知識工学と人工知能　52－3
（１９８７．５．７）

代表元を使った知識の処理方式について

坂間　千秋　　伊藤　英則
（財）新世代コンピュータ技術開発機構

ホーン論理で表された知識ベースにおいて，リテラルの集合を意味上の同値類に分類し，その代表元によって知識を処理する方式について述べる．本方式によれば，同じ知識の異なった表現による処理，分散した知識の統合処理，あるいは状況の変化に応じた知識のダイナミックな処理が可能となる．

HANDLING KNOWLEDGE BY ITS REPRESENTATIVE

Chiaki SAKAMA　and　Hidenori ITOH
ICOT Research Center
1-4-28, Mita, Minato-ku, Tokyo, 108 Japan

In such a knowledge base represented by Horn logic, we present a method of classifying a set of literals into equivalent classes by their meaning and handling them by their representatives.
This method enables us to deal with some knowledge expressed in a different manner uniformly, to merge some distributed knowledge, and to utilize knowledge dynamically according to the situation.

1．はじめに

知識表現の手段として述語論理が有効である理由はその構文，意味が明確に定義されていること，及び推論結果の正当性が保証されていることである．特に Robinson の導出原理に基づく汎用プログラミング言語としての Prolog が開発されて以来，DCG (Definite Clause Grammar) などによる自然言語処理における応用，または関係データベースとの親和性等が確認されその重要性は再確認されてきた．

しかし，一方では知識表現言語としての立場からPrologのような一階述語論理の部分クラスであるホーン節に限った記述言語の表現能力の不足，あるいは導出，ユニフィケーションの機能に対する拡張要求などが問題点として挙げられ，これまでにもいくつかの提案がされてきた．（例えば〔Nakashima 84〕,〔Ishizuka 85〕,〔Ait-kaci 86〕,〔Porto 86〕,〔Mukai 86〕,〔Yamaguchi 86〕,〔Yokomori 86〕etc.）

ところが多くの論理型言語がそうであるように，異なる名前のリテラルは異なる概念を表すという仮定（unique name assumption）の下では，ある概念の表現方法はプログラム中で一定でなければならず，リテラルの同値関係を表現するためにはその関係をプログラム中に陽に記述しておく必要がある．

本稿では，このようなリテラル間の同値関係を陰に扱うために，表現上は異なるが意味上は同値関係にあるリテラルを集合として扱い，その代表元によって処理を行う方式について述べる．以下，第２章ではホーン節集合におけるリテラルの分類とその代表元について定義し，第３章では代表元による知識のコンパイルの方法とその意義について議論する．また，第４章では知識ベースにおける問い合わせ処理における適用について述べる．

2．リテラルの分類

2.1 リテラルの同値関係

あるホーン節集合がプログラムとして与えられたとき，そこに表われるリテラルの集合を意味上の同値関係によって分類することを考える．そこで，先ずリテラルの同値関係を定義する．

定義（同値関係）
リテラルの集合 Λ が与えられたとき，

p，q∈ Λ について （p ⊃ q ）かつ（q ⊃ p ）

ならば，p と q は同値（equivalent）であるといい，pと同値関係にある元の集合を p の同値類（equivalent class）という．

このとき Λは同値類の直和に分割され，この直和分割をΛの同値に関する類別（classification）という．また Λの類別において各同値類から１つずつ取り出した Λの元をそれらの同値類の代表元（representative）という． □

例
リテラルの集合 Λ

Λ = { ancestor(X,Y), parent(X,Y),
descendant(Y,X), offspring(Y,X) }

に対して，同値類 Φ_1，Φ_2

Φ_1 = { ancestor(X,Y), descendant(Y,X) }
Φ_2 = { parent(X,Y), offspring(Y,X) }

及び，それらの代表元

ϕ_1 = ancestor(X,Y)
ϕ_2 = parent(X,Y)

が定義される． □

2.2 代表元の選択

いま，同値類 Φ とその代表元 φ が

Φ = { ancestor(X,Y), descendant(Y,X) }
φ = ancestor(X,Y)

