自然言語処理 93−12
(1993. 1. 14)

# プログラム変換による意味の解釈

野村 祐士　赤間 清　宮本 衛市

北海道大学　工学部　情報工学科

本論文では、プログラム変換を用いた意味の解釈について述べる。文の意味、対象世界の知識、状況は節集合で表現でき、これらの和集合を論理プログラムと考える。このプログラムを、プログラム変換により、新たな意味表現に変換することで意味解釈を行う。また、一般的な unfold 変換以外に、意味候補を扱う変換を新たに考える。これらの変換を一般化論理プログラムの理論に基づいて行うことにより、理論的に明快で、拡張性のある意味処理が可能になることを示す。

# Semantic Interpretation by Program Transformation

Yuji Nomura　Kiyoshi Akama　Eiichi Miyamoto

Depertment of Information Engineering, Faculty of Engineering
Hokkaido University

This paper presents a method for semantic interpretation by program transformation. Meanings of sentences, knowledge and situations can be represented as a set of clauses, that is a logic program. This program can be transformed to another expression by program transformation. We introduce some operators for this transformation. Using these operators based on theory of generalized logic programs, we aim for a theoretical and expansible semantic interpretation.

## 1 はじめに

一般に、意味処理（意味解釈）とは、意味モデルの意味表現を、何らかの手続きで変換する処理である。従来の意味処理では、意味のモデルでは宣言的な記述を扱っているが、その表現変換には多くの複雑な処理手続き記述していた。このため、意味処理自体に、一貫した理論があるとは言い難く、このことは、高度な意味の解釈を困難にしている原因の一つにもなっている。

本論文では、一般化論理プログラム(GLP)の理論[2]を意味解釈の基礎とする。その上で、論理式による意味記述を、プログラム変換という方法で、一貫して扱う手法を提案する。

まず、GLPの枠組では、単一化が拡張されるため、論理オブジェクトの表現力が高く、意味とその対象となる世界の知識、状況を簡潔な論理プログラムで記述できることを示す。その結果、プログラムの宣言性が維持され、表現の変換に複雑な手続きが必要なくなるため、プログラム変換という明快な手続きによって、表現変換が可能になることを示す。加えて、GLPのプログラム変換の操作（オペレータ）、制御、意味解釈に関する理論的な背景についても述べる。

なお、本論文の意味解釈は、GLPに基づく論理型言語UL/$\alpha$によって、意味解釈システムとして実現されている。

## 2 本論文で扱う意味の解釈の基本的枠組

この章では、意味解釈の本論文での定義と、その特徴について述べる。

### 2.1 意味解釈の定義

自然言語文の意味を大別すると、文面だけで表現している意味と、文面に現れない知識や状況を加味した解釈の2種類に分けることができる。

例えば図1のような将棋の盤面をもとに、次の文が与えられたとする。

「先手の歩を4－5に動かせ」

この文は、その文面だけを考えると、

1.「先手の歩」という物（駒）を「4－5」という場所に「動かせ」

ということを意味している。さらに、将棋の知識と盤面の状況を考慮して、この文の解釈すると、

2. 文中の「先手の歩」という駒は、盤面上の「4－6」の駒である。つまり、意味は、「先手の4－6の歩」を「4－5」に動かせ

である、といえる。

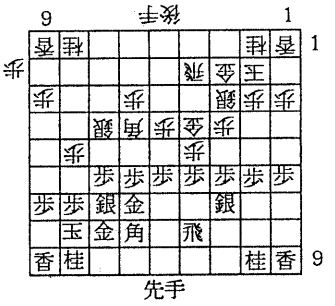


図1 与えられた将棋の盤面

ここでは、1のような文面だけの意味を浅い意味、2の知識と状況を考慮した解釈を深い意味と呼ぶことにする。本論文では、文の浅い意味から深い意味を作る意味解釈を扱う。

### 2.2 文の意味、知識、状況の統合

ここでは、手続き的な意味処理の考え方と、本論文における意味解釈との違いについて述べる。

先程の例文の浅い意味で、動かす対象として指定された駒には、「先手の歩」という情報しか与えられていない。しかし、「先手の歩」は盤面上に複数実在していて、「先手の歩」が先手のどの歩を意味するのかは、判明しない。このため、対象世界の知識と状況を用いて、「先手の歩」の意味を明確にする、つまり浅い意味から深い意味を導出する処理が必要となる。

