自然言語処理 73−16
(1989. 6. 30)

意味計算 I

A Semantic Calculus I

－認識の逐次更新過程と集合束縛変数－

The Process of Successively Updating Interpretations and Set Bound Variables

赤間 清

Kiyoshi Akama

北海道大学工学部情報工学科

Dept. of Information Engineering, Faculty of Engineering, Hokkaido University

［あらまし］ 現在の自然言語処理研究にとって最も重要な課題の１つは，自然言語の意味処理の基礎的な部分の明快な技術体系を提示することである．そのような技術体系を意味計算(semantic calculus)と呼ぶことにする．我々は，制約を扱う論理型の知識表現言語 PALとその上に作られた自然言語処理実験システム TALK と広範な知識表現を統一的に扱うGLPの理論を基礎として，意味計算の体系の構築を目指している．それらを懸引する考え方の１つは，「自然言語の理解過程は認識の逐次更新の過程である」という観点である．本論文では，PAL の提供する集合束縛変数が，認識の逐次更新過程としての意味処理を達成するために単純で強力な道具となることを示し，PAL/TALK/GLP のアプローチが意味計算を確立する上で重要な役割を果すことを示唆する．

[Abstract] Theory and techniques for basic semantic analysis in natural language processing should be constructed and commonly used. We call this a semantic calculus. In order to construct a semantic calculus, we give a new approach consisting of PAL (a logic programming language which can deal with constraints), TALK (an experimental natural language processing system implemented in PAL) and GLP theory (a theory of generalized logic programs that can give the theoretical foundation for many extensions of prolog including PAL). This approach is based on the observation that the process of understanding natural language sentences is a process of succesively updating interpretations. In this paper we show that the set bound variables in PAL is a simple and powerful tool for realizing such processes and suggest that the approach of PAL, TALK and GLP plays an important role in constructing the semantic calculus.

１． まえがき

現在の自然言語処理研究にとって重要な課題の１つは，自然言語の意味処理を明快に体系化することである．ここでは，自然言語の意味処理の奥深い部分まですべて体系化することを要請しているわけではない．自然言語の意味処理が永遠の課題であることは承知している．ここで要請しているのは，意味処理の基礎的な部分の体系化である．自然言語処理のうち，文法や構文解析などに関する部分はかなり体系的な研究の蓄積がある．それは簡単にいえば，文脈自由文法という明快な舞台が共有されているからである．しかし意味に関する研究はそれに比較すれば少なく，体系化が進んでいるといはいえない．共有されたパーザーのうえに思い思いの意味処理プログラムを工夫しているというのが実情に近いであろう．意味処理があまり共有化されていないために，各研究者は多くの技法を再発見しなければならない．この状況を打開するには，意味処理の奥深い部分に過度にこだわることなく，意味処理の基礎的な部分に関して明快に共有できる技術的体系を提示することが有効である．我々はそのような技術体系を意味計算(semantic calculus)と呼ぶ．

では意味計算を意味処理全体からうまく切出し，エレガントな体系として構築するにはどうすればよいだろう

か．この問題に答えるために我々は図１のような構成を持つ一群の研究によって，新たな自然言語処理研究のアプローチを提案している．それは，制約を扱う論理型の知識表現言語 PAL[赤間87ab]とその上に作られた自然言語処理実験システム TALK [赤間88b,89ab]と広範な知識表現を統一的に扱うGLP(generalized logic program)の理論[赤間88c,89cdefg]からなる．PALとTALKの方法は認識の動的な変化過程を，エレガントにしかも高速に実現する具体的方法を提示している．またGLPの理論は PAL／TALKの方法の基礎的な構造が厳密かつ明快な裏付けを持つことを保証している．

