Vol. 25 No. 2 情報処理学会論文誌 Mar. 1984

双方向性階層的関数型プログラミング Bi-HFP とその構文解析への応用†

田村直良†† 片山卓也††

本論文では双方向性階層的関数型プログラミング Bi-HFP を提案する．HFP では，一つの処理単位をモジュールによって表す．各モジュールは値の授受のために入力属性，出力属性のリストをもつが，この属性の値がある条件（結合条件）を満たしたときに，親モジュールをより簡単な処理を行う子モジュールに分割する．分割時には，親，子モジュール間の属性方程式（意味規則）に従ってさらにいくつかの属性値が決定される．Bi-HFP においては HFP と同一な記述を，結合条件が満たされたときに子モジュールが親モジュールに統合されるというようにも拡張解釈する．Bi-HFP の記述例としてパーザについて述べる．われわれの方法では，扱う文脈自由文法の非終端記号をモジュールに，生成規則をモジュールの分割統合に対応させる．処理する入力文字列を親モジュールの入力属性に与え，この文字列の先頭が適したものであるかどうかを結合条件に用いるとトップ・ダウン・パーザが記述できる．また，子モジュールに対応する部分文字列が連続であるかどうかを結合条件とするとボトム・アップ・パーザが得られる．両方式とも自然言語の意味についての属性，属性方程式を導入することによって自然言語処理へと拡張することが可能である．われわれはまた，Bi-HFP の操作的意味を定義する．Bi-HFP の計算の状態は，モジュールの階層的分割関係を示す木（計算木）の集合により表されるが，この状態に関する2項関係により Bi-HFP の計算過程は定義される．

1. まえがき

本論文で述べる Bi-HFP のもとになった階層的関数型プログラミング HFP は，片山により提案された仕様記述法であり，階層的記述法におけるモジュールへの入出力関係を関数的に記述することにより階層的かつ関数的なプログラム記述が実現される[4]．HFP は，一般のプログラムにおける計算過程を属性文法[5]における属性値評価過程に対応させて処理の記述を行うようにしたものである．

HFP によってパーザを記述するとき，対象とする文脈自由文法の非終端記号をモジュールに，生成規則をモジュールの分割に対応させることにより，容易に recursive descent なトップ・ダウン・パーザを記述できる．このような方式のパーザとして DCG (Definite Clause Grammar) がある[7]．DCG では，非終端記号／生成規則を Prolog 上で clause/assertion に対応させることによって直構文的なトップ・ダウン・パーザが実現されている．しかし，DCG も HFP による方法も自然言語処理に対して有効なボトム・アップ・パーザは直構文的に記述できない．本論文では，HFP におけるモジュール分割の概念を双方向性に拡張し，ボトム・アップ・パーザを直構文的に記述できる双方向性階層的関数型言語 Bi-HFP を提案する．Bi-HFP は，言語の記述形式は HFP と同一であるが，HFP における記述の解釈を拡張したものである．すなわち，モジュールの属性値がある条件（結合条件）を満たしたときにそのモジュールをいくつかの子モジュールに分割，あるいは，いくつかのモジュールの属性値が結合条件を満たしたときにそれらのモジュールを一つの親モジュールに統合するというように HFP における記述を解釈するようにしたものである．

以下，まず2章で Bi-HFP とその計算結果を定義する．3章では，Bi-HFP により記述したボトム・アップ・パーザを示す．この方式は，意味に関する属性を導入することによって自然言語処理へと拡張することが可能である[10),11)]．4章では，Bi-HFP の操作的意味論について述べる．

2. 双方向性階層的関数型言語 Bi-HFP

この章では，Bi-HFP とその計算過程，計算結果について形式的に定義する．

2.1 Bi-HFP 概説

まず，次の文法に対して記述されたトップ・ダウン・パーザを例として Bi-HFP を概説する．

1 : S→a·S

2 : S→b

† Bi-directional Hierarchical and Functional Programming : Bi-HFP and Its Application to Syntax Analysis by NAOYOSHI TAMURA and TAKUYA KATAYAMA (Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology).

