自然言語処理　62－8
（1987・7・24）

集合束縛変数に基づく意味表現とユニフィケーション

赤間　清

（北海道大学　文学部　行動科学科）

継承階層 prolog：ＰＡＬに導入されている集合束縛変数について記述する．それが，(1) Ｓ式の表現力を拡大する，(2) 推論を高速化する，(3) 認識の変化過程の記述に貢献する，などの理由で，自然言語処理などにおける意味表現において非常に有効な役割を果たす可能性を指摘する．また，複雑な対象の情報を比較し，それらが同一の対象とみなしうる可能性があるときに，それらの情報を統合して新しい対象記述を得るユニフィケーション・プログラムがＰＡＬのもとで容易に得られることを示す．

Semantic Representations based on Set Bound Variables and their Unifications

Kiyoshi Akama
(Hokkaido University, Sapporo-shi, 060, Japan)

Set bound variables (SBVs) are logical variables with set constraints. They are available in the inheritance hierarchy prolog : PAL, and class bound variables are the most useful SBVs in natural language processing. SBVs extend the expressive power of S-expressions, and they are very useful in representing the meaning of sentences and multiple concepts (for example, "mother" is represented by (*1^woman (<==mother *2^human)) which means that a woman who is a mother of a person) as well as simple concepts (for example, "man" is represented by *1^man). This enables us to make clear the process of semantic analysis of sentences, which merges them to get the meaning of the whole sentence starting from the meanings of the words in the sentence. This paper also gives a unification algorithm to unify two semantic representations into a single semantic representation when they can be assumed to be the same object.

1．　まえがき

継承階層 prolog：ＰＡＬの集合束縛変数について記述し，それが自然言語処理などにおける意味表現において非常に有効な役割をはたしうることを指摘する．

2．　集合データ型

2．1　集合データ型

継承階層 prolog：ＰＡＬの集合データ型は（現在のところ），Ｆ型，Ｉ型，Ｃ型の３種類存在する．Ｆ型は，アトムの有限集合を表わすデータで，

｛〈アトム〉〈アトム〉 … ｝

という形で書かれる（Ｆは Finite Set のＦである）．たとえば，

｛dog cat pig elephant｝

は，dog，cat，pig，elephant の４つの要素からなる集合を表わすＦ型データである．Ｉ型は，閉開区間（これはもちろん特別な形の実数の集合である）を表わす集合データ型である（ＩはInstance のＩである）．Ｉ型は一般には，

［〈左端数〉〈右端数〉］

の形に書かれる．ここで〈左端数〉と〈右端数〉はともに（システムがサポートする範囲での）数であり，〈左端数〉は〈右端数〉より小さいものとする．このＩ型データは，〈左端数〉以上で〈右端数〉より小さい実数全体の集合を表わす．たとえば，

［1987　2000］

は，

｛ x ｜ 1987 ≦ x ＜ 2000 ｝

なる閉開区間を意味する．Ｃ型の集合（クラス）は，継承階層に依存してアトムの集合を指定する（Ｃは Class のＣである）．これを用いるには，ユーザーがあらかじめ，組込述語 asc，asi などで継承階層を宣言する必要がある．asc，asi による宣言の一般形は，

（asc〈子クラス名〉〈親クラス名〉）

（asi〈子インスタンス名〉〈親クラス名〉）

である（親子の順序が，以前の論文［２など］から上記のように改められた）．たとえば，

```
(asc car vehicle)
(asc bicycle vehicle)
(asi mycar car)
(asi mybic bicycle)
(asi bic1 bicycle)
```

は，つぎのような継承階層を意味する．

```
vehicle--+-- car --------[mycar]
         |
         +-- bicycle  ---[mybic  bic1]
```

この継承階層は，各クラスに対して次のようにアトム（インスタンス）の集合を対応させる．

car = ｛ mycar ｝
bicycle = ｛ mybic bic1 ｝
vehicle = car ∪ bicycle
　　= ｛ mycar mybic bic1 ｝