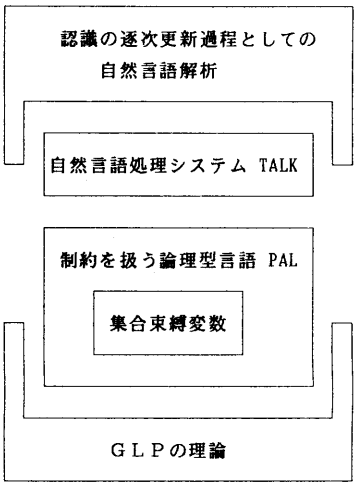

図１　認識の逐次更新過程としての
自然言語解析に対する
新しいアプローチの枠組み

このアプローチを懸引している主要な考え方の１つは，「自然言語理解過程は認識の逐次更新過程である」という観察である[赤間 88b]．認識の逐次更新という視点は自然言語解析にとってきわめて基本的なものであり，自然言語処理や知識表現の数多くの研究にこの観点からの接近あるいは反映が見られる [Mellish 84, Sowa 84, Mukai 85, 井佐原 86, 大澤 86, 橋田 88, 丸山 88]．しかし既存の自然言語処理の研究は，この観点から見た理解過程を充分に明快に認識したとはいえない．実際，既存の自然言語処理システムの実現手法を見れば，この観点を達成するに不向きな方法をいくつも発見することができる．これを改善するためには，認識の逐次更新を適切に表現できる知識表現系[赤間 88b]の採用から始めねばならない．

我々の方法は，制約を扱う論理型の知識表現言語 PAL を基礎としている．PAL は認識の逐次更新過程の実現を促進する．PAL の中心は集合束縛変数という単純で強力な部品である．我々が切出す意味計算とは，おおよそ，集合束縛変数やその応用技術などで実現できる意味処理の範囲である．それは，概念などを表現するクラスの間の計算で達成できる意味処理の範囲を含む．

意味計算の技術体系はすべてが完成したわけではない．しかし我々は上記の枠組みにそって，すでに文解析から文生成に至る自然言語処理システムを試作している．その結果，意味計算の核となる部分の基礎技術は得られている．我々は，それらの諸技術を，深化し発展させながら，意味計算のタイトルの下に提示して行きたいと思う．

２．　認識の逐次更新過程への接近

自然言語理解過程とは，情報を逐次的に受取って，認識を動的に変化させていく過程（図２）とみなすことができる．図３はその例で，ある動物に対する認識が次々に変化している．読み手は逐次的に文字列を手に入れ，文字列の解釈などからなる自分の認識をどんどん変化させる．全体の文の解釈は，最後まで文を読み終えたとき，読み手の頭のなかに築かれた認識の一部分として得られる．これは，文章（文の列）を読む場合も同様と考えられる（以下特に必要ないかぎり文で代表させる）．

通常は暗黙のうちに文を前から読むものと仮定することが多いが，認識の逐次更新というモデル化は，文の各部分を読む順序とは直接的には関係なく成立するものである．それは読む文が多くの部分から成り立っており，全体の文の解釈を，各部分が示す解釈と情報を順に組み上げて，整合をとりながらだんだん大きくしていく過程を経ないで，一挙に合成することができないという事情からの自然な帰結にほかならない．

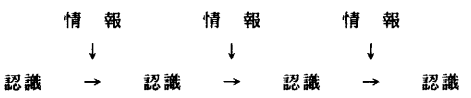

図２　自然言語の意味理解過程は　認識の逐次更新の過程　である

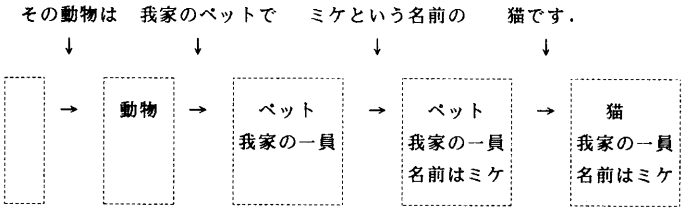

図３　認識の逐次更新の例

本論文では，認識の逐次更新過程を次のような２つの過程の合成としてとらえる．１つは，「もらった」と「もも」の解釈から「もらったもも」の解釈ができ，さらに「もらったももを食べる」の解釈に至るような，文法構造にそった解釈の合成過程である．もう１つは，「もも」の解釈（股または桃の実または桃の木）が「もらう」との関係で制約を受けて「桃の実または桃の木」に限定され，さらに「食べる」の関係で制約をうけて「桃の実」に限定されるように，解釈の構成要素が制約にしたがって互いに変化していく過程である．本論文では，認識の逐次更新過程を①文法構造にそった解釈の合成過程と②制約にしたがって部分解釈が互いに変化していく過程の合成としてとらえ，いずれか一方だけでは不十分との立場で考察している．