で与えられていて，ある代入 θ＝{ taro/X, hanako/Y } に対して Φ のインスタンス

Φθ = { ancestor(taro,hanako),
descendant(hanako,taro) }

がまた同値類として定義されるとき，Φθ の代表元を φ のインスタンス

φθ = ancestor(taro,hanako)

となるように選ぶことが出来る．

ところが，一般にはある同値類 Φ の代表元 φ を Φ のインスタンス Φθ に対して定義される同値類の代表元が φθ となるように一意に選ぶためには，同値類における代表元の選択に対する基準が必要になる．

そこで，以下のような代表元の選択基準を定める．

＜代表元の選択基準＞
同値類の集合 F＝{ Φ_i| i = 1,2,..,n } が与えられたとき，それぞれの同値類の代表元 ϕ_i(i = 1,2,..,n) を以下のように定める．

① 同値類 $\Phi_i \in F$ が他のどの同値類 $\Phi_j \in F$ ($i \neq j$) の元ともユニファイ可能な元を持たないとき，Φ_iの任意の元を ϕ_iとして選べる．
② 同値類 $\Phi_i \in F$ が他のある同値類 $\Phi_j \in F$ ($i \neq j$) の元とユニファイ可能な元を持てば，その元の中から ϕ_i を選ぶ．

例
同値類 Φ_1，Φ_2が次のように与えられたとする．

Φ_1 = { boss(X,Y), master(X,Y) }
Φ_2 = { boss(X,X), master(X,X), chief(X) }

（ここで，X が Y のボスであるとは X が Y のマスターであることであり，X が X のボスであるとは自分自身がチーフであることと等しいと読む．）

いま Φ_1，Φ_2 に対して代入 θ_1 = { taro/X, taro/Y }，θ_2 = { taro/X } が存在して，それぞれのクラスのインスタンス

$\Phi_1\theta_1$ = { boss(taro,taro), master(taro,taro) }
$\Phi_2\theta_2$ = { boss(taro,taro),
master(taro,taro), chief(taro) }

が存在するとき，同値類 Φ_{12}

Φ_{12} = { boss(taro,taro),
master(taro,taro), chief(taro) }

が定義されるが，Φ_{12} の代表元を一意に決めるためには上の選択基準によって Φ_1 の代表元として boss(X,Y) または master(X,Y) を選ぶことが出来るが，Φ_2 の代表元としては boss(X,X) または master(X,X) しか選べない．（このとき Φ_{12} の代表元は boss(taro,taro) または master(taro,taro) となる．）（図１）． □

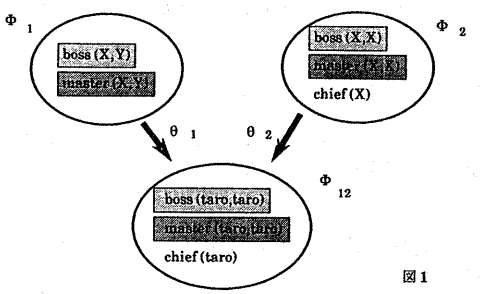

図１

こうして選ばれた候補の中から，代表元をそれぞれ一つずつ決められる．ところが，次のような例が考えられる．

例
同値類 Φ_1，Φ_2，Φ_3

Φ_1 = { father(taro,Y), parent(taro,Y) }
Φ_2 = { parent(X,goro), has_a_son(X,goro) }
Φ_3 = { father(X,hanako), has_a_daughter(X,hanako) }

において Φ_1，Φ_2 に対して代入 θ_1 ={ taro/X, goro/Y } が存在して，

$\Phi_1\theta_1$ = { father(taro,goro), parent(taro,goro) }
$\Phi_2\theta_1$ = { parent(taro,goro), has_a_son(taro,goro) }

で，一方 Φ_1，Φ_3 に対して代入 θ_2 ={ taro/X, hanako/Y } が存在して，

$\Phi_1\theta_2$ = { father(taro,hanako), parent(taro,hanako) }
$\Phi_3\theta_2$ = { father(taro,hanako),
has_a_daughter(taro,hanako) }

のとき，同値類 Φ_{12}，Φ_{13}

Φ_{12} = { father(taro,goro), parent(taro,goro),
has_a_son(taro,goro) }
Φ_{13} = { father(taro,hanako), parent(taro,hanako),
has_a_daughter(taro,hanako) }