これらを手続き的に処理すると、例えば、

- 浅い意味のうち、「先手の歩」の意味が明確でない、と判定し、
- 将棋の駒の動きの知識から、「先手の歩」の移動先が4－5であるときの、元の位置を計算し、
- それが盤面の状況、つまり盤面上に「先手の歩」に相当する駒が存在することを確認する。
- 最後に浅い意味表現に情報を付加する

といった手続きを書く必要がある（図2）。このような手法では、意味表現が複雑化すればそれだけ、処理手続きの記述にかかる労力も莫大になる。さらに、このように記述された手続きは、ある対象領域に依存した記述であり、拡張性に乏しい。

このような意味処理の欠点を克服するため、本論文では、プログラム変換という、統一的な方法で意味を扱う

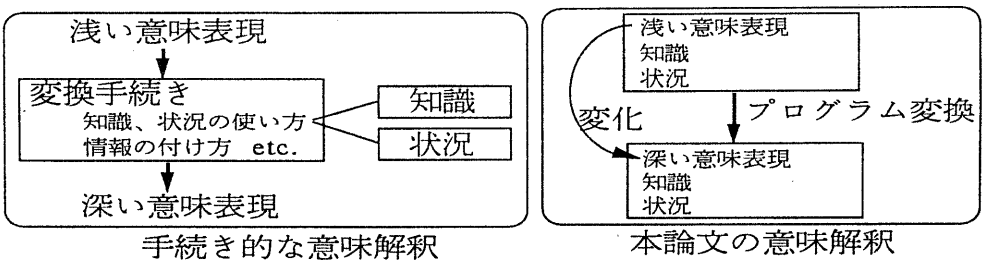


図2 手続き的な意味処理と本論文での意味解釈

ことを提案する。ここでは、文の意味、知識、状況をそれぞれ、独立したものとは考えず、3つを合わせて一つの論理プログラムとして捉えることにする。そして、プログラム全体が、プログラム変換という汎用の操作によって、違う表現のプログラムに変換される (図2)。

また、変換前後のプログラムは意味が等しいという関係が成り立つ。つまり文の意味、知識、状況をまとめたプログラムは、等価に変換できることになる。このことから、本論文では意味解釈は、

深い意味「先手の4－6の歩を4－5に動かせ」
＋盤面の状況 ＋将棋の知識
‖
浅い意味「先手の歩を4－5に動かせ」
＋盤面の状況 ＋将棋の知識

という、等価な関係が成立すると考える。

## 3 プログラム変換による意味解釈

文の浅い意味、知識、状況はそれぞれ節集合で表現できる。この3つの節の和集合をプログラムとみなすと、プログラム変換を用いて文の意味解釈が行える。プログラム変換とは、与えられたプログラムを別の表現のプログラムに変換することである。

この章では、まず、プログラム変換で意味解釈を行うための準備として、プログラム変換を、オペレータ、アルゴリズム、制御という3つの点で述べる。次に、それらを用いた、プログラム変換による意味解釈の概要を述べる。

### 3.1 プログラム変換オペレータ

プログラムを変換するオペレータとして、本論文中では次の3つを用いている。

1. [unfold(変換)] オペレータ
   変換する節の *body* アトム（原子論理式）を、これとユニファイ可能な *Head* を持つ節の *body* で置き換える。unfold 変換は、置換えが可能なすべての述語に作用する。
   プログラム $P_1$ から $P_2$ への unfold 変換は GLP の枠組では、以下の手順で行われる。
   (a) $P_1 = \{C_j \mid j \in J\}$ から節
   $C_k = (H \leftarrow A_1, \cdots, A_i, \cdots, A_q)$ を選ぶ。
   (b) 節 $C_k$の *body* から $A_i$を選ぶ。
   (c) プログラム $P_1$の各節 $C_j(j \in J)$ に対して $C_k$の アトム $A_i$と $C_j$の *Head* の *unifier* 集合の任意の一つを $U_j(j \in J)$ とする。
   (d) $U_j(j \in J)$ を用いて、
   $$R_j = \{(H\theta \leftarrow A_1\theta, \cdots, A_{i-1}\theta, B_1\sigma, \cdots, B_b\sigma, A_{i+1}\theta, \cdots, A_a\theta) \mid (\theta, \sigma) \in U_j\}$$
   $R = \cup_{j \in J} R_j$
   $P_2 = P_1 - \{C_k\} \cup R$
   として $P_2$を得る。
2. [候補の絞り込み] オペレータ
   述語 MEMBER の第一引数により、第二引数のオブジェクトリストを制限する。詳細は5.4節で述べる。
3. [MEMBER の確定] オペレータ
   述語 MEMBER の第二引数が一個のオブジェクトから成るリストのとき、第一引数を確定する。詳細は5.7節で述べる