このうち②の過程に注目すべきである．①の文法構造にそった解釈の合成過程は，どんな自然言語処理システムでも実現されている．しかし解釈に出現する概念などが文法構造にそった解釈の合成過程で変化していくことは軽んじられている．概念の階層構造をどう扱うかがそれを端的に示している．既存の自然言語処理システムでは，概念の階層構造とは，あるクラスまたはインスタンスｘが上にたどって性質を受継ぐためのものである．ｘ自体は変化しない．それに対して PAL では，ある概念ｘはその祖先クラスの性質を受継ぐことも，性質によって概念ｘ自身が変化することもできる．この概念の変化こそが，認識の逐次更新の要である．この特質によって，PAL/TALK の自然言語処理は，既存の多くの方法とは異なる体系をつくるのである．

３．　認識の表現方法

３．１　集合束縛変数の定義と表現力

PAL の集合束縛変数とは，変数ｖと制約ｓのペアである．制約ｓは１つの集合であり，変数ｖがｓの範囲の対象になりうることを表現している．集合束縛変数の例を図４に示す．集合束縛変数を定義するためには，制約集合になりうる集合の族Ｋを与えればよい．ユニフィケーションを高速化するためには，集合族Ｋは積の演算で閉じていることが望ましい．なぜなら，２つの集合束縛変数のユニフィケーションは束縛集合の積の演算を引起こすからである．

```
Ｆ型　*hobby^{fishing swimming}　　釣または泳ぎのどちらかである *hobby
Ｉ型　*age^[20 30]　　20 以上 30 未満である *age
Ｃ型　*animal^(+ dog cat)　　犬または猫である *animal
```

図４　集合束縛変数の例

集合族Ｋを１つ定めるごとに，一群の集合束縛変数が与えられる．たとえば，ある有限集合の空でない部分集合全体のなす族はＦ型の集合束縛変数を与える．また，空でない区間全体の集合はＩ型の集合束縛変数を与える．以下では，自然言語処理に主要な役割を演じるＣ型の集合束縛変数について述べる．

Ｃ型の集合束縛変数は，asc や asi 述語などで定義されるクラスやインスタンス（１個の元からなる集合と考える）などから和，積，差の有限回の演算で構成できるすべての空でない集合のなす集合族K_cから作られる．K_cは和，積，差について閉じている．以前の論文[赤間89b]でＣ型のＭ型などと呼んだものも，ここではＣ型と呼び直している．

PAL には束縛集合の間の演算として和(+)，積(x)，差(-)などが用意されているので，クラスやインスタンスから出発して集合族K_cのすべての集合を記述すること

ができる．たとえば，animal，dog，cat，pig，human などが asc やdefcで定義されたクラスであるとき，

(+ dog cat pig)

は，「犬または猫または豚」を意味し，

(- animal human)

は「人間以外の動物」を意味する．またpoti が dogの，mike が catのインスタンスであるとき，

(+ pig mike (- dog poti))

は「豚またはミケまたは，ポチ以外の犬」を表す．

このような式は defs 述語の第２引数としても書くことができる．defs 述語はシンボルに対して集合を割当てるものである．たとえば，

```
(defs dislike
      (+ pig mike (- dog poti)))
```

はシンボル dislike に

```
(+ pig mike (- dog poti))
```

なる集合を割当てる．defs で集合を割当てられたシンボルをセットと呼ぶことにする．

クラスやインスタンスやセットだけでなくそれらを和，積，差で有限回演算した式はすべて，集合を限定する制約になったり（Crestrict 述語などによる），図４のように直接的に集合束縛変数を作ることができる．この枠組みにより，対象をとても自由に表現することができることがわかる．それは[奥村 89] における個体の選言と否定などを特殊な例として含む．実際，個体の選言は，たとえば *^(+ mike poti) のように表現可能である．また個体の否定は，all をすべてのクラスのスーパークラスとするとき，all と個体の差を使って表せばよい．たとえば，

*^(- all mike)