が定義される．このとき Φ_1 の代表元としては，Φ_2 における parent(X,goro) とユニファイ可能な parent(taro,Y) と Φ_3 における father(X,hanako) とユニファイ可能な father(taro,Y)，また，Φ_1 のインスタンスを含む同値類 Φ_{12}，Φ_{13} の代表元はとしてそれぞれ parent(taro,goro) と father(taro,goro)，及び parent(taro,hanako) と father(taro,hanako) が定義される（図２）． □

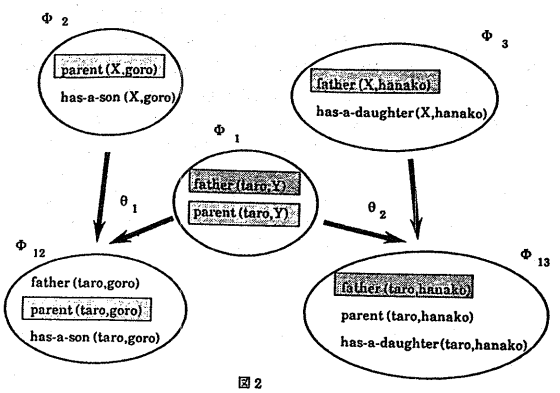

図２

このように，上の選択基準によって代表元が一意に決まらない場合は最小限の複数の代表元を選ぶことにする．

3．代表元によるプログラム変換

3.1 代表元プログラム

あるホーン節集合からなるプログラムが与えられた場合，プログラム中の全てのリテラルに対してそれぞれ同値関係にある代表元リテラルが定義されるとき，もとのプログラムは代表元のみからなるプログラム（代表元プログラム）に変換することが出来る．

例
プログラム S が以下のように与えられたとする．

S = { ancestor(X,Y) ← parent(X,Y).
ancestor(X,Y) ← parent(X,Z),ancestor(Z,Y).
parent(taro,hanako).
child(jiro,hanako). }

ここで，同値類と代表元がそれぞれ

Φ_1 ={ ancestor(X,Y) }, ϕ_1 = ancestor(X,Y)
Φ_2 ={ parent(X,Y), child(Y,X) }, ϕ_2 =parent(X,Y)

で定義されているとき，S は代表元プログラム S'

```
S' = { ancestor(X,Y) ← parent(X,Y).
       ancestor(X,Y) ← parent(X,Z),ancestor(Z,Y).
       parent(taro,hanako).
       parent(hanako,jiro).  }
```

に変換される．□

上の例では S を S' に変換することによって，例えば S からは ancestor(taro,jiro)が演繹されないが，S' からは演繹される．即ち，演繹結果はもとのプログラムにおける親子関係の表現方法に依存しない．

ここで，Sに例えば節 parent(X,Y) ←child(Y,X). を加えればS'と等しくなるが，節 child(X,Y) ←parent(Y,X). も S に加えた場合，Prolog のような深さ優先探索においては，これらが相互再帰によるループを生じないような制御が必要となる．また一般に，このようにリテラルの同値関係をプログラム中に宣言的に記述しておく場合，例えば n 個のリテラルの同値関係を表現するには少なくとも n 個の節が必要となり，これらは導出に対する負荷となる．

さて，このようにプログラムを代表元プログラムに変換して処理することによって，例えば次のような複数のプログラムの統合処理が可能になる．

例

プログラム S_1，S_2 がそれぞれ以下のように与えられたとする．

```
S1 = { ancestor(X,Y) ← parent(X,Y).
       ancestor(X,Y) ← parent(X,Z),ancestor(Z,Y).
       parent(taro,hanako).
       child(jiro,hanako).  }

S2 = { descendant(X,Y) ← offspring(X,Y).
   descendant(X,Y)←offspring(X,Z),descendant(Z,Y).
   offspring(jiro,hanako).
   offspring(ichiro,hanako). }
```

ここで，同値類と代表元がそれぞれ

Φ_1 = { ancestor(X,Y), descendant(Y,X) }
Φ_2 = { parent(X,Y), child(Y,X), offspring(Y,X) }
ϕ_1 = ancestor(X,Y)
ϕ_2 = parent(X,Y)