2．2　積の演算

上に述べたように，集合データ型のデータは，それぞれ１つの集合とみなすことができる．したがって，集合データ型の２つのデータに対して，集合の和集合，積集合などを求める演算を考えることができる．とくに重要なのは，積集合をもとめる演算である．それは積集合を求める演算（以下では簡単のために，単に積という）が，後述の集合束縛変数どうしのユニフィケーションの主要部分となるからである．Ｆ型，Ｉ型，Ｃ型の各集合データ型は，積が空集合でないかぎり積の演算について閉じている．また同じ型に属する集合データ型どうしの積はきわめて簡単に求めることができる．

たとえば，Ｆ型のデータである　ｆ１＝｛dog cat elephant｝と　ｆ２＝｛dog tulip elephant cherry｝の共通部分は，ｆ３＝｛dog elephant｝である．この積の演算は，〈Ｆ型〉の集合データ型の内部表現を

bit 列としておけば非常に高速である．この例では，もし bit 列の各桁が左から順に，cherry, dog, cat, tulip, elephant に対応してコード化されていたとすると，ｆ１，ｆ２，ｆ３はそれぞれ次のようになる．

ｆ１＝ "01101"
ｆ２＝ "11011"
ｆ３＝ "01001"

積の演算はあきらかに bit演算の and に対応する．Ｉ型の集合データ型どうしの積も簡単に求まる．Ｉ型のデータ，[x1　y1] と [x2　y2] の積が空集合にならないのは，

x2 < y1　かつ　x1 < y2

のときで，それらの積は，

[max(x1, x2)　min(y1, y2)]

となる．Ｃ型のデータである animal と dog の積は dog である．これは dog が animal の下位概念であることを確かめることによって結論される．一方のクラスが他方のクラスの下位概念であることは，そのクラスから継承階層を上の方にたどって他方のクラスに行き着くことを確認することで判定できる．したがってユニフィケーションのコストは，最悪の場合でも，両方のクラスから極大クラスまで上方向に継承階層をたどる手間以下で済む［1］．

<Ｆ型> ＝ { <アトムの列> }
<Ｉ型> ＝ [<数> <数>]
<Ｃ型> ＝ <アトム>
<アトムの列>
　＝ <アトム> | <アトム> <アトムの列>

図１　集合データ型のシンタックス

３．　集合束縛変数とユニフィケーション

３．１　集合束縛変数

＊を先頭に持つシンボル・アトムで変数を表わすと約束する（＊だけでもよい）．

<変数> ＝＊　|　＊<シンボル・アトム>

変数は任意のＳ式と unify する．拡張 prolog ：ＰＡＬでは，そのような通常の意味の変数の他に，集合束縛変数を利用することができる．集合束縛変数は，unify しうるＳ式の範囲が（Ｆ型，Ｉ型，Ｃ型の）集合データ型のデータによって制約をうけている変数である．集合束縛変数と対比するとき，任意のＳ式と unify しうる変数を，通常変数と呼ぶことがある．集合束縛変数は一般に「^」を用いて

変数^束縛

のように書かれる．たとえばＦ型のデータ：

{ dog cat pig lion }

に束縛された変数 *v は，

*v^ { dog cat pig lion }

となり，「dog または cat または pig または lion である *v」を表わす．同様にＩ型のデータ：

[1987　2000]

に束縛された変数 *v は，

*v^[1987　2000]

となり，「1987以上2000未満である実数 *v」を表わす．また，animal がＣ型のデータであるとき，animal に束縛された変数 *var は，

*var^animal

と書かれる．これは「動物である *var」を表わす．

３．２　集合束縛変数どうしのユニフィケーション

集合束縛変数は，Ｆ型，Ｉ型，Ｃ型 のいずれか１つの集合データに束縛されている．したがって集合束縛変数も，集合データの型名を流用して，Ｆ型，Ｉ型，Ｃ型と分類することにする．

Ｆ型の集合束縛変数どうしのユニフィケーションは，双方の変数を束縛するＦ型の集合が共通部分を持つとき成功し，その共通部分を表わすＦ型の集合に束縛されたＦ型の集合束縛変数になる．このことはＩ型どうしやＣ