はミケ以外の対象を表す．

以上では，変数ｖに付随した制約ｓに関する表現力を説明したが，集合束縛変数にはもう１種の表現力がある．それはあるＳ式にいくつかの集合束縛変数が出現するとき，変数部分を一致させることによって，それらの同一性を表現するものである．

```
<対象> ::= ( <分子> ・ <関係対象ペアリスト> )
<分子> ::= <アトム> | <集合束縛変数> | <変数>
<関係対象ペア列> ::= ( ) | ( <関係対象ペア> ・ <関係対象ペア列> )
<関係対象ペア> ::= ( <関係> ・ <対象> )
<関係> ::= <シンボル>
```

図５　TALK における＜対象＞の定義

```
(*1^SENTENCE
   (pred see)
   (mod past)
   (theme *2^human (==>name *3^hidemi))
   (agent *2^human (==>name *3^hidemi))
   (object *4^woman (<==mother *2^man)))
```

(a) 秀美は母を見た

```
(*1^SENTENCE
   (pred eat)
   (mod present)
   (object *2^peach_food
         (z *4^SENTENCE
            (pred get)
            (mod past)
           (object *2^peach_food))))
```

(b) もらったももを食べる

図６　TALK における文の解釈の表現例

３．２　解釈の表現

TALKで解釈を表現する方法[赤間89abなど]は図５の<対象>で定義されるシンタックスを持つＳ式である．例を図６にあげる．ここでの表現は，名詞句や文などに対応する解釈を統一的に扱っていることに注意したい．たとえば，「秀美は母を見た」の解釈は *1^SENTENCE を分子とするＳ式である．その

```
(agent *2^human (==>name *3^hidemi))
```

の部分は（ここではhidemiはnameのsubclassとする），

```
(agent . (*2^human
          (==>name *3^hidemi)))
```

と書換えることができるから，「秀美という名前の人」という意味の

```
(*2^human (==>name *3^hidemi))
```

という構造が含まれていることがわかる．これは「秀美」

に対する解釈であり，やはり図５の＜対象＞で定義されるシンタックスを持つ．認識の逐次更新は文の部分的な解釈を合成して次第に大きい部分の解釈を作り，ついには全体の文あるいは文章の解釈を求めるものであるから，それらの各段階での解釈（名詞句や文に対する解釈）が同じシンタックス（＜対象＞）を持つことは有用である．

第２の注意点は，ここでの表現が，変数の一致によって対象の同一性を表現していることである．たとえば，

```
(*2^human (==>name *3^hidemi))
```

に現れた *2^human が，

```
(object *4^woman
        (<==mother *2^human)))
```

にも現れており，それにより「母」とは秀美の母であることが表現されている．

４．　集合束縛変数と逐次更新過程

４．１　認識の逐次更新過程の必要条件

自然言語解析過程を認識の逐次更新過程として実現する場合に最も基本的で重要な点が少なくとも３つある．その第１はいくつかの可能な解釈をなるべく同居させるような知識表現を採用することである．第２は各解釈を局所的な制約条件を満たすように高速に更新することである．第３は，局所的な変更が全体として整合性をもつように高速に制約を伝播させて解釈の整合性を管理することである．これら３点を簡単に説明しよう．

①　可能な解釈を同居させる表現力

たとえば「太郎は花子にももをもらいました．」という文で，「もも」という単語の３つの解釈，すなわち「股」，「桃の実」，「桃の木」を別々に扱えば，文全体の解釈を合成するに至る途中の時点で整合性を失って失敗しbacktrack する運命である多くの中間的な解釈を別々に作ることになり，無駄な計算が増大する．これに対して，これらの解釈を同居させた意味表現（この例では「股または桃の実または桃の木」）を使うことによって，解析の計算量は著しく減少する[赤間89b]可能性がある．ここでの条件は，可能な解釈をすべて同居させることを要請してはいない．PAL/TALKの枠組み（GLPの枠組み）は，あまりにも同居しにくい解釈は，backtrack などを使った数え上げなどに任せることを禁止するものではない．

②　局所的な制約条件による高速な解釈の更新

自然言語文を読み進める過程では情報がつぎつぎに手に入り，それによって認識をすばやく変化させる必要がある．たとえば「太郎は花子にももをもらいました．」という文は「もも」の解釈に「もらう」という動詞と結びつくという制約を与える．このような制約によって「もも」の解釈を「股または桃の実または桃の木」から「桃の実または桃の木」へ高速に変更する[赤間89b]ことができねばならない．