で定義されているとき，S_1，S_2 はそれぞれ以下の代表元プログラム S_1'，S_2' に変換される．

```
S1' = { ancestor(X,Y) ← parent(X,Y).
        ancestor(X,Y) ← parent(X,Z),ancestor(Z,Y).
        parent(taro,hanako).
        parent(hanako,jiro).  }

S2' = { ancestor(Y,X) ← parent(Y,X).
        ancestor(Y,X) ← parent(Z,X),ancestor(Y,Z).
        parent(hanako,jiro).
        parent(hanako,ichiro). }
```

この結果，変換前の２つのプログラムの和 $S_1 \cup S_2$ からは例えば ancestor(taro,ichiro) は演繹されないが，変換後の２つのプログラムの和 $S_1' \cup S_2'$ からは演繹される．□

上の例は，異なるプログラマによる S_1，S_2 のような異なる表現で書かれたプログラムを，それぞれ代表元プログラムに変換することによって統合して処理出来ることを示している．この結果，例えば S_1 のプログラマは自分の知らない S_2 の知識を利用することが出来る．

3.2 環境に応じた推論

プログラム中のリテラルに関する同値関係は一意的なものではなく，状況が異なれば同値関係も異なる場合が考えられる．そこで，このような複数の同値関係を使ってプログラムを処理する方法について述べる．

先ず，同値関係の強弱を定義する．

定義（同値関係の強弱）

リテラルの集合 Λ における２つの同値関係 E_1，E_2 に対して

$\forall x, y \in \Lambda$ について　$x E_1 y$ ならば $x E_2 y$

であるとき，E_1 は E_2 より強い（strong），あるいは E_2 は E_1 より弱い（weak）という．□

例

２つの同値関係 E_1，E_2 に対して，それぞれ同値類 Φ_{E1}，Φ_{E2} が

Φ_{E1} = { parent(X,Y), child(Y,X) }
Φ_{E2} = { parent(X,Y), child(Y,X), mother(X,Y) }

で定義されるとき，E_1 は E_2 よりも強い（図３）．□

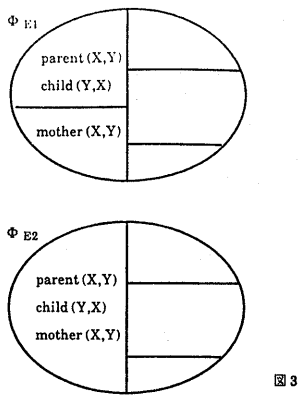

図3

このような強弱による順序関係について，リテラルの集合の同値関係全体の集合は完備束を成すが，以下ではこの

ような複数の同値関係が定義されている場合のプログラムの代表元プログラムへの変換について述べる.

例

プログラム S が以下のように与えられたとする.

```
S = { marriage(X,Y) ← engaged(X,Y).
      engaged(ichiro,sachiko).
      lovers(jiro,momoko). }
```

このとき, ある同値関係 E_1 の下では同値類と代表元がそれぞれ

Φ_1 = { marriage(X,Y) } , ϕ_1 = marriage(X,Y)
Φ_2 = { engaged(X,Y) } , ϕ_2 = engaged(X,Y)
Φ_3 = { lovers(X,Y) } , ϕ_3 = lovers(X,Y)

と定義されていて, また別の同値関係 E_2 の下では同値類と代表元がそれぞれ

Φ_1' = { marriage(X,Y) } , ϕ_1' = marriage(X,Y)
Φ_2' = { engaged(X,Y), lovers(X,Y) } ,
ϕ_2' = engaged(X,Y)

と定義されていたとする. このとき, E_1 の下では S は代表元プログラム S'

```
S' = { marriage(X,Y) ← engaged(X,Y).
       engaged(ichiro,sachiko).
       lovers(jiro,momoko). }
```

に等しいが, E_2 の下では S は代表元プログラム S"

```
S"= { marriage(X,Y) ← engaged(X,Y).
      engaged(ichiro,sachiko).
      engaged(jiro,momoko). }
```

に変換される.
この結果, S' からは marriage(jiro,momoko) が演繹されないが, S" からは演繹される. □

上の例では, E_1 の方が E_2 よりも強い (細かい) 同値関係を表しているが, 例えば E_1 と E_2 がそれぞれ時間的に前後関係にある同値関係と考えると, E_1 から E_2 へ同値関係を変化させることによって, 同じプログラムに対してダイナミックに異なる意味を与えることが出来る.
もう一つ例を挙げよう.