†† 東京工業大学工学部情報工学科

このパーザは，非終端記号 S の処理を行うモジュールSにより構成され，次の二つの分割記述 p1, p2 により表される．記述中，モジュール S の入力属性，出力属性は "↓", "↑" によって明示され，一つの分割記述内に同じモジュール名が現れる場合，添字によって区別する．

p1 : S_0 ↓i-string, ↑o-string, ↑π
　　S_1 ↓i-string, ↑o-string, ↑π
when
　first [i-string. S_0]='a'
with
　$\pi.S_0=1\|\pi.S_1$
　i-string. S_1=rest [o-string. S_0]
　o-string. S_0=o-string. S_1

p2 : S ↓i-string, ↑o-string, ↑π
when
　first [i-string. S]='b'
with
　$\pi.S=2$
　o-string. S=rest [i-string. S]

Bi-HFP では一つの処理単位をモジュールによって表す．各モジュールには値の授受のために入力属性，出力属性のリストが付加されている．

Bi-HFP の一つの記述単位を分割記述といい，次の三つから構成される．

（a） 分割統合（または分割統合規則）：一つの親モジュールと任意個（≧0）の子モジュールの階層的分割関係を表したもの．

（b） 結合条件：分割統合を行ってよいかどうかを判定する述語で，**when** 以下に示される．

（c） 意味規則：各モジュールがもつ入出力属性間の関係を等式により表現したもので，**with** 以下に示される．

p1 では，親モジュールの入力属性 i-string. S_0 へ与えられた文字列の先頭を調べて分割の可否を判断する．分割が可能な場合，親モジュールに渡された入力文字列から先頭の終端記号を除去した残りの文字列が子モジュールへ渡され (i-string. S_1)，子モジュールで未処理の文字列 (o-string. S_0) が親モジュールの出力属性 o-string. S_0 に返される．パーズの結果 $\pi.S_0$ は，生成規則の番号に子モジュールより得られた部分パーズ $\pi.S_1$ を連接 (‖) することによって得られる．この方式では，LL(1) に属する文法に対しては決定的な計算過程をもつ．

計算を起動するために最初に与えるモジュールを初期モジュールといい，次のように初期記号に相当するモジュール S の入力属性 i-string にパーズしようとする文字列を与える．

S ↓i-string, ↑o-string, ↑π
with
　i-string. S='aaaaaaab'

計算の結果は，最上位モジュールと指定されたモジュールの出力属性の値である．この例の場合は，S が最上位モジュールである．

2.2 Bi-HFP の形式的定義

次に，Bi-HFP を定義する．

[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．

$$\text{Bi-HFP}=<\mathbf{M}, \mathbf{M}_{init}, \text{Top}, A, \mathbf{D}, V, E>$$

ここで，

（1） $\mathbf{M}$ はモジュールの集合である．

（2） $\mathbf{M}_{init} \subset \mathbf{M}$ は初期モジュールの集合である．

（3） $\text{Top} \in \mathbf{M}$ 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である．モジュール $M \in \mathbf{M}$ の入力属性集合と出力属性集合を，おのおの IN[M], OUT[M] と定義するとき，

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

かつ，

$$IN[M] \cap OUT[M]=\phi.$$

（5） $\mathbf{D} \subset \mathbf{M} \times \mathbf{M}^*$ はモジュール分割統合の集合であり，$d \in \mathbf{D}$ 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** $C_d(n \geqq 0)$,

ここで，$M_0, M_1, \cdots, M_n \in \mathbf{M}$ であり，とくに $n=0$ の場合 $d : M_0$ **when** C_d を終端分割規則と呼ぶ．

$a \in A[M_k]$, $0 \leqq k \leqq n$ のとき，$a.M_k$ を分割統合 d 中の属性生起と呼ぶ．

C_d を結合条件と呼び，これが真のときモジュール M_0 が，$M_1, \cdots, M_n$ 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．結合条件 C_d は d 中の属性生起より記述され，その記述中の属性生起の集合を条件集合 CS_d と呼ぶ．

（6） V は属性の値域を要素とする集合を表す．

（7） E は意味規則の集合を表す．つまり，E は分割統合 d における親モジュールの出力属性や子モジ

ュールの入力属性の値を計算するための方程式 $E_{d,v}$ の集合であり，

$$E=\bigcup_{d,v} E_{d,v}$$

分割統合 $d: M_0 \to M_1 \cdots M_n$ の親モジュール M_0 の出力属性生起 $a.M_0$ (ただし，$a \in OUT[M_0]$) と，子モジュール M_k ($1 \leqq k \leqq n$) の入力属性生起 $b.M_k$ (ただし，$b \in IN[M_k]$) に対して，次の形式の属性方程式 $E_{d,v}$ がそれぞれ存在する．

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

ここで，

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

または，

$$v=b.M_k,\ b \in IN[M_k]$$

ただし，$1 \leqq k \leqq n$

そして $v_1, \cdots, v_m$ は分割統合 d 中の他の属性生起である．

2.3 Bi-HFP の計算過程および計算結果

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，はじめにモジュールに対する可能なすべての分割統合の結果を表現した分割統合木を定義する．

[定義 2.2] (分割統合木)