③　高速な制約伝播による解釈の整合性管理

たとえば，「太郎は花子にもらったももを植えました」という文において，「もも」という単語が，「植える」との関係では桃の木として解釈され，「もらう」との関係では桃の実として解釈されるように，１つの文で同じ単語の解釈が異なってはならない．「もも」に対する解釈が文の解釈の中で１ヵ所にしか出現しないならば，いつも同じところが更新されるのでそこから解釈の矛盾は起らないが，「もも」という単語の解釈は複数箇所に出現しうるのである．実際 PAL/TALK では上記の文の解釈に，「もも」に対する解釈を２ヵ所に登場させている．複数個の「もも」の解釈を自然言語処理プログラムのレベルで管理することは非常に繁雑になるので避けるべきである．「もも」の解釈のある出現に与えられた制約をプログラム言語が自動的に他の出現に伝播して，整合性を管理してくれることがここでの要請である．

指示語の問題が絡んだ場合にも，同種の問題が起る．たとえば「太郎は花子にももをもらいました．彼はそれを植えました．」の文で，「それ」が「もも」を指すという仮説を採用する場合，次の３つの制約を考えに入れなければならない[赤間89b]．

(a)　「もも」は「もらう」の目的語

(b)　「それ」は「植える」の目的語

(c)　「もも」と「それ」は同じもの

これらのうち，(a)の制約にしたがえば，「もも」は股であってはならない．また(b)にしたがえば，「それ」は木や種でなければならない．「もも」と「それ」は別の単語なので，それらの解釈はそもそも別々に出現する．しかし，(c)にしたがえば，「もも」と「それ」同じ対象であり，それらの解釈は整合しなければならない．「もも」が桃の実であって，「それ」が桃の木であれば(a)(b)の制約は満たすが，それは(c)に反するから起ってはならない．このように(c)のような制約があれば，同一であるべき解釈の集合は動的に変化することになる．それは同一であるべき解釈の集合をユーザープログラムで管理する事をなおいっそう難しくする．そのようなことは知識表現言語に委ねユーザーはある解釈に部分的制約を加えるだけで，知識表現言語がうまく整合をとってくれることが望ましい．

①によれば多義語の可能な解釈はなるべく同居させることになるので，解釈の整合性が問題になる場合は非常に多い．したがって解釈の整合性管理を高速に行なえることは必須の要請である．整合的な解釈を高速に求めるためには，それぞれの局所的制約から得られる解釈の限定を高速に伝播しなければならない．

４．２　集合束縛変数の利点と逐次更新過程

ここでは，集合束縛変数が逐次更新過程の３つの要請を明快に達成する単純で強力な部品であることを示す．

集合束縛変数の第１の利点は，それが「ある程度分っているもの」を直接的に表現できることである．たとえば，*x^dog は「犬である *x」を意味するが，それは通常の変数 *x で表される「あるもの *x」よりは限定されており，定数 poti で表される「犬のポチ」までは限定されていない存在である．自然言語で言及されるものは，大部分は［ある程度わかったもの］であることを考えれば集合束縛変数の重要性が分るであろう．

この利点は，逐次更新過程の条件①［可能な解釈を豊富に同居させる表現力］を満足するのに貢献する．たとえば，「もも」の解釈として，「股」，「桃の実」，「桃の木」を考えるとき，

```
(*^(+ thigh peach_food peach_tree))
```

という表現によって３つの解釈の同居を図ればよい[赤間89b]．これもまた，［ある程度分ったもの］である．

集合束縛変数の第２の利点は，束縛集合をすばやく変化させる操作が可能なことである．たとえば *1^mammal を，「蛇または犬である」という情報によって *1^dog に変化させるためには，

```
(Crestrict *1^mammal (+ snake dog))
```

という呼出しを起動するか，あるいは，

```
(= *1^mammal *2^(+ snake dog))
```

というようにある「蛇または犬」（新しく生成した架空のものでもよい）と同一視すればよい．

「ももをもらう」の例では，

```
(Crestrict *1^(+ thigh peach_food peach_tree)
           (- object animal_body))
```