例

プログラム S が以下のように与えられたとする.

```
S = { stranger(X) ← foreigner(X).
      american(bob).
      japanese(taro).
      japanese(hanako). }
```

このとき, ある同値関係 E_1 の下では同値類と代表元がそれぞれ

Φ_1 = { stranger(X) } , ϕ_1 = stranger(X)
Φ_2 = { foreigner(X), american(X) } ,
ϕ_2 = foreigner(X)
Φ_3 = { japanese(X) } , ϕ_3 = japanese(X)

と定義されていて, また別の同値関係 E_2 の下では同値類と代表元がそれぞれ

Φ_1' = { stranger(X) } , ϕ_1' = stranger(X)
Φ_2' = { american(X) } , ϕ_2' = american(X)
Φ_3' = { foreigner(X), japanese(X) } ,
ϕ_3' = foreigner(X)

と定義されていたとする. このとき, E_1 の下では S は代表元プログラム S'

```
S' = { stranger(X) ← foreigner(X).
       foreigner(bob).
       japanese(taro).
       japanese(hanako). }
```

に変換されるが, E_2 の下では S は代表元プログラム S"

```
S"= { stranger(X) ← foreigner(X).
      american(bob).
      foreigner(taro).
      foreigner(hanako). }
```

に変換される.
この結果, S' からは stranger(bob) が演繹され, S" からは stranger(taro) あるいは stranger(hanako) が演繹される. □

上の例では, E_1, E_2 間には強弱関係はないが, それぞれ日本人, アメリカ人による同値関係と考えると, それぞれの同値関係の下で同じプログラムに対して異なる意味を与えることが出来る.

このように, 同じ表現でも使われる状況によって異なる意味を持ち得るという解釈は状況意味論における立場であり ([白井 86]), これらはそこでの環境に応じた推論 (situated inference) の一例である.
また, このような複数の同値関係があった場合, どの同値関係を選択するかはプログラムを処理する状況に応じて決定されなければならないが, 例えば前節の例のように複数のプログラムを統合処理する場合には, ある一定の同値関係を固定して同じ状況の下でプログラムを処理しなくてはいけない.

3.3 代表元プログラムの意味

次に代表元プログラムの意味について考える.
いま, ホーン節集合からなるプログラムを S, S 中のリテラルの代表元への変換を T , 変換後の代表元プログラムを T (S) で表したとき以下の定理が成り立つ.

定理

S からの導出で定理 p が証明可能ならば，T（S）からの導出で定理 T（p）が証明可能である．

（証明）

S からの pの導出の段数に関する数学的帰納法による．

i）Sから pが 0段の導出で証明可能ならば，p∈S である．このとき，T（p）∈ T（S）であるから T（S）から T（p）が 0 段の導出で証明可能である．

ii）Sからpがn段の導出で証明可能であるとき，T（S）から T（p）が n段の導出で証明可能であると仮定する．

いま，Sから pが n＋1 段の導出で証明可能な場合，n＋1 段目の導出は以下のようになっている．

$$\frac{\begin{array}{cc} : & : \\ \Gamma_1 \vee \Delta_1 & \Gamma_2 \vee \neg\Delta_2 \end{array}}{\Gamma_1\theta \vee \Gamma_2\theta} \quad \begin{array}{l}(n\ 段)\\ (n+1\ 段)\end{array}$$

（ここで，Γ_1，Γ_2はリテラルの論理和，Δ_1，Δ_2はリテラル，$\Gamma_1\theta \vee \Gamma_2\theta = p$，また θは Δ_1とΔ_2の mgu で $\Delta_1\theta = \Delta_2\theta$ である．）

このとき，仮定より次のような T（S）からの n 段の導出が存在する．

$$\frac{\begin{array}{cc} : & : \\ T(\Gamma_1 \vee \Delta_1) & T(\Gamma_2 \vee \neg\Delta_2) \end{array}}{} \quad (n\ 段)$$

