自然言語処理 34－5
（1982. 12. 8）

双方向性階層的関数型言語Bi-HFPによる自然言語処理の記述について

田村直良　片山卓也
（東京工業大学情報工学科）

1．はじめに

最近、機械翻訳をはじめとする人工知能の種々の分野で自然言語解析の重要性は増しており、その解析手法についても拡張LINGOL[1] など多くが提案、実現されている。

さて、階層的関数型言語HFP[2]は、属性文法に基礎を置き、プログラムを階層的にモジュール分割することにより、より具体的なレベルへ詳細化していく仕様記述法である。HFP の一つの記述単位（分割記述）は、モジュールの分割を表わす分割規則、各モジュールの入出力属性間の関係を等式により静的に表わす意味規則、その分割記述を採用してよいかどうかを示す分割条件の３つにより構成される。HFP プログラムは、最上位モジュールの入力属性に値を与えることによって起動され、このモジュールの出力属性として結果を得る。

このHFP を用いて、cfg の非終端記号をモジュールに、生成規則を分割規則に対応させることにより容易にrecursive descent なtop-down parser が記述できる。しかし、自然言語処理などに対してより柔軟性のあるbottom up parserは直構文的には記述できない。

ここで述べるBi-HFPは、HFP を双方向性に拡張したものである。つまり、子モジュールの属性値がある条件（ 結合条件 ）を満たした時に、それらを一つの親モジュールに統合するという機能をHFP に付加したものである。

本稿では、Bi-HFPを定義し、実現可能なサブクラスにおいて　簡単な日本語文法による文の処理過程を記述した。

2．Bi-HFPについて

2．1　Bi-HFPの定義

Bi-HFPの定義を示す。

＜定義２．１＞　（Bi-HFPの形式的定義）

Bi-HFPは次の７タップルとして定義される。

$Bi\text{-}HFP = \langle \mathbb{M}, \mathbb{M}_{init}, Top, A, \mathbb{D}, V, E \rangle$

ここで、

(1)　$\mathbb{M}$はモジュールの集合であり、分割／統合を終了させる特別なnullモジュールを含む。

(2)　$\mathbb{M}_{init} \subseteq \mathbb{M}$ は　初期モジュールの集合である。

(3)　$Top \in \mathbb{M}$ は　最上位モジュールである。

(4)　Aは　モジュールの入出力属性集合。モジュール$M \in \mathbb{M}$の入力属性集合と出力属性集合を各々IN[M], OUT[M] と定義するとき、

$$A[M] = IN[M] \cup OUT[M]$$

かつ、 $IN[M] \cap OUT[M] = \phi$ とする。

(5)　$\mathbb{D} \subset \mathbb{M} \times \mathbb{M}^*$ はモジュール分割／統合の集合であり、$d \in \mathbb{D}$は分割／統合（または分割／統合規則）と呼ばれ、次のように表現される。

$d : M_0 \rightarrow M_1 \cdots M_n$ when C_d , $n \geq 0$

ここで、$M_0, M_1, \cdots, M_n \in \mathbb{M}$

であり、

特に$n = 0$の場合 $d : M_0$ when C_d を終端分割規則と呼ぶ。

C_d を結合条件と呼び、これが真の時モジュールM_0 が$M_1, M_2, \cdots, M_n$ に分割、またはモジュール$M_1, \cdots, M_n$ がM_0 に統合されるという。

$a \in A[M_k]$, $0 \leq k \leq n$の時、$a.M_k$ を分割／統合 d 中の属性生起と呼ぶ。結合条件C_d はｄ中の属性生起より記述され、その属性生起の集合を条件集合（Condition Set ）CS_d と呼ぶ。