とすればよい．ここで thigh は animal_body のサブクラスであるとしている．

集合束縛変数の第３の利点は，集合束縛変数が複数個ある場合，それらの対象の同一性の表現が自然に達成され，集合束縛変数ｘのある出現を変化させるとその変化が複数個の場所のｘに瞬時に伝播すること，また backtrack すれば制約伝播の効果が高速に解除されることである．

たとえばＳ式の異なる場所に *x^dog が２度出現した場合，それらは同じ変数部分 *x を持つことによって同じ対象であることが表現されている．また，Ｓ式の異なる場所に出現した「２匹の」動物 *x^(+ bird mammal) と *y^(+ snake mammal) は，

```
(= *x *y)
```

というユニフィケーションによって同一視され，制約の伝播の結果「１匹の」哺乳類と判明する．*x と *yが同一視されても，それらが２つの場所に出現することには変りがない．しかし哺乳類 *x にさらに制約が加えられて，それが猫に変更されたとき，この情報は瞬時に別の場所の *y（*x と同一視されている*y）にも伝わる．このあと何かの不都合でbacktrack した場合，同一視は瞬時に解除され，同一視する前の状態に復帰する．このようなことは同一視される変数がいくら多くても，またそれらがいかに広範囲に分布していても同様に達成される．制約をかける*xは１つでよく，他に *x がどこに何個あるかをまったく知る必要がない．

これを使えば，解釈の整合性管理は非常に簡単にしかも高速に達成できる．単語の解釈を同じ変数で表現しておけば，その変数の束縛集合は常に同一に保たれるからである．指示語の場合のように，別のものが同じと認定される場合，それらを unify すれば同一の変数となるので，以後やはり同様の効果を得ることができる．

５．　PAL/TALK の理論的基礎

新しいユニフィケーションなどを導入することによって Prolog を拡張する研究は多い．たとえば，PAL では集合束縛変数が導入され，それに関して新たなユニフィケーションが必要である．しかしそれらのユニフィケーションはいかなる意味で正当なものなのだろうか．多くの場合，直観だけがその支えであり，正当性の理論的根拠は存在しなかった．GLP の理論は多くの「拡張 Prolog」に理論的な基礎を与えることができる．

Prolog を基礎付けるのが LP(logic program)の理論であるのに対して，GLPの理論は Prolog を含むかなり広い範囲の論理型言語の理論である．現在 Prolog は論理型言語の中心的な存在であるが，GLPの理論は Prolog が論理型言語の１つの例に過ぎず，有用な論理型プログラム言語としてどれを選ぶべきかは，今後の課題であることを明確にしている．PAL の例で言えば，GLPの理論は，「PAL と Prolog を対等のものとして扱うことのできる高い視点」を提供しており，Prolog を中心的／原初的なものとして特別扱いする必要はまったくない．PAL は Prolog を完全に包含している．

PAL は近似的には CLP＝制約論理型言語[Jaffer 87]とみなすこともできる．GLPの理論はそのCLPの理論を１分枝として含む理論体系である．

６．他の方法との比較

PAL/TALK の方法を他のいくつかの方法と比較して見よう．図７は自然言語処理のためのプログラム言語／知識表現系としてどのようなものを選択するかに関する選択の分岐である．a の選択点は，論理型言語を選ぶか否

かの分岐点である．ここで論理型言語を選べば，我々は少なくとも，自動 backtrackを使えるという重要な利点を享受できる．その自然言語処理における意味を考えて見よう．自然言語理解過程には，ある句がどの名詞句を修飾するものと想定するか，ある多義語の意味としてどれを選ぶか等，非常に多くの or 選択肢（＝仮説）がある．それらをうまく選べば，１つの正しい解釈が得られる．論理型言語で自然言語理解過程を記述すれば，選択を誤った場合のことは何も考えずに済む．単に可能な選択肢を書けば，言語自身が backtrack などを使って正しい選択を発見してくれるからである．