ここで，$T(\Gamma_1 \vee \Delta_1) = T(\Gamma_1) \vee T(\Delta_1)$，$T(\Gamma_2 \vee \neg\Delta_2) = T(\Gamma_2) \vee \neg T(\Delta_2)$ で，また $\Delta_1\theta = \Delta_2\theta$ より，$T(\Delta_1)\theta = T(\Delta_2)\theta$ であるから，

$$\frac{T(\Gamma_1) \vee T(\Delta_1) \qquad T(\Gamma_2) \vee \neg T(\Delta_2)}{T(\Gamma_1)\theta \vee T(\Gamma_2)\theta}$$

によって，$T(\Gamma_1)\theta \vee T(\Gamma_2)\theta$ が導出される．

ここで，

$T(\Gamma_1)\theta \vee T(\Gamma_2)\theta$

$= T(\Gamma_1\theta \vee \Gamma_2\theta) = T(p)$

より，T（S）から T（p）が n＋1 段の導出で証明される． □

上の定理から，Sの解集合を M（S），T（S）の解集合を M（T（S））で表すと以下の包含関係が成り立つ．

$$T(M(S)) \subseteq M(T(S))$$

ここで，T^{-1}を代表元からそれを含む同値類の元全体への逆変換とすると上式より，

$$M(S) \subseteq T^{-1}(M(T(S))) \qquad (*)$$

（*）式は，一旦代表元に変換してから求めたプログラムの解集合を逆変換したものは元のプログラムの解集合を含むことを示している（図4）．

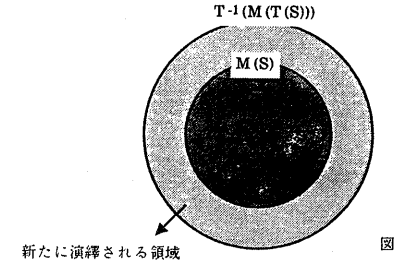

図4

即ち，代表元プログラムへの変換によってもとのプログラムからは演繹されなかった解が演繹され得る．

3.4 代表元プログラムにおける冗長性除去

一般に，もとのプログラム中に同じ概念が違う表現で書き替えられているような場合，代表元プログラムへの変換後は代表元による等しい表現となるため冗長性が発生する．

例

プログラム S

```
S = { ancestor(X,Y) ← parent(X,Y).
  ancestor(X,Y) ← parent(X,Z),ancestor(Z,Y).
  descendant(X,Y) ← offspring(X,Y).
  descendant(X,Y)←offspring(X,Z),descendant(Z,Y).
  parent(X,Y) ← child(Y,X).
  child(hanako,taro).
  child(jiro,hanako).
  offspring(jiro,hanako). }
```

が，同義類と代表元

Φ_1 = { ancestor(X,Y), descendant(Y,X) }
Φ_2 = { parent(X,Y), child(Y,X), offspring(Y,X) }
ϕ_1 = ancestor(X,Y)
ϕ_2 = parent(X,Y)

によって代表元プログラムS'

```
S' = { ancestor(X,Y) ← parent(X,Y).
       ancestor(X,Y) ← parent(X,Z),ancestor(Z,Y).
       ancestor(Y,X) ← parent(Y,X).
       ancestor(Y,X) ← parent(Z,X),ancestor(Y,Z).
       parent(X,Y) ← parent(X,Y).
       parent(taro,hanako).
       parent(hanako,jiro).
       parent(hanako,jiro). }
```

に変換されるとき，S' は冗長な4つの節

```
       ancestor(X,Y) ← parent(X,Y).
```

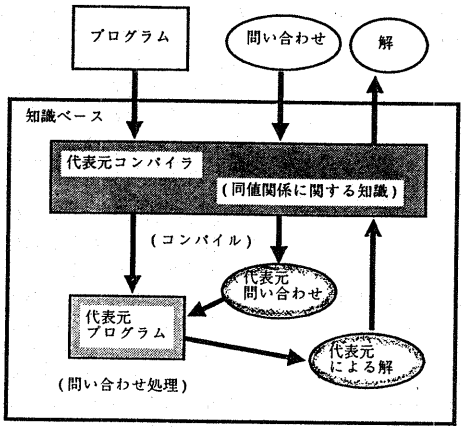

図5

```
        ancestor(X,Y) ← parent(X,Z),ancestor(Z,Y).
(または ancestor(Y,X) ← parent(Y,X).
        ancestor(Y,X) ← parent(Z,X),ancestor(Y,Z).)
        parent(X,Y) ← parent(X,Y).
        parent(hanako,jiro).
```