(8)　Vは属性の値域の集合を表わす。

(7)　Eは属性生起に関する方程式の集合である。分割／統合 $d : M_0 \rightarrow M_1 \cdots M_n$ 中の親モジュールM_0 の出力属性生起 $a.M_0$, $a \in OUT[M_0]$ と、子モジュールM_k , $1 \leq k \leq n$の入力属性生起 $a.M_k$, $a \in IN[M_k]$ に対して、次の形式の属性方程式$E_{d,v}$が存在する。

$$E_{d,v} : v = f_{d,v}(v_1, \cdots, v_m)$$

ここで、

$$v = a.M_0 , \quad a \in OUT[M_0] ,$$

または、

$$v = a . M_k \quad , a \in IN[\ M_k\ ]$$

ただし、$1 \leqq k \leqq n$。

そしてv_1 ，・・・，v_mは分割／統合ｄ中の他の属性生起であり、

$$DS_{d,v} = \{ v_1 , \cdots , v_m \}$$

を$f_{d,v}$の依存集合（Dpendency Set)と呼ぶ。すなわち、$E_{d,v}$は　分割／統合ｄにおける親モジュールの出力属性値と、子モジュールの入力属性値を計算するための方程式である。そして、

$$E = \cup_{d,v} \quad E_{d,v}$$

とする。

２．２　計算木

次に、Bi-HFPの計算過程を定義するために計算木を定義するが、はじめに　あるモジュールに対する可能な全ての分割／統合の結果を表現した分割／統合木を定義する。これは、cfg の生成規則から導出される導出木に対応しているが、結合条件C_dを考慮せずに次のように再帰的に定義される。

(a)　モジュールは分割／統合木である。

(b)　$d : M_0 \rightarrow M_1 \cdots M_n$を分割／統合規則とするとき、$M_0$を根とし　$t_1 , \ldots , t_n$を部分木とする木$M_0 [\ t_1 \ \ldots \ t_n\ ]$は分割／統合木である。

計算木は、各節点が対応する属性生起によってラベル付けされた分割／統合木のうちで、次の条件を満足するものである。すなわち、分割／統合木の節点M_0に適応された分割／統合規則を$d : M_0 \rightarrow M_1 \cdots M_n$ when C_d とするとき、

(a)　結合条件C_dが成立する。

(b)　分割／統合木にラベル付けされた属性生起ｖに関して、その値が未定義であるか　または、属性方程式$E_{d,v}$が成立するような値が定義されている。

２．３　Bi-HFPの計算結果

最も一般的なBi-HFPの計算結果は、計算木を基に定義される。

＜定義２．２＞　　(Bi-HFP計算結果の定義)

Bi-HFP＝＜$\mathbb{M}$，$\mathbb{M}_{init}$，Top ，Ａ，$\mathbb{D}$，Ｖ，Ｅ＞を満たす計算木において、最上位モジュールTopの出力属性の値が定義されているとき、その値をBi-HFPの計算結果という。

２．４　Bi-HFPの計算過程

次は、プログラムとしてBi-HFPタップルが与えられたときの計算過程を定義するものである。

＜定義２．３＞　　(Bi-HFP計算過程の定義)

Bi-HFP＝＜$\mathbb{M}$，z_0 ，Top ，Ａ，$\mathbb{D}$，Ｖ，Ｅ＞が与えられたとき、計算木集合の列

$$z_1 , z_2 , \cdots$$

をBi-HFPプログラムの計算過程という。ここで、z_{i+1}は　計算木の結合よりz_iから求められたものである。もし、適当なｉ(＞0)に対してz_iが最上位モジュールTopを根とする計算木ｔを含んでいて、ｔにおいてTopの出力属性が定義されていれば　その値をBi-HFPの計算結果という。

＜定義２．４＞　　(計算木の結合)

計算木$t_0 , t_1 , \ldots , t_m \in z_i$と、$1 \leqq j \leqq m$なるすべてのｊに対して、（１）$t_0$の葉モジュール$X_j$と$t_j$の根モジュール$X_j$'の名前が一致し、（２）$X_j$と$X_j$'を持つ分割／統合の結合条件がそれぞれ満たされるとき、t_0の葉X_jを計算木t_jで置換え、評価可能な属性生起をすべて評価した計算木ｔを構成する。