分割統合木は，次のように再帰的に定義される．

(a) 一つのモジュールは分割統合木である．

(b) $M_1, \cdots, M_n$ を根とする分割統合木 $t_1, \cdots, t_n$ に対して分割統合規則 $d: M_0 \to M_1 \cdots M_n$ が存在するとき，M_0 を根とし $t_1, \cdots, t_n$ を部分木とする木 $M_0[t_1 \cdots t_n]$ は分割統合木である．

計算木は，分割統合木に結合条件，意味規則を考慮したもので，次のように定義される．

[定義 2.3] (計算木)

計算木は，各節点を対応する属性生起の値によってラベル付けした分割統合木において，次の条件を満足するもののうち，可能な限り属性値が評価されているものである．すなわち，分割統合木の節点 M_0 に対応した分割統合規則 $d: M_0 \to M_1 \cdots M_n$ **when** C_d と，分割統合木にラベル付けされた属性生起 v に対して，

(a) 条件集合 CS_d の要素すべてが定義されているとき，条件 C_d が成立する．

(b) 属性生起 v に関してその値が定義されているとき，属性方程式 $E_{d,v}$ が成立する．

分割統合規則も一つの計算木である．Bi-HFP の計算結果は，計算木の根である最上位モジュール Top から得られる．

[定義 2.4] (Bi-HFP 計算結果)

最上位モジュール Top を根とする計算木において，根モジュール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．

Bi-HFP の計算木，計算結果は以上のように静的に定義されるが，与えられた初期モジュール集合に対してしだいに分割統合が行われて結果が得られるとする動的な立場から，計算過程が次のように定義される．

[定義 2.5] (Bi-HFP 計算過程)

Bi-HFP=$\langle \mathbf{M}, z_0, \text{Top}, A, \mathbf{D}, V, E \rangle$ が与えられたとき，計算木集合の列

$$z_0, z_1, \cdots$$

を Bi-HFP プログラムの計算過程という．ここで，z_{i+1} は次のようにして計算木の結合により z_i から求められたものである．計算木 $t_l, t_u \in z_i \cup \mathbf{D}$ に対して，t_u の一つの葉モジュール X と t_l の根モジュール X' の名前が一致するとき，t_u の葉 X を計算木 t_l で置き換えた分割統合木 t を構成し，そのとき，t に対応する計算木 t' が存在するならば，

$$z_{i+1}=z_i \cup \{t'\}$$

とする．この操作を計算木の結合という．

図 1 に計算木の結合を示す．結合操作の特別な場合としてモジュールの分割と計算木の統合があるが，こ

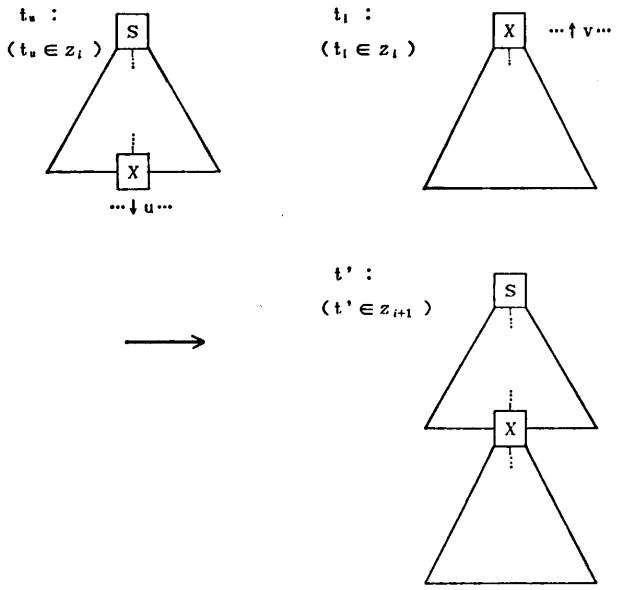
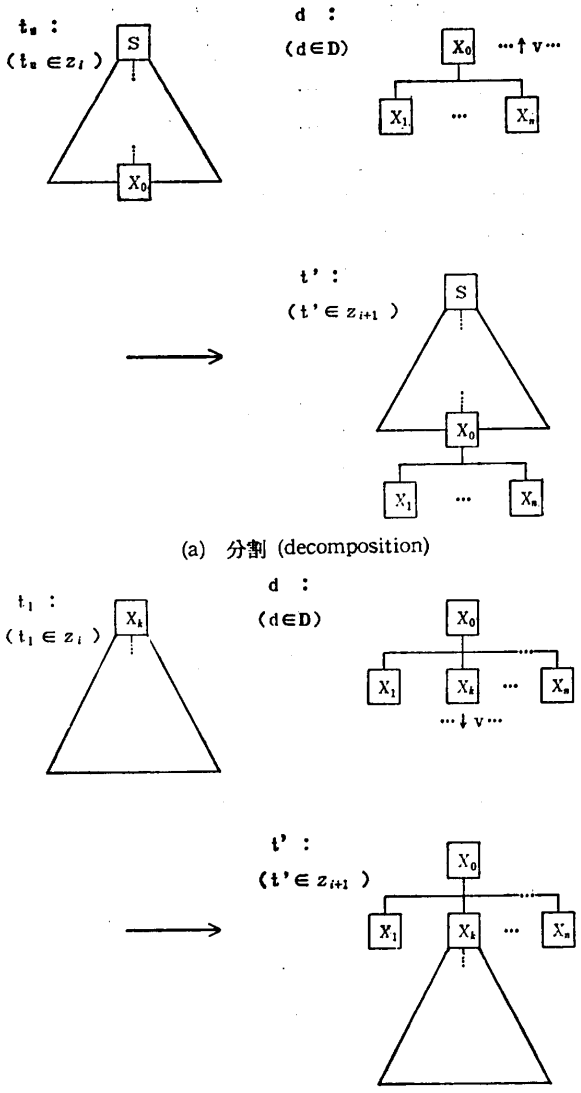