を含む. □

このような代表元プログラムにおける冗長性は，実はもとのプログラムが陰に含んでいた冗長性が，変換後は陽に表われたにすぎないのであるが，代表元プログラムの処理の効率化の上ではこのような冗長性を除去することは重要である．そこで，このような冗長な節の除去方法として，先ず以下の消去戦略([Chang 73]) を考える．

① ある節 C においてボディ部のあるリテラルがヘッド部のリテラルと一致する場合（トートロジー）C を除去する．
② ある節 C が他の節 C' を包摂 (subsume) する場合 ($C\sigma \subseteq C'$, σ：代入) C' を除去する．

これによると，上の例では先ず ① によって節

```
parent(X,Y) ← parent(X,Y).
```

が S' から除去される．次に，② によって２つの節

```
ancestor(Y,X) ← parent(Y,X).
parent(hanako,jiro).
```

がそれぞれ節

```
ancestor(X,Y) ← parent(X,Y).
parent(hanako,jiro).
```

によって包摂されるので，S' から除去される．
次に，残った２つの節

```
ancestor(X,Y) ← parent(X,Z),ancestor(Z,Y).
ancestor(Y,X) ← parent(Z,X),ancestor(Y,Z).
```

であるが，これらの節が等価であることは上の消去戦略からは判定できない．

このような遷移的 (transitive) な場合を含めた再帰節 (recursive clause) の等価関係は，それぞれの節のコンパイル結果（ボディ部の展開結果）を比較しなければ判定出来ない場合が多い．この場合はどちらの節もコンパイルの結果が

$$ancestor(X,Y) \leftarrow \bigcup_1^n parent^i .$$

$$\left(\bigcup_1^n parent^i = \{\ parent(X,Y),\ parent(X,Z),parent(Z,Y)\ ,\ \ldots\ ,\ parent(X,W),parent(W,\_),..,parent(\_,V),parent(V,Y)\}\ \right)$$

となるため２つの節が等しいことが分かる([Han 86])．

一般に，ホーン節集合で与えられたプログラムを等価変換して，冗長な節，リテラルを完全に除去することは簡単な場合を除いてかなり負荷の重い処理であり，多くの場合決定不可能であることが知られている([Sagiv 86])．従って，これをどの程度まで実現するかはプログラムの効率とのバランスによって決定されねばならない．

4．代表元コンパイルと問い合わせ処理

次に，代表元プログラムを使った知識ベースにおける問い合わせ処理方法について述べる．

いま，リテラルの同値関係，代表元に関する知識が定義されているとき，これらを予めプログラムに対するメタ知識として知識ベースに貯えておいて，これに基づいて知識ベースに対する入力プログラムを代表元プログラムに変換（コンパイル）することを考える（図５）．

このとき，もとのプログラムに対する問い合わせは一旦代表元による問い合わせに変換して代表元プログラムにおいて処理を行い，その処理結果に対しては逆変換を施して

出力する．ここで，代表元プログラムにおける計算メカニズムとしては通常のＳＬＤ演繹を適用する．

例

プログラム S

```
S = { descendant(X,Y) ← offspring(X,Y).
  descendant(X,Y) ← offspring(X,Z),descendant(Z,Y).
  offspring(jiro,hanako).
  parent(taro,hanako).  }
```

に対する問い合わせ ?- descendant(X,Y). の処理は，S の代表元プログラム S'

```
S' = { ancestor(Y,X) ← parent(Y,X).
       ancestor(Y,X) ← parent(Z,X),ancestor(Y,Z).
       parent(hanako,jiro).
       parent(taro,hanako).  }
```

(ここで，Φ_1 = { ancestor(X,Y), descendant(Y,X) }
ϕ_1 = ancestor(X,Y)
Φ_2 = { parent(X,Y), offspring(Y,X) }
ϕ_2 = parent(X,Y))

