継承階層を使えば，問題は次のように表現できる．

```
(defc vehicle (bicycle car))
(defi bicycle (bic1 bic2 bic3 bic4 bic5 bic6 bic7 bic8 bic9))
(defi car (car1 car2 car3 car4 car5 car6 car7 car8 car9 mycar))
(defp own ((I mycar)))
(defp has ((*1^vehicle tires)) ((*2^car doors)))
(defp test ((*) (has * tires) (has * doors) (own I *)))
```

問題のパラメーターnと表現方法を変えて，所要時間を比較すると表1が得られる．

nの値	3	6	9	18	27	
方法1(DCKR)	2.4	6.1	11.	63.	210.	秒
方法2(AKO)	2.9	7.3	13.	57.	150.	秒
方法3(PAL)	0.04	0.04	0.04	0.04	0.04	秒

表1　各表現方法ごとの求解の所要時間

ＰＡＬの継承階層による方法は，既存の方法と比べて圧倒的に速い．それは単にインプリメントの方法のレベルの問題ではなく，基本的な推論の方法の改善によって達成されたものである．第1の方法や第2の方法では，解の候補となるクラスやインスタンスは次のように変化する．

```
vehicle,
bicycle,bic1,bic2,bic3,bic4,bic5,bic6,bic7,bic8,bic9,
car,car1,car2,car3,car4,car5,car6,car7,car8,car9,
mycar
```

これらの最後にようやくmycarで成功するのである．これは明らかに無駄が多過ぎる．

ＰＡＬはこの無駄を，クラス束縛を用いて解消する．推論の方法は，上記の述語testを逐次実行してみれば明らかになる．デバッグのモードへの移行はコマンドｄを用いる．

```
Pld                                    ;逐次実行モードにせよ
----------> debug on
Pl(test *)
---> call  (test *)
---> call  (has * tires)
<--- succ  (has *1^vehicle tires)
---> call  (has *1^vehicle doors)
<--- succ  (has *2^car doors)
---> call  (own I *2^car)
<--- succ  (own I mycar)
(test mycar)
```

これが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，方法1や方法2の探索の推移とは質が違うことに注意せよ．それらの探索のように，

```
vehicle, car, mycar, …
```

としらみつぶしに探索しているのではなく，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言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．

6. 実現

ＰＡＬはFranz lisp上で，構造共有方式(structure sharing)を用いて実現されている．この場合，通常のprologの束縛はＳ式と環境のペアを変数に対応させることによってなされる．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するのは簡単である．なぜなら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からである．従って，Ｓ式と環境のペアを変数に対応させる通常の変数束縛に倣って，クラス束縛ではクラス名を変数に対応させ，それらを区別する機構（ＰＡＬでは単にコンスかアトムかで分けている）を設けておけばよい．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，

(*x env_x) => (human) ーー> (*x env_x) => (*y env_y)
(*y env_y) => (woman) ーー> (*y env_y) => (woman)

と変化する．また，*x^humanと*yもユニファイ可能で，

(*x env_x) => (human) ーー> (*x env_x) => (human)
*yの束縛なし ーー> (*y env_y) => (*x env_x)

*x^humanとadamはユニファイ可能である．束縛の変化は，

(*x env_x) => (human) ーー> (*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下位から上位方向に木構造を最大２回たどる程度の手間で容易に求めることができる．

7. むすび

継承階層とクラス束縛変数により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下方拡大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．

本論文では，継承階層とクラス束縛変数をprologに取込むことに関して，最も基本的な骨組みだけを述べた．継承階層の表現力の増加や多重継承の問題，さらなる高速化に向けてのシステムの拡張などについては，稿を改めて報告する予定である．

文献

[1] 赤間 清：「知識の構造化を重視した学習のモデル」
情報処理学会 知識工学と人工知能研究会資料45-5 (1986)

[2] 小山春生，田中穂積："Definite Clause Knowledge Representation"，
Proc. of the Logic Programming Conference '85 (1985)

[3] 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 (1984)

[4] 戸村 哲：「TDProlog:項記述可能なProlog処理系」
Proc. of the Logic Programming Conference '85 (1985)

[5] 中島秀之：「知識表現とprolog/kr」産業図書 (1984)

[6] Hideyuki Nakashima："Term Description": A simple powerful Extension
to Prolog Data Structures, Proc. of IJCAI-IX (1985)