### 3.2 プログラム変換の制御

プログラム変換では、変換される節と、変換に用いる節を節集合から選択する必要がある (前節の手続きの (a) と (b))。

本節では、節選択の方法を適切に決めることにより、プログラム変換を制御する方法について述べる。

本論文では、変換される節(a)を、文の意味を表す節に限定する。つまり、プログラム変換によって、文の意味を表す節だけを順次変化させる。プログラム変換は、プログラム変換がそれ以上行えない場合、あるいはプログラム変換を継続するのに不適当な場合（プログラムのサイズが大きくなり過ぎた場合など）に終了する。また、文の意味節をbodyのない節(fact)にすることを目標に、変換を進めてゆく。

次に、変換に用いる節(b)の選択法であるが、例えば、プログラム変換で、変換される節(a)が

```
C:  H(X,Y) ← member(X,[a,b]),B(X,Y)
```

であるとする。この時、Cに作用する節集合として、以下の3つの節が考えられるとする。

```
member(X,[X|S]) ←
member(X,[_|S]) ← member(X,S)
B(b,c) ←
```

もし、節Cの第1 bodyアトム、member(X,[a,b])を1回unfold変換すると、

```
C1: H(a,Y) ← B(a,Y)
C2: H(X,Y) ← member(X,[b]),B(X,Y)
```

という2つの節が生じる。この後、変換を続けると、C1は述語Bでは変換できず、消滅する。C2はさらにunfold変換ができ、以下の節を得る。

```
H(b,c) ←
```

次に、節Cの第2 bodyアトムB(b,c)を変換する場合を考えると、Cは、

```
H(b,c) ← member(b,[a,b])
```

になり、さらに、

```
H(b,c) ←
```

となるので、節は1つのままである。前の変換に比べ後の変換は、無駄な変換がなく、効率が良い。

そこで、このように複数の節が生じる変換はなるべく行わないことにする。つまり、複数の節定義のからなる述語（ここではmember）よりも、1つの節定義の述語(B)の変換を先に試みる。

自然言語の解釈においては、可能性のある解釈の数だけ意味表現の節を増加させると、組合せ爆発が起きるので、このような制御が必要となる。

また、引数に必要な情報が不足するなどして、変換が行えないbodyアトムが存在する場合もある。このようなときも、先程のように変換を保留して、その他の述語の変換から、必要な情報が伝播されるのを待つ制御を行なう。

以上をまとめると、本論文では、節内の変換可能なbodyアトムに対して、なるべく節が増加しないようにunfold変換を適用する、という制御を行う。

### 3.3 プログラム変換による意味解釈

文の意味、知識、状況の節集合$P_s, P_k, P_c$で構成されるプログラム

$$P = P_s \cup P_k \cup P_c$$

を考え、その宣言的意味をここでは

$$\mathcal{M}(P)$$

と表すことにする。

プログラム$P_1$を$P_2$にunfold変換する時、次のことが知られている。

**定義**　Unfold変換が健全かつ完全

$$\Leftrightarrow \mathcal{M}(P_1) = \mathcal{M}(P_2)$$

与えられたプログラムは、1~3のオペレータを繰り返し作用させて、別の表現のプログラムに変換される。変換毎に意味の等価性が保証されるので、与えられた節集合の意味はすべての変換を通じて等価である。よって、1~3のオペレータにより、プログラム$P_1 = P_{s1} \cup P_k \cup P_c$を$P_2 = P_{s2} \cup P_k \cup P_c$に変換した際も、

$$\mathcal{M}(P_1) = \mathcal{M}(P_2)$$

という、意味の等価性がある。

以上で説明したプログラム変換による意味解釈によって、文の意味、知識、状況をまとめて考えると、次のような変換が起きる。

浅い意味「先手の歩を4－5に動かせ」
＋盤面の状況 ＋将棋の知識
↓複数回の変換
深い意味「先手の4－6の歩を4－5に動かせ」
＋盤面の状況 ＋将棋の知識

これにより、浅い意味から深い意味への変換が達成される。

## 4 文の浅い意味、知識、状況の節表現

この章では、文の浅い意味、知識、状況を、論理オブジェクトを用いた節形式で、どのように表現するかについて述べる。

### 4.1 論理オブジェクト

節表現で扱う、論理オブジェクト（以下、オブジェクトと呼ぶ）を本論文では、<変数:属性/属性値,...>と表記することにする。このオブジェクトは、いくつかの属性と属性値のペアが、情報としてついている変数を意味する。