図 1 計算木の結合
Fig. 1 A connection of computation trees.



(a) 分割 (decomposition)

(b) 統合 (composition)

図 2 モジュールの分割統合

Fig. 2 Composition and decomposition of modules.

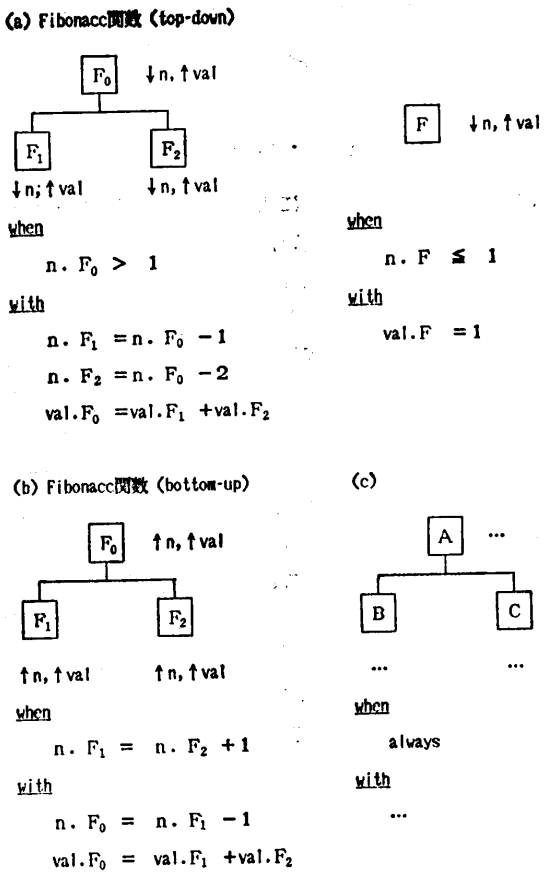


図 3 Bi-HFP の記述例

Fig. 3 An example of Bi-HFP discription.

れらは結合における一方の計算木が分割統合規則である場合である（**図2参照**）．ある計算木の集合 z_i には，一般に複数の結合の仕方が考えられる．このために，Bi-HFP の計算過程は非決定性である，**図3**(a)，(b)，(c)にそれぞれ分割，統合，結合の行われる簡単なBi-HFPプログラム例を示す．条件集合が親モジュールの入力属性のみから成る分割統合規則によって構成された計算木が葉モジュールAをもつとき，結合の可否は(a)のようにモジュールAを分割しようとしている分割統合規則の結合条件による．つまり，この分割統合規則はトップ・ダウン的に動作してAがB，Cに分割されることを示す．図3(b)の形式で書かれたプログラムでは，逆にモジュール B, C を根とする計算木が存在するとき，B, C を A に統合しようとしている分割統合規則の結合条件が満たされるとAを根とする計算木がボトム・アップ的に生成される．図3(c)はつねに結合条件が成立するので，トップ・ダウン的動作にもボトム・アップ的動作にも使用される．

つまり，初期モジュールとして最上位モジュールだけが与えられ，親モジュールの入力属性生起のみを使って結合条件が記述されているような Bi-HFP は，HFP とまったく同じトップ・ダウン的動作を行う．また，子モジュールの出力属性のみを使って結合条件が書かれているような Bi-HFP はボトム・アップ的動作を行う．

なお，[定義 2.5] で定義される計算結果は [定義 2.4] のサブセットである．つまり，[定義 2.4] では，すべての分割統合木のなかから与えられた結合条件，意味規則を満たす計算木を選択しているのに対

し，[定義 2.5] では，計算過程におけるすべての分割統合木で結合条件，意味規則が成立することが要求される．

3. ボトム・アップ