に対する代表元問い合わせ ?- ancestor(Y,X). に変換されて処理される．

この結果，代表元問い合わせに対する解集合

```
{ ancestor(hanako,jiro), ancestor(taro,hanako),
  ancestor(taro,jiro) }
```

は，もとの問い合わせに対する解集合

```
{ descendant(jiro,hanako), descendant(hanako,taro),
  descendant(jiro,taro) }
```

に変換されて出力される．□

このように，プログラム，及び問い合わせ中のリテラルに関する同値関係と代表元が知識ベース中に定義されている場合，上のように代表元による問い合わせに変換して処理を行うことが出来る．

5．おわりに

ホーン論理で表現された知識ベースにおいて，リテラルの同値関係に基づいた代表元を使って知識の処理を行う方法について述べた．この結果，

i) 知識の処理は代表元に対して行われるので，もとのプログラム中の知識の表現方法には依存しない．
ii) ローカルな表現を共通の代表元に変換することによって，分散した知識の統合処理が可能となる．
iii) 状況に応じて同値関係を変化させることによって，一つのプログラムに対してダイナミックな解釈を与えることが出来る．

このようなモデルにおいては，代表元をもとのプログラムに対するアセンブラと見做し，知識の処理はこのアセンブラに対して行われると考えると，知識独立性の高い柔軟なシステムが構築出来ると考えられる．

また，本稿ではホーン論理の枠組みにおいて議論してきたが，ここで述べた手法は一般のデータベース，知識ベースシステムにおいても，それぞれのデータ構造に応じた適用が可能であると考えている．

最後に問題点について触れる．

本稿では，プログラム中に表われるリテラルに関する同値関係はメタ知識として予め知識ベース中に定義されていると仮定したが，知識ベース側で未定義なリテラルの処理要求が起こったような場合，知識ベースにおける同値類を更新していくような機能が必要になる．また，代表元が一意に定まらないような場合，変換後のプログラムは複数の代表元によって記述されている必要が生じ，それだけ冗長性が増えることになる．今後，これらの課題を含めて検討を続けていく．

謝辞

本研究を進めるに当たり，貴重なコメントを頂いた横田一正，坂井公，松本裕治 各氏，並びにＫＢＭプロジェクトの諸兄に感謝します．

参考文献

[Nakashima 84] Nakashima,H.:"Knowledge Representation in Prolog/KR", Proc. of Int. Symp. on Logic Programming, pp.126-130, 1984.

[Ishizuka 85] Ishizuka,M. and Kanai,N.:"Prolog-ELF Incorporating Fuzzy Logic", New Generation Computing, vol.3, No.4, 479-486, 1985.

[Ait-kaci 86] Ait-kaci,H. and Nasr,R.:"LOGIN: A Logic Programming Language with Builtin Inheritance", Jour. of Logic Programming, vol.3, No.3, pp.185-215, 1986.

[Porto 86] Porto,A.:"Semantic Unification for Knowledge Base Deduction", Proc. of Workshop on Foundations of Deductive Databases and Logic Programming, 1986.

[Mukai 86] Mukai,K.:"CIL reference mannual", ICOT-TR, 1986.

[Yamaguchi 86] Yamaguchi,J.:"Boolean-valued Prolog", NEC-TR, 1986.

[Yokomori 86] Yokomori,T.:"On Analogical Query Processing in Logic Database", Proc. of 12th Int. Conf. on VLDB, pp.376-383, 1986.

[白井 86] 白井英俊:"状況意味論の立場から", 情報処理 vol.27, No.8, pp.887-896, 1986.

[Chang 73] Chang,C.L. and Lee,C.T.L.:"Symbolic logic and mechanical theorem proving",Academic Press, 1973.

[Han 86] Han,J.and Henschen,L.J.:"Handling Redundancy in the Processing of Recursive Database Queries", Nor-thwestern univ.-TR, 1986.

[Sagiv 86] Sagiv,Y.:"Optimizing Datalog Programs", Proc. of the Workshop on Foundation of Deductive Data-bases and Logic Programming, pp.136-162, 1986.