例えば、先程の例文における動かす対象は「先手の歩」である。これをオブジェクトで表現すると、以下のよう

になる。

<p1:kind/歩,player/先手>

同様に、「後手の4－5の歩」は、

<p2:kind/歩,player/先手, 横/4, 縦/5>

のように表現する。

オブジェクトの情報（駒であれば、指し手、位置、種別の情報）は、意味解釈過程で刻々と変化する。これらのオブジェクトの変化は、単一化によって定義する*。例えば、将棋の駒オブジェクトは単一化により、

<x:kind/歩> <y:player/先手> $\stackrel{unify}{\Longrightarrow}$

<z:kind/歩,player/先手>

と、変化できるように定義してある。

単一化させる2つのオブジェクトが、同じ属性を持っていても、その属性値が単一化できなければ、失敗することになる。例えば、

<x:kind/歩> <y:kind/王>

$\Longrightarrow$ 単一化失敗

となる。これは、「歩」と「王」は、決して同じ駒には解釈されないことを意味している。

また、一つの節内で変数が同じオブジェクトは、同一のオブジェクトとみなす。つまり、先ほどのような単一化が起きた場合、節内にあるすべての
<x:kind/歩>と <y:player/先手>が
<z:kind/歩,player/先手>
に変化する。

このようなオブジェクトを用いることにより、prologのオブジェクトに比べ、複雑な情報の伝播[7]を簡潔に実現できる。これらの様々なオブジェクトの単一化は、GLPの理論的基礎がある[2]。

### 4.2 文の浅い意味の節表現

本節以降では、節表現の方法を例示する。

先程の例文「先手の歩を4－5に動かせ」の浅い意味は、以下のような節で表現する。ただし、'*' が付いた*場所 などは変数を表す。

H1 (動かす <x:kind/歩> *場所) ←
B1 (動かせる <x:kind/歩> *場所)
B2 (player <x:kind/歩> 先手)
B3 (place *場所 (横/4 縦/5))

節内の各述語は、以下のことを述べている。

H1 (種類が) 歩の駒を *場所 に「動かす」
B2 歩の駒を *場所 に動かすことは、
将棋のルールに適合する
B2 歩の駒は先手の駒である
B3 *場所 は将棋盤上の座標「4－5」である

このような文の浅い意味は、将棋の知識などを利用することなく、構文解析だけで得ることができる。

### 4.3 知識の節表現

4.2節の例に現れたB1の「動かせる」という述語は、将棋の知識として以下のように節表現可能である。

a.(動かせる *obj *place) ←
b. (実在する *obj)
c. (盤面上 *place)
d. (動きのルール *obj *place)
e. (通り道に駒がない *obj *place)
f. (自分の駒がない *place)

節内の各述語は、以下のことを述べている。

a. *obj を *place に動かす、とは
b. *obj が盤面上に実在し、
c. 移動先 *place は盤面上の座標であり、
d. *obj を *place に動かせる将棋のルールがあり、
e. *obj を *place に動かす間に、将棋の駒がなく、
f. 移動先の *place に自分の駒がない

知識節のbodyであるb～fに現れる各述語も、それらを含むアトムを *Head* として持つ節集合によって記述する。例えば、bの「実在する」は、以下のような知識として記述する。

b.(実在する *obj) ←
g. (状況 *盤面)
h. (MEMBER *obj *盤面)

節内の各述語は以下のことを述べている。

b. *obj が実在するとは、
g. *盤面 に盤面の全ての駒が状況として与えられ、
h. *obj が *盤面 の駒のうち、どれか一つである。

### 4.4 状況の節表現

前節のgの「状況」という述語は以下のようなリストを、第一引数とするfact節で記述される。ただし、このリストは、盤面のすべての駒（ここでは、39個の駒）オブジェクトを各要素としている。

i.(状況 (
<p1:kind/歩,player/先手, 横/1, 縦/6>
<p2:kind/香,player/先手, 横/1, 縦/9>
:
<p38:kind/歩,player/後手, 横/9, 縦/3>
<p39:kind/香,player/後手, 横/9, 縦/1>)