このとき、

$$z_{i+1} = z_i \cup \{ t \} 。$$

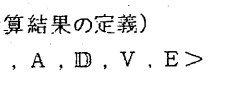
３．　自然言語処理の記述

簡単な日本語的文法に対してその文の意味を抽出するBi-HFPプログラムについて説明する。

３．１　構文解析

入力文字列は　lexical analizerによって音節単位に分割され、対応するモジュールが割当てられる。このモジュールの集合が初期計算木集合$\mathbb{M}_{init}$となる。初期モジュールにはpositionという属性を付加する。（with以下でその値を与える。）

例)　入力文字列$a_1 \ a_2 \ \ldots \ a_n$の各ｉに対して、次のようなモジュールを作る。



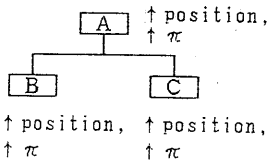

```
with
  position = i
```

構文規則ｐ：Ａ→Ｂ＋Ｃ、ｑ：Ａ→ａに対するBi-HFP分割記述はそれぞれ次のようになる。when句

の結合条件では、B以下、C以下でそれぞれ処理された２つの文字列の連続性が調べられる。with句で示される意味規則は、各モジュールに付加された属性間の関係を表わす。

position.Aには、position.B, position.Cの連接が与えられる。πはパーズ木である。

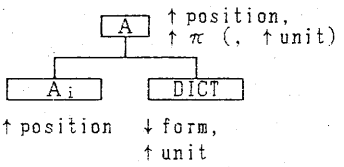
p : A → B + C



when
contiguous[position.B, position.C]
with
position.A = concat[position.B, position.C]
π.A = make-tree[DATAp , π.B , π.C]

q : A → a_i に対する記述は　初期モジュール A_i との結合　及び、辞書モジュールDICTの呼出しを行なう。（辞書引きを行なわないで　この分割記述内に A_i の意味を記述することも可能である。）

q : A → a_i



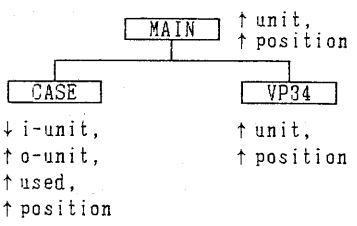
when
always
with
position.A = position.A_i
π.A = DATAq
(form.DICT = 'a_i'
unit.A = unit.DICT)

初期モジュール（A_i）が計算木の葉に相当するものであること、p、q両記述の結合条件が子モジュールの出力属性のみ使われているこから、計算木はbottom up 的に成長する。モジュールDICTはA_i よりtop down的に起動され、属性unitにより値を返す。

３．２ .述語の格フレームについて

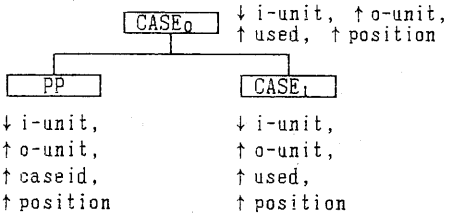
構文解析が終了するとparse 木と対応した計算木が得られる。述語の格フレームは、前節のqのような形式で辞書モジュールDICTより検索されて計算木

p1: MAIN → CASE + VP34



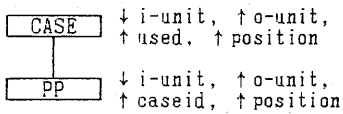
when
contiguous[position.CASE, position.VP34]
with
i-unit.CASE = unit.VP34
unit.MAIN = o-unit.CASE
position.MAIN = concat[position.CASE, position.VP34]

p2: CASE → PP + CASE



when
contiguous[position.PP, position.$CASE_1$]
and caseid.PP ∉ used.$CASE_1$
with
i-unit.PP = i-unit . $CASE_0$
i-unit . $CASE_1$ = o-unit.PP
o-unit . $CASE_0$ = o-unit.$CASE_1$
used . $CASE_0$ =
used.$CASE_1$ ∪ {caseid.PP}
position.MAIN = concat[position.PP, position.$CASE_1$]