型どうしの場合にもまったく同様である．

各型の例をあげる．

(1) *1^{ akira satoru hirosi } と *2^{ hirosi minoru satosi } は unifyして，*1 = *2, *1^{ hirosi }, *2^{ hirosi } となる．

(2) *1^[1887 1997] と *2^[1990 2000] は unify して，*1 = *2, *1^[1990 1997], *2^[1990 1997] となる．

(3) *1^animal と *2^dog は unify して，*1 = *2, *1^dog, *2^dogとなる．

異なる型のクラス束縛変数は unify しないことにする．これは，ユニフィケーションの高速性を保つための措置である．

3．3 集合束縛変数とアトムのユニフィケーション

集合束縛変数とアトムのユニフィケーションについて定義する．集合束縛変数とアトムが unify するのは，そのアトムが，変数を束縛している集合の要素であるときである．そしてその結果，変数は集合束縛変数であることをやめ，そのアトムと等しくなる．たとえば，F型の集合束縛変数：

*1^{ dog cat lion }

はアトム lion と unify し，*1 = lion となる．I型の集合束縛変数：

*1^[1982 1990]

はアトム 1988 と unify し，*1 = 1988 となる．C型の集合束縛変数である *1^dog や *2^animal は poti と unify し，その結果，それぞれ，*1 = poti, *2 = potiとなる．これらのユニフィケーションに要する計算コストは，集合束縛変数どうしのユニフィケーションの場合と同様に，ごくわずかである．

3．4 継承階層 prolog におけるユニフィケーション

継承階層 prolog におけるすべての対象はS式である．それは次のように定義される．

〈S式〉 = 〈アトム〉 | 〈通常変数〉 | 〈集合束縛変数〉 | 〈コンス〉
〈アトム〉 = 〈シンボル〉 | 〈数〉
〈シンボル〉 = 〈クラス〉 | 〈インスタンス〉 | 〈その他のシンボル〉
〈コンス〉 = （〈S式〉 ・ 〈S式〉）

S式のユニフィケーションを定義する．

(1) 〈アトム〉と〈アトム〉 ： まったくおなじ場合にのみ成功．

(2) 〈アトム〉と〈通常変数〉 ： 無条件に成功．変数はそのアトムになる．

(3) 〈アトム〉と〈集合束縛変数〉 ： アトムが束縛集合の要素である場合にのみ成功．変数はそのアトムになる（上述）．

(4) 〈アトム〉と〈コンス〉 ： 無条件に失敗．

(5) 〈通常変数〉と〈S式〉 ： 無条件に成功．変数はそのS式になる．

(6) 〈集合束縛変数〉と〈集合束縛変数〉 ： ２つの集合束縛変数が同じ型に属し，２つの束縛集合が空でない積集合を持つ場合に成功．集合束縛変数はその積集合に束縛された同じ型の集合束縛変数になる（上述）．

(7) 〈集合束縛変数〉と〈コンス〉 ： 無条件に失敗．

(8) 〈コンス〉と〈コンス〉 ： car部分とcdr部分のそれぞれのユニフィケーションが成功するとき全体のユニフィケーションも成功する．

4． 集合束縛変数の意義

4．1 パターンの拡張としての集合束縛変数

prolog が他の言語に優越する特徴の１つは，prolog のパターンによる記述力の高さである．しかしどんなに言語を拡張しても，すべてのパターンを提供することは原理的にありえない．したがって prolog の能力を拡大するには，より有効なパターンが何である

かを見出し，それを追加することが必要である．集合束縛変数は，パターンの拡張として極めて自然である．

4．2　高速化の手段としての集合束縛変数

prolog の計算の基本は，n個の条件をすべて満足する対象を求めるものである．もしそれぞれの条件に対して，その条件を満足する対象の集合が（1組の）パターンとして表現できているならば，n個の条件を満たす対象は（n組の）パターンのユニフィケーションで求められる．しかしもし（1組の）パターンで表現できない条件があれば，それはいくつかの（組の）パターンの和で代用される．和でかかれた部分は or選択肢になる．or 選択肢をすべてつくすために backtrack のメカニズムが採用されている．