* 単一化は、UL/$\alpha$の述語 defUnify で定義する。

)←

このように、プログラム変換による意味解釈では、状況は知識と同等の節で扱える。

## 5 意味解釈例

本章では、「先手の歩を 4-5 に動かせ」という浅い意味が、深い意味に変換される過程を例示し、その概略を述べる。

### 5.1 変換例

例文の浅い意味表現の節は、おおよそ以下のように変換される。

```
S1 (動かす <a:kind/歩> *場所) ←
    (動かせる <a:kind/歩> *場所)
    (player <a:kind/歩> 先手)
    (place *場所 (4 5))
      ↓「player」 と「place」 を [unfold]
S2 (動かす <b:kind/歩,player/先手>
          <x:横/4, 縦/5>) ←
    (動かせる <b:kind/歩,player/先手>
              <x:横/4, 縦/5>)
        ↓「動かせる」 を [unfold]
S3 (動かす <b:kind/歩,player/先手>
          <x:横/4, 縦/5>) ←
    (実在する <b:kind/歩,player/先手>)
    (盤面上 <x:横/4, 縦/5>)
    (動きのルール <b:kind/歩,player/先手>
                  <x:横/4, 縦/5>)
    (通り道に駒がない
     <b:kind/歩,player/先手>
     <x:横/4, 縦/5>)
    (自分の駒がない <x:横/4, 縦/5>)
      ↓「実在する」と「状況」 を [unfold]
S4 (動かす <b:kind/歩,player/先手>
          <x:横/4, 縦/5>) ←
    (MEMBER <b:kind/歩,player/先手>
            [盤面上のすべての駒のリスト])
                :
    (自分の駒がない <x:横/4, 縦/5>)
```

ただし、

```
[盤面上のすべての駒のリスト] =
 (<p1:kind/歩,player/先手, 横/1, 縦/6>
  <p2:kind/香,player/先手, 横/1, 縦/9>
                :
  <p38:kind/歩,player/後手, 横/9, 縦/3>
  <p39:kind/香,player/後手, 横/9, 縦/1>)
      ↓ [候補の絞り込み]
S5 (動かす <b:kind/歩,player/先手>
          <x:横/4, 縦/5>) ←
    (MEMBER <b:kind/歩,player/先手>
            [9 つの先手の歩のリスト])
                :
    (自分の駒がない <x:横/4, 縦/5>)
      ↓数回の [unfold],[候補の絞り込み]
S6 (動かす <c:kind/歩,player/先手, 横/4>
          <x:横/4, 縦/5>) ←
    (MEMBER <c:kind/歩,player/先手, 横/4>
            (<d:kind/歩,player/先手
                              , 横/4, 縦/6>)
                :
    (自分の駒がない <x:横/4, 縦/5>)
              ↓ [MEMBER の消去]
S7 (動かす
    <d:kind/歩,player/先手, 横/4, 縦/6>
    <x:横/4, 縦/5>) ←
    (通り道に駒がない
     <d:kind/歩,player/先手, 横/4, 縦/6>
     <x:横/4, 縦/5>)
    (自分の駒がない <x:横/4, 縦/5>)
        ↓ body の 2 つを [unfold]
S8  (動かす
     <d:kind/歩,player/先手, 横/4, 縦/6>
     <x:横/4, 縦/5>) ←
```

### 5.2 各変換の概要

S1 から S2 への変換

最初に与えられた文の意味節 S1 は

```
(player <X:player/*player> *player) ←
  <X:..>という駒は *player の駒である
(place <Y:横/*横, 縦/*縦> (*横 *縦)) ←
  <Y:..>という駒の縦座標、横座標は
  それぞれ *横 *縦である
```

の 2 つの知識節で順次 [unfold] が適用され、S2 となる。この時、オブジェクトは以下のように変化し、変化は各オブジェクトに伝播する。

$$\texttt{<a:kind/歩>} \underset{unfold}{\Longrightarrow} \texttt{<b:kind/歩,player/先手>}$$

$$\texttt{*場所} \underset{unfold}{\Longrightarrow} \texttt{<x:横/4, 縦/5>}$$

S2 から S3 への変換

4.3節の a ～ f の知識の *Head* a と S2 の *body* が単一化して、S2 の *body* が b ～ f に置き換えられる。

S3 から S8 までの変換

さらに b は g,h に、その g は i に置き換えられ、S4 となる。以下の変換は、5.4節以降で詳しく触れるが、2回の [MEMBER の制限] と [MEMBER の確定]、及び数回の [unfold] により最終的には S8 が得られる。