p3: CASE → PP



when
always
with
used.CASE = {caseid.PP}

上を流れて行き(unit)、名詞句を処理するモジュールPPによりその名詞句が表わす格に対応するスロットは埋められる。表層格の検出には　名詞に後続する助詞を用いている。

p1では、モジュールVP34により述語を処理して得られた格フレームunit.VP34 が　格を処理するモジュールCASEに送られる。CASEにより得られた情報(unit.CASE) は、モジュールMAINの結果となる。p2の初めの３行の属性方程式は属性unitの”流れ”を作ってやるものである。caseidは、1 つの格を処理するモジュールPPが示している格の種類を表わし、usedは　すでに認識された格を表わす。これを結合条件で調べることによって同一の格が複数回認識されることを防いでいる。これらによりfig.3.1 のようなunitの”流れ”ができる。p3のように、意味規則で指定されていない同一名の属性は　同じ値を持つものとする。

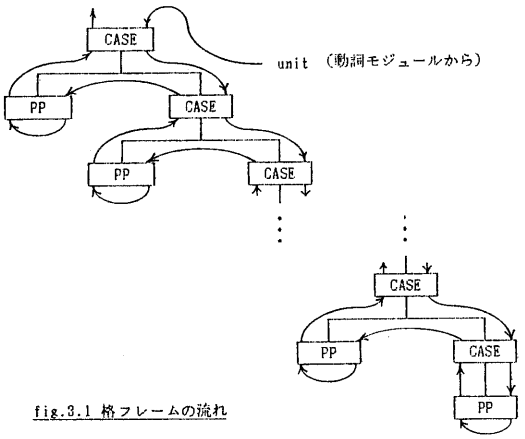


fig.3.1 格フレームの流れ

３．３　名詞の修飾

名詞は形容詞，数量詞（数字に単位がついたもの）などから修飾される。修飾は、被修飾名詞のフレームに　修飾語より得られたスロットを加えるという形で行なわれる。スロットは次のような形をしている：

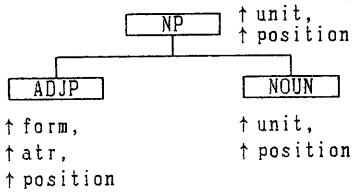
（IRO = SIRO）．．．「色が白い」

（OMOSA = ５Kg）．．「重さ５Kg」

形容詞による名詞修飾についての処理過程は、次の分割記述に示される。

ADJPは形容詞を、NOUNは名詞を処理するモジュールで、両者より名詞句を処理するモジュールNPは構成される。属性atr は形容詞が表わす’性質’を、formは形容詞の原型を示している。これらにより（関数listより）スロットを構成し，関数add-slotによって名詞のフレーム(unit)にこのスロットを加えた新しいフレームをつくり、NPのunitとする。

```
p4: NP  →  ADJP + NOUN
```




```
when
    contiguous[position.ADJP,  position.NOUN]
with
    unit.NP  = add-slot[unit.NOUN,
                        list[atr.ADJP,
                             '= ,
                             form.ADJP]]
    position.NP = concat[position.ADJP,
                         position.NOUN]
```

４．　Bi-HFP評価システムの実現について

４．１　最も一般的なクラスに対して、

このクラスBi-HFPプログラムにおいては、計算の各過程で計算木は根方向にも葉方向にも成長し得る。この場合、計算木を一つのデータ型とみなし、計算木を操作するといった方法により評価器を実現できる。動作はおよそ次のようになる：

評価器は、計算木集合中の２つの計算木を選びそれらが結合可能であるかどうかを判断する。結合可能であれば２つを結合して新しく評価可能になった属性生起に対して属性値計算を行ない　そうしてできた計算木を計算木集合に加える。

２つの計算木を結合するとき、互いに他方のモジュールの属性生起の値を参照することが可能となる。このために新しくいくつかの属性生起の値を計算することができるようになる。計算木上の属性値計算は、意味規則に示される属性値計算のための依存関係をたどることによって　計算木中の計算可能なすべての属性生起を発見できる。