したがってこの枠組みのもとで推論の高速化をもたらすには，十分高速なユニフィケーションを保証しつつ，有効なパターンを増加させればよい．新しいパターンは backtrack を抑えた分だけ高速化に貢献する．

集合束縛変数は，その最も単純な応用例である．すでに述べたように，F型，I型，C型ともにユニフィケーションのコストは非常に小さく，応用範囲は広い．

4．3　S式の進化

lisp の世界における主要な表現の手段はS式である．そこでのS式は以下のように定義されている．

〈第1種S式〉 ＝ 〈アトム〉｜
　　〈第1種コンス〉

〈第1種コンス〉
　＝ （〈第1種S式〉 ・ 〈第1種S式〉）

S式シンタックスを持つが集合束縛変数を持たない標準的 prolog（例えば prolog/KR）における主要な表現の手段は変数を含むS式である．そこでのS式は以下のように定義されている．

〈第2種S式〉 ＝ 〈アトム〉｜
　　〈通常変数〉｜
　　〈第2種コンス〉

〈第2種コンス〉
　＝ （〈第2種S式〉 ・ 〈第2種S式〉）

さらに，S式シンタックスと集合束縛変数を持つ prolog（例えばPAL）における主要な表現の手段は変数と集合束縛変数を含むS式である．そこでのS式は以下のように定義されている．

〈第3種S式〉 ＝ 〈アトム〉｜
　　〈通常変数〉｜
　　〈集合束縛変数〉｜
　　〈第3種コンス〉

〈第3種コンス〉
　＝ （〈第3種S式〉 ・ 〈第3種S式〉）

DEC-10 prolog などの主要な表現手段は項であるが，これは第2種S式と同等の表現力を持つ．

第1種S式，第2種S式，第3種S式と表現力は拡大する．しかしそれは有効な拡大なのだろうか．拡大された表現力は適切な応用を見出されてはじめて実質的な意味を持つ．

変数が使えない第1種S式に比べて，第2種S式が実際にずっと有効であることは，lisp 対 prolog の比較の中で明らかにされ，未定構造の利用という形で定式化された．では第2種S式と第3種S式の比較はどうであろうか．第3種S式は，以下で述べるような表現を想定すれば，自然言語処理における意味表現においてきわめて有効な働きをするものである．その有効性を重視するとき，われわれは，第1種から第2種への飛躍と同等の飛躍を第2種から第3種の間にも認めてよいと考える．この意味で，第1種から第3種のS式の変化は，lisp，prolog そして palの能力の段階的増加を明確に示すものとなる．

4．4　認識の変化過程と集合束縛変数

第1種S式と第2種S式と第3種S式の違いは，われわれが実現すべき主要な情報処理の性格が以下のように変化していることを明

確に認識するとき，本質的であると思われる．それを説明する．我々の扱う対象は，究極的には，すべて第１種Ｓ式で表現できる．だから第１種Ｓ式を効率よく操作する言語を提供すれば十分であろう．これがおそらく，第１種Ｓ式を操作する言語である lisp を考案するときの直観であったろう．そこでは，われわれが行うべき主要な情報処理のイメージは，第１種Ｓ式で表現されたデータを，変換する規則を表現する知識であるプログラムによって加工することであった．しかし prolog が lisp に優越しうる本質的な理由のひとつは，第１種Ｓ式から第２種Ｓ式への拡大（扱うデータが拡大され，データの中に変数が入ったこと）である．もしわれわれが計算機で行うべき主要な情報処理のイメージを次のように変更することを認めるならば，その重要性は非常に大きいものとなる．情報処理の新しいイメージとは，いろいろな情報や知識を総合して新しい情報や知識をつくりだすことである．そこでの情報や知識は対象の部分的な特徴を反映したものに過ぎないので，（少なくとも構文的には完全知識の表現を想定している）第１種Ｓ式では容易に表わしえない．第２種Ｓ式ならば，未知の部分を変数にすることによって，計算（推論）のある時点での認識の状態をうまく表現できる場合がある．つまりprologは第２種Ｓ式（またはその等価物）によって，情報処理の新しいイメージに（図らずも）対処できたのである．しかしそれは（意図的でなかっただけに）やはり十分にはなりえなかった．未知の部分を変数にする方法だけでは，認識の変化はあまりうまくは表現できない．動物であるとだけわかっていたものが，別の情報によって実は犬だと分ったというような変化を，第２種Ｓ式は直接的には表現できない．このようにして第２種Ｓ式から第３種Ｓ式への飛躍の重要性があきらかとなる．