この時もオブジェクトは以下のように変化する。

<b:kind/歩,player/先手> $\Longrightarrow_{unfold}$
<d:kind/歩,player/先手, 横/4, 縦/6>

### 5.3 意味候補の表現

4.3節の実在するという知識 b,g,h には MEMBER という述語を使っている。この MEMBER は、以下のような形をしている。

(MEMBER <オブジェクト> [候補])

これは、<オブジェクト>が、[候補] というリストの要素うちのどれか一つである、ということを意味する。また、MEMBER は 5.7節で述べる [MEMBER の確定] が適用されない限り、消去されない。

例えば、5.1節の変換例では、S3 から S4 への変換時に、述語「実在する」は、以下のように変換された。

(MEMBER <kind/歩,player/先手>
[盤面上のすべての駒のリスト])

これは、「先手の歩」は、盤面上のすべての駒のうちのどれか一つであることを意味している。

### 5.4 候補の絞り込みオペレータ

S4 において、はじめて述語 MEMBER が出現する。これにより、従来の [unfold] 変換の他に、[候補の絞り込み] 変換が適用可能になる。[候補の絞り込み] は、

(MEMBER <オブジェクト> [候補])

の [候補] で表現された意味候補を、<オブジェクト>と単一化可能な候補だけに絞り込むオペレータである。このオペレータにより、例えば S4 の MEMBER の候補のうち、

<p1:kind/歩,player/先手, 横/1, 縦/6>

は<b:kind/歩,player/先手>と単一化可能で意味候補になりうるが、

<p2:kind/香,player/先手, 横/1, 縦/9>

は単一化できないので、意味候補から外される。このような絞り込みの結果、MEMBER は以下のように変化する。

(MEMBER <b:kind/歩,player/先手>
[盤面上のすべての駒のリスト])
↓
(MEMBER <b:kind/歩,player/先手>
[9つの先手の歩のリスト])

よって S4 は S5 に変換される。この変換は、「先手の歩」という駒は、盤面上の全ての駒のうち、「9つの先手の歩」のいずれかである、と表現し直す意味を持つ。

### 5.5 意味候補の保持

5.4節では、「先手の歩」の候補は絞られたものの、依然として「先手の歩」が盤面上のどの駒かは、判明していない。3.2節で述べたように、この場合、他の述語の変換を試みる。

そこで、S5 内の述語「動きのルール」を [unfold] してゆくと、歩の移動先の横座標が 4 であれば、歩の横座標も 4 になることが自然に出てくる。この結果、オブジェクトは

<b:kind/歩,player/先手> $\Longrightarrow_{unfold}$
<c:kind/歩,player/先手, 横/4>

と変化する。

このように、他述語が変換されている間、述語 MEMBER は「先手の歩」の意味候補を保持する役割も担っている。

### 5.6 意味候補の変化

前節のように、オブジェクトが変化したことで、再び [候補の絞り込み] オペレータで、以下のように候補が絞られる。

(MEMBER <c:kind/歩,player/先手, 横/4>
[9つの先手の歩のリスト])
↓
(MEMBER <c:kind/歩,player/先手, 横/4>
(<p7:kind/歩,player/先手, 横/4, 縦/6>))

つまり、「横座標が 4 の先手の歩」は「9つの先手の歩」のうち、「先手の 4-6 の歩」ただ一つであることがわかる。

以上の 5.5節からの一連の変換によって、5.1節の S5 から S6 への変換が行なわれる。そして、保持された意味候補は不変のものではなく、オブジェクトの変化に追従して変化する。

### 5.7 意味の確定

S6 では、保持された意味候補は一つしか存在してない。このような場合には、[MEMBER の確定] が適用される。これは、一つだけの意味候補を MEMBER の第一引数の<オブジェクト>とユニファイさせ、MEMBER を取り除くオペレータである。