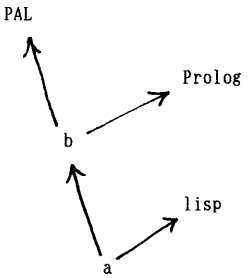

図７　自然言語処理用プログラム言語の選択

認識の逐次更新を達成するために知識表現系に要請される第１の点は，それが可能な解釈を同居させる表現力を持つことであった．この表現力は，知識表現系がどのような対象を直接に扱うかに関係しているので，その観点から lisp, prolog, PAL が扱うＳ式を比較して見よう．ただし項を扱うprolog とＳ式を扱うprolog は基本的には等価なので，ここでは簡単のためにprologはＳ式を扱うものとする．lisp の扱うＳ式では基礎となる対象は［定数＝完全に分ったもの］だけである．prolog の扱うＳ式ではそれに加えて［変数＝まったく分らないもの］が扱われる．さらに PALの扱うＳ式では，それらに加えて集合束縛変数が含まれる．それは［ある程度分ったもの］を直接的に表現することができる（図８）．

このように，直接表現できる対象は a と b の分岐で左右のいずれを選択するかによって大きく変化する．そして我々は PALの持つ集合束縛変数が可能な解釈を同居させるために貢献することをすでに見た．表現力の観点から見てPAL が有望なのは明らかである．

認識の逐次更新を達成するために知識表現系に要請された第２の点は，局所的な制約条件によって認識（解釈）を高速に変更できることである．この観点から b の分岐を検討する．認識の変更に関して Prolog は非常に不利である［赤間 88a］．Prolog は論理変数を扱うが，論理変数の値は backtrack せずには変更ができないからである．「動物」を animal で表すように，概念（それは認識の一部である）をアトムで表したら最後，それらは「死んだ」概念となってしまう．そのアトムが割当った論理変数は，いくら「それは犬だ」という新たな情報が得られても，更新することはできない．

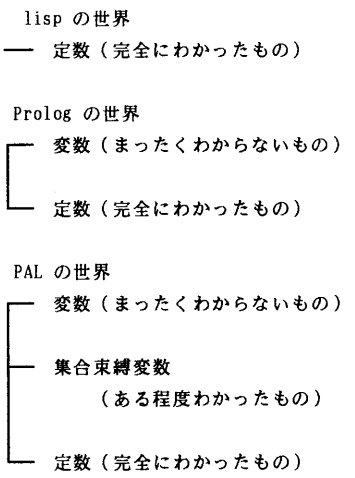

図８　lisp, Prolog, PAL の世界の比較

これを打開する１つの方法は，概念をアトムではなく，構造体にすることである．たとえば，［奥村 89］では，「動物」を，

```
(animal . *1)
```

なる構造体で表現している．これなら，「それは犬だ」という新たな情報が得られたら，

```
(animal . (dog . *2))
```

に変化させればよい．この変化は，

```
*1 = (dog . *2)
```

というユニフィケーションで実現できる．

［奥村 89］の方法に関していくつかのことを指摘しておこう．第１に［奥村 89］では PAL のユニフィケーションをサブセットとして含むと主張しているが，これは誤りである．実際にはその逆で，彼らのユニフィケーションが PAL のサブセットである．第２に，その方法は結局 PAL/TALK のサブセットの Prolog による実装方法の１つに他ならない．それは簡単に言えば，

```
(animal . (dog . *2))
```

のような構造体を導入して，それに新しいインタープリタを適用することによって，

*^dog

なる集合束縛変数を模擬するものである．第３に，著者自身も「新しい制約論理型言語」と説明しているように，そこで得られたシステムはもはや Prolog ではない．新しいユニフィケーションをするために新しいインタープリタが準備されているからである．Prologで実装していることによって Prolog の枠内で逐次更新過程が達成されたかのような錯覚に陥いることのないように注意したい．第４に，ある言語，特に Prolog のような高級言語で別の言語をインプリメントした場合，得られたシステムの時間的，空間的効率は一般にかなり悪くなる．奥村の実装方法では特に空間的効率が悪くなると予想される．第５に，PAL/TALK の方法を Prolog で模擬しようとするよりも，GLPの理論を援用して直接 PAL/TALK を理解するほうが，理論的基礎が厳密であり，本質を明快に議論でき，今後の拡張にも指針が得られると考えられる．

文　献

[Jaffer 87] Jaffer,J. and Lassez,J.L.: "Constraint Logic Programming", Proc. 14th ACM POPL Conf., (1987)