５．　集合束縛変数に基づく表現

５．１　対象の表現

自然言語処理のために使われる意味表現や，意味解析方法は，いろいろな研究者によって数多くの提案がある．われわれは，そのうちのいくつかを参考にしながら，継承階層 prolog と（第３種）Ｓ式を基礎とした独自の，知識表現方法の開発と自然言語処理システムの実現を進めつつある．われわれは，既存のいくつかの優れた研究の成果を統合し，より明確化し，発展させていく上で，われわれの方法が，非常に有効であると考えている．

ここでは，集合束縛変数を用いた表現の方法を１つ与え，そのユニフィケーションの定義とプログラムを示すことにする．その表現は，われわれが自然言語処理で用いている意味表現の一種であり，次の<対象>で定義される（第３種）Ｓ式である．

```
<対象>   =   ( <分子> )  |
             ( <分子>  ・  <関対リスト> )
<分子>   =   <アトム> |
             <クラス束縛変数> |
             <通常変数>
<関対リスト>  =   ( <関対> )  |
             ( <関対>  ・  <関対リスト> )
<関対>       =   ( <関係> ・ <対象> )
<関係>       =   <述語>
```

ここで，<述語>，<アトム>などは

```
<述語>      = <シンボル>
<アトム>    = <シンボル> | <数>
<シンボル>  = <クラス> |
              <インスタンス> |
              <その他のシンボル>
```

なる関係を満たしている．

５．２　表現例

対象の簡単な例をあげる．

```
(*hl^man)
```

------(1)

は，「ある男 *h1」に対応する意味表現であり，

```
(*h2^human
    (name santarou)
    (address *p2^tokyo))
                    ------(2)
```

は，名前が三太郎で，住所が東京である人間 *h2」を意味する．また，

```
(*h3^man
    (name santarou)
    (address *p3^meguro_ku)
    (age  37)
    (work *k3^kaisha_in
        (kaisha *s3
           (address *a3^chiyoda_ku))
        (joushi *j3^man)))
                    ------(3)
```

は，「名前が三太郎，住所が目黒区，年齢が３７，職業が会社員で，その会社の住所は千代田区，上司は男性である男 *h3」であり，

```
(*h4^human
  (address *p4^tokyo)
  (age *n^[30 40])
  (work *k4^salary_man
        (joushi *j4^human
                (name mannen)
                (mibun katyou))))
                    ------(4)
```

は，「住所が東京，年齢が３０代，仕事はサラリーマンで，その上司が万年という名前の課長である人間*h4」を表現している．

もちろんこれらの表現の背後には，クラスやインスタンスを定義する記述が必要である．ここでは，それらは次のように定義してあるものとする．

```
(defc all (object place sosiki shokugyo))
(defc object (animal plant machine))
(defc animal (dog cat bird human))
(defc human (man woman))
(defc place (japan america))
(defc japan (tokyo sapporo osaka kyoyo))
(defc tokyo (chiyoda_ku meguro_ku
                shibuya_ku))
(defc sosiki (kaisha kantyou))
(defc kaisha (yuugen_kaisha
                kabushiki_kaisha))
(defc shokugyo (salary_man))
(defc salary_man (kaisha_in))
(defi man (santarou))
```

これらは，後にユニフィケーションの例を説明するためだけに作られたもので，われわれが自然言語処理の意味表現として推奨するものとは異なる．これらの全体の意味はあきらかと思うが，ただ，クラス place の子孫クラスについて説明しておくほうがよいかもしれない．それらの地名は普通は全体－部分の関係を持つとされ，継承階層に組込んで，is_a の関係で表現するのは奇異に映る可能性があるからである．しかしそれらは次のように考えれば矛盾を引起こすものではないことが理解できる．すなわち，ここでの地名は，それが部分として包含する場所全体の集合なのである．たとえば，tokyo は，chiyoda_ku や meguro_ku やshibuya_ku のなかのすべての場所に対応している．そのように考えるとそれらは集合の包含関係で結ばれ，is_a 階層のなかに置けることが理解できよう．そのとき，

```
(address *p4^tokyo)
```