最も一般的なBi-HFPプログラムの計算過程は非決定的である。したがって、評価器を実現するためには、以上の他にバックトラッキング等の探索機構を

考慮しなければならない。これは基本となるデータ型が計算木という複雑な構造のために、状態の保存等に空間的時間的に大きなコストを要してしまい、非常に非効率的である。

我々は Bi-HFPに制限を加えたサブクラスを設定し、より簡単に評価が行なえるようにした。

4．2 Bi-HFPの２つのサブクラス

両制限とも詳細は複雑であるので概略を述べる。１つめの制限(D Bi-HFP 制限) は、決定的の計算過程を持ち、計算木をボトムアップに成長させるためのものである：

(1) すべての分割記述の結合条件は、子モジュールの出力属性生起によって記述されていること。

(2) 初期計算木の要素を分割するような分割記述が存在しないこと。

(3) 一つの計算木には高々一通りの統合の仕方しか存在しないこと。

もう一つの制限によるBi-HFPのサブクラス(EOS Bi-HFP)は、一本のスタックと固定的領域を使って決定的に評価を行なうものである。動作はボトムアップ的だが 計算木はつくらない。これは、入力属性は大域的領域の値の更新という形で実現し、出力属性はスタック上で実現することによって不要な情報（参照されない計算木）を除くようにしたものである：

(1) D Bi-HFPであること。

(2) 入力属性は大域的な使用に限定され、それ以外は出力属性のみであること。

(3) 大域的な属性は、常に左から右に”流れる”こと。

(4) 入力属性の大域的な”流れ”のうちで、一番右の属性生起のみ参照してもよいということ。

4．4 EOS Bi-HFP評価器の実現について

Bi-HFPとLR(1) をfig4.1のように対応させることにより、スタックを基にした評価アルゴリズを考案した。

LR(1) パーザと同様 この評価器もgoto tabele と action table を用いている。goto tableは、LR(1) パーザと同様のものであるが、action tableでは、shift/reduce conflict, reduce/reduce conflictをBi-HFPの結合条件を反映させることによってカバーできるようにしてある。

LR(1)	EOS Bi-HFP
• 終端記号	• initial module
• 非終端記号	• 他のmodule
• 開始記号	• Top module
• 生成規則	• 分割記述
• shift	• initial moduleを スタックへpush
• reduce	• 属性計算とreduce
• accept	• 計算（正常）終了
• error	• 計算（異常）終了

fig4.1 Bi-HFPとLR(1) との対応

属性計算は、スタック上の各モジュールが持つ属性値、及び 大域的固定領域の値から 統 合(reduce)時に計算される。これは、計算木が完成してしまうと根モジュールの出力属性のみ重要で他の情報は保存する必要がないということを利用している。また、固定領域を共有することで大域的属性の値の”流れ”を実現できる。EOS Bi-HFP制限から、全体の計算木が完成していなくても、この”流れ”における大域的属性の計算は可能である。

評価器システムでは、はじめに preprocessor が Bi-HFP意味抽出プログラムからtable を作成し、また、属性計算のための関数を抽出する。評価器は入力処理用モジュールを制御して 入力文字列から終端記号を切取って初期モジュールを作り、initial module queueに enqueueする。この処理と同時にqueueの先頭の初期モジュールとスタック、固定領域の値からtable を参照して評価を進める。計算結果は出力用モジュールによって出力される。

入力文字列から初期モジュールを切出す機構は、action tableが 状態とinitial module queueの先頭の初期モジュールをindex にしていることを利用している。つまり、各状態において評価器は、action tableを見て 値が"error" とならないようなindex を選び、index 中の初期モジュール名を候補とする。