[Lloyd 84] Lloyd,J.W.: Foundations of Logic Programming, Springer-Verlag, p.124 (1984) 邦訳「論理プログラミングの基礎」佐藤，森下訳

[Mellish 84] Mellish,C.S.: Computer Interpretation of Natural Language Descriptions, Ellis Horwood Limited (1984), 邦訳「コンピュータのための自然言語意味理解の基礎」田中穂積訳 (1987)

[Mukai 85] Mukai,K. and Yasukawa,H. : Complex Indeterminates in Prolog and its Application to Discourse Models, New generation computing, 3 pp.441-466 (1985)

[Sowa 84] Sowa,J.F. : Conceptual Structures, Addison Wesley P.C., p.481 (1984)

[赤間 87a] 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)

[赤間 87b] 赤間清：継承階層 Prolog の高速化機構，人工知能学会誌，Vol.2, No.4, pp.492-500 (1987)

[赤間 87c] 赤間清：集合束縛変数に基づく意味表現とユニフィケーション，情報処理学会，自然言語処理研究会資料，62-8, pp.53-60 (1987)

[赤間 87d] 赤間清：Multiple class bound variables, 日本ソフトウェア科学会第４回大会論文集，D-2-1,pp.207-210 (1987)

[赤間 87e] 赤間清：自然言語の意味処理とその高速化，日本ソフトウェア科学会第４回大会論文集，B-4-4,pp.359-362 (1987)

[赤間 88a] 赤間清：Blackboard in Prolog，情報処理学会，知識工学と人工知能研究会資料，57-6, pp.43-50 (1988)

[赤間 88b] 赤間清：認識の逐次更新過程としての意味処理，電子情報通信学会技術研究報告，NLC87-28, pp.33-40 (1988)

[赤間 88c] 赤間清：拡張されたロジック・プログラムの理論Ｉ，人工知能学会研究会資料，SIG-FAI-8803-3, pp.21-30 (1988)

[赤間 89a] 赤間清：認識と知識の逐次更新と知識表現系，博士論文，p.199 (1989)

[赤間 89b] 赤間清：集合束縛変数とその自然言語処理への応用，人工知能学会誌，Vol.4, No.2, pp.177-184 (1989)

[赤間 89c] 赤間清：ＧＬＰの理論Ｉ，ＷＯＬ'８９論文集 (1989)

[赤間 89d] 赤間清：ＧＬＰの理論Ⅱ，情報処理学会，知識工学と人工知能研究会資料，63-9,pp.77-85 (1989)

[赤間 89e] 赤間清：ＧＬＰの理論Ⅲ，情報処理学会，知識工学と人工知能研究会資料，63-10,pp.87-96 (1989)

[赤間 89f] 赤間清：ＧＬＰの理論Ⅳ，情報処理学会，知識工学と人工知能研究会資料，64-4, pp.31-40 (1989)

[赤間 89g] 赤間清：ＧＬＰの理論Ⅴ，人工知能学会研究会資料，SIG-FAI, (1989)

[井佐原 86] 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)

[奥村 89] 奥村学，田中穂積：制約蓄積による増進的曖昧性解消モデル，情報処理学会，自然言語処理研究会，71-1, pp.1-8 (1989)

[大澤 86] 大澤一郎：オブジェクト指向概念を拡張した知識表現言語ＫＲＳとその応用，コンピュータ・ソフトウエア，Vol.3, No.4, pp.21-27 (1986)

[高木 87] 高木朗，伊東幸宏：自然言語の処理，丸善 p.200 (1987)

[滝川 88] 滝川雅巳：宣言型知識表現言語の構築，北海道大学修士論文，p.60 (1988)

[橋田 88] 橋田浩一：ＡＩとは何でないか－情報の部分性について－，Bit, Vol.20, No.8 (1988)

[丸山 88] 丸山宏，渡辺日出男：制約伝播アルゴリズムを用いた対話的日本語解析システム，日本ソフトウェア科学会第５回大会論文集，pp.17-20 (1988)

[武藤 88] 武藤幸好，辻井潤一，長尾真：ＫＧＷ＋ｐにおける解析の効率化と優先度の計算，電子情報通信学会技術研究報告，NLC87-26, pp.17-24 (1988)