これにより、S6 の MEMBER は、

(MEMBER <c:kind/歩,player/先手, 横/4>
(<p7:kind/歩,player/先手,

```
                                横/4,縦/6>))
                                ↓
(MEMBER <d:kind/歩,player/先手,
                                横/4,縦/6>
        (<d:kind/歩,player/先手,
                                横/4,縦/6>))
                                ↓消去
```

と変化する。これは、意味候補がただ一つの時、候補を意味として確定し、かつ消去できることを意味する。

こうしてS6がS7に変換され、さらに、S7の節の*body*は[unfold]によりすべて消去され、最後にS8が与えられる。

## 6 おわりに

本論文で提案したGLPのプログラム変換による意味解釈は、以下の特徴を持つ。

- 意味の解釈を、プログラム変換という理論的基礎のある、明快で汎用のアルゴリズムで行える。
- 背景知識や状況などをモジュラー性高く記述できる。
- 高級な論理オブジェクトを用いれば、情報の伝播、付加は節の記述と単一化の定義だけで容易に実現できる。つまり、意味解釈にあたっては、情報伝播や置き換えを明示する必要がない。
- 浅い意味の情報の不足箇所を調べる必要がない。また、情報を手続き的に付加して深い意味を導出する必要もない。
- 知識や状況を記述するだけで、さまざまな表現の文の意味を解釈できる。また、対象世界を変更しても対応できる。

プログラム変換による自然言語処理を扱った研究としては、

- 伝ら[6]の統語解析と統語生成を、プログラム変換を用いて統合する試み、
- Tudaら[5]のcu-prologのプログラム変換による、JPSGベースの構文解析、
- 橋田[8]の依存伝播を計算する手法としてのプログラム変換の試み、

などがある。本論文との主な相違点は、以下の点である。

- これらの研究は主に統語解析や統語生成を扱っているが、本論文では意味処理を中心課題としている。
- これらの研究の理論的基礎はCLPであるが、我々はGLPに基づいている。

CLPとGLPに関して比較を行うと、

- GLPの理論はCLPの理論を包含する。つまり、GLPの方がより一般的である[1]。
- CLPのプログラム変換はGLPのプログラム変換を用いることにより、より簡単で、より適切に行える。これは、制約伝播が自動的に達成されること、などの理由による。[4]。
- GLPのオブジェクトは表現力があり、素性構造など自然言語処理に特にに有用な構造をより明快に捉えられる。また、typeを付加するなどの拡張にも向いている[3]。
- GLPでは問題に適したオブジェクトを用いることが容易である。適切なオブジェクトはプログラム変換の計算量が減少させ、変換効率を良くする[9]。
- 統語処理に比べ、意味処理では対象世界の記述に高度なオブジェクトが必要となる。GLPでは新しいオブジェクトを容易に導入できる。

などとなる。

今後の課題としては、

- プログラム変換のより適切な制御
- より複雑な意味の解釈

などの達成が挙げられる。

### 参考文献


[1] Akama,K.:A New Theoretical Foundation of Constraint Logic Programs, *Hokkaido University Information Engineering Technical Report*, HIER-LI-9220,1992.

[2] Akama,K.:Unfolding of Generalized Logic Programs, *Preprints Work. Gr. for Artificial Intelligece*, IPSJ 83-3-AI,pp.43-52,1992

[3] 赤間 清:タイプつきユニフィケーション文法の新しい基礎, 情報処理学会, 自然言語処理研究会, 本号,1992.

[4] 大松正樹, 赤間清, 宮本衛市:制約論理プログラムの部分計算, 日本ソフトウェア科学会, 合成変換研究会資料,1992.

[5] H.Tuda,K.Hashida,H.Sirai:JPSG Parser on Constraint Logic Programming, Proceedings of the European Chapter of ACL'89,1989,pp.95-102.

[6] 伝康晴, 松本裕治, 長尾真:認知的制約プログラミングと統合的自然言語処理— 統語解析と統語生成の統合について, コンピュータソフトウェア, Vol.8,No.6,pp.38-50,1991.

[7] 橋田浩一:自然言語の構文解析と文生成の統合, 情報処理,Vol.33,No.7,1992.

[8] 橋田浩一:制約と言語, コンピュータソフトウェア, Vol.6,No.4,pp.16-29.,1989.

[9] 馬淵浩司 赤間清 宮本衛市:一般化論理プログラムのunfoldingによる変換ルールの合成, 情報処理学会, 人工知能研究会資料,84-8,1992.