5． EOS Bi-HFPによる自然言語処理について

EOS Bi-HFPによる記述は、辞書の扱いを除いて３章で示された形式で行なわれる。大域的属性としては 与えられた文の環境、動詞の格フレーム等を流

す。日本語の場合、動詞は文末にあるので、格フレームはパーズ木（計算木）上を右から左へ流す方が有利である。このため、EOS Bi-HFP制限を満たように、入力文字列と生成規則を反転させ、文末から解析する手法をとっている。general HFP では、計算木の葉で辞書モジュールDICTを呼出すことによって述語の格フレームが得られ、これを計算木上に流していた。EOS Bi-HFPでは、このような大域的属性は最上位モジュールからの指定のよってinitializeされる。このために何も書かれていないフレームが述語処理のモジュールへ渡され、そこで（関数によって）辞書から得られた述語の格フレームが与えられる。fig5.1(a) はそうして得られた計算木である。fig5.2(b) に(a) における格フレームの流れのうちいくつかの点でのフレームの値を示す。

6．　まとめ

Bi-HFPを定義し自然言語処理の記述を行なった。効率上から、Bi-HFPに制限を加えたサブクラスEOS Bi-HFPを設定し、スタックを用いた評価器を考えた。さらに EOS Bi-HFP によっても　十分に自然言語処理の記述が行なえることがわかった。

参考文献

[1] 電総研推論機構研究室，　拡張LINGOL　(1978)

[2] Katayama, T. :" HFP: A HIERARCHICAL AND FUNCTIONAL PROGRAMMING BASED ON ATTRIBUTE GRAMMAR " , Proc. 5ICSE

[3] Aho, A.V & Ullman, J.D :" Principles of Compiler Design " , ADDISON-WESLEY PUBLISHING COMPANY (1977)

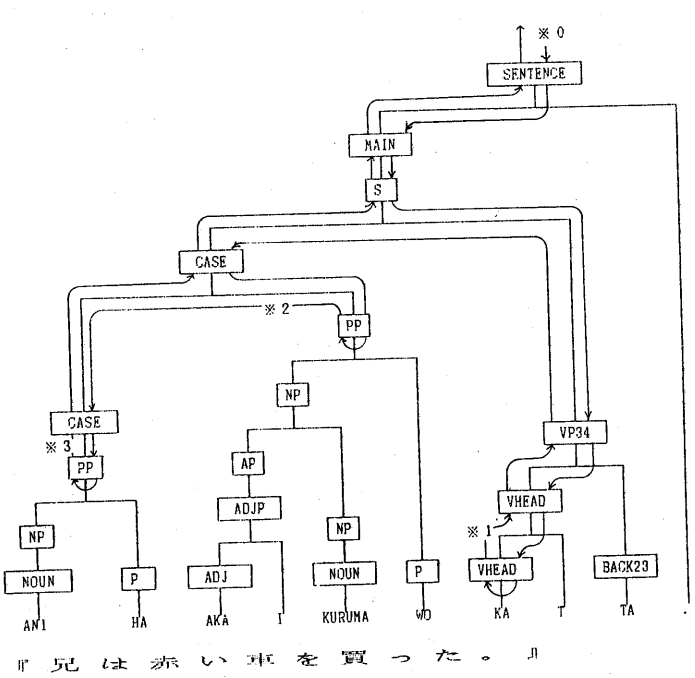


fig.5.1(a)　解析結果の例

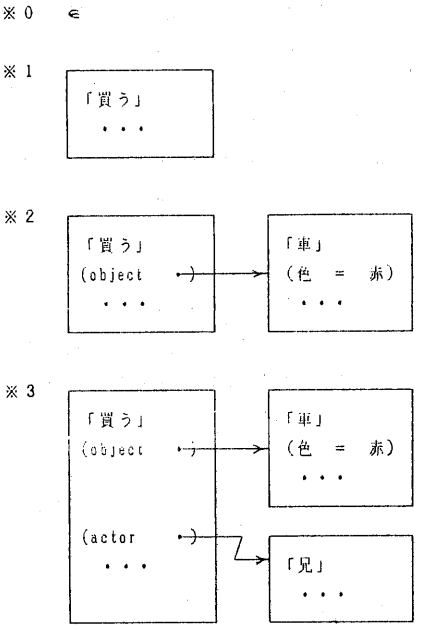


fig.5.1(b)　計算木中のunitの値