は，住所が東京のどこかであることを表現していることになる．

５．３　対象どうしのユニフィケーションのアルゴリズム

ＡとＢを，ある物語のなかに登場する人物や動物としよう．その物語のなかでＡやＢに対する記述は，いろいろな形で語られる．しかしそれぞれの記述だけを見ても，表面的にはどれがＡに対する記述で，どれがＢに対する記述か明示されているとは限らない．たとえば「私は秋田犬を飼っている．我が家の愛すべき動物は恐ろしく寂しがりやである．」などという文があるとする．おそらく第２文

の「寂しがりやの動物」とは「私の飼っている秋田犬」のことであろう．このとき，それらが一致するとの仮説のもとに，双方の情報を総合すれば，「私が飼っている」のは，「寂しがりやの秋田犬」であるとの理解に至る．つまり，２つの対象記述が一致するかもしれないという仮説をたて，それが可能な場合にはそれらを総合してより詳細な記述を得ることは，物語理解の必須の一部分である．そのような情報処理を行うための述語名をここでは unifyとする．unify 述語は，われわれの枠組みの中で簡単につくることができる．図２がその最も簡単な定義である．

```
(as (unify (*o . *r1)
           (*o . *r2)
           (*o . *r))
    (unify_1 *r1 *r2 *r))

(as (unify_1 *x *y ((*p . *oz) . *zz))
    (memrest (*p . *ox) *x *xx)
    (memrest (*p . *oy) *y *yy)
    (unique *p)
    (cut)
    (unify *ox *oy *oz)
    (unify_1 *xx *yy *zz))
(as (unify_1 *x *y *z)
    (append *x *y *z))

(as (memrest *x (*x . *r) *r))
(as (memrest *x (*y . *r) (*y . *rr))
    (memrest *x *r *rr))
```

図２　簡単な unify プログラムの例

上で挙げた対象をつかって，

```
(unify (1) (2) *unif)
```

と unify 述語を起動すれば，*unif として

```
(*h2^man
    (name santarou)
    (address *p2^tokyo))
```

が得られる．すなわち(1)と(2)の二人の人間は同一視することが可能で，そのときの人物像が *unif に求まったことになる．同様に(3)と(4)を与えて unify を呼出せば，次の対象が得られる．

```
(*h3^man
    (name santarou)
    (address *p3^meguro_ku)
    (age  37)
    (work *k3^kaisha_in
        (kaisha *s3
            (address *a3^chiyoda_ku))
        (joushi *j3^man
                (name mannen)
                (mibun katyou))))
```

６．　むすび

本論文で示した集合束縛変数に基づく意味表現を用いて，自然言語処理のシステムの作成の試み［３］が成功裏に進行中である．

文　献

［１］　赤間清：概念階層クローズ・インデキシング，情報処理学会，知識工学と人工知能研究会資料，51-2，pp.9-16 (1987)

［２］　赤間清：ＰＡＬ：継承階層を扱う拡張PROLOG，情報処理学会論文誌　Vol.28 No.4 pp.27-34 (1987)

［３］　赤間清：継承階層 prolog による自然言語処理，情報処理学会，自然言語処理研究会，（本号）(1987)

［４］　井佐原均，石崎俊，半田剣一：文脈解析システムにおける概念表現とその照合法，情報処理学会第３２回全国大会講演論文集，6M-2，pp.1283-1284 (1986)

［５］　Sowa,J.F. : Conceptual Structures, Addison-Wesley P.C., p.481 (1984)

［６］　Ait-kaci,H. and Nasr,R : LOGIN: A Logic Programming Language with Built-in Inheritance,the journal of logic programming, Vol.3, No.3,pp.185-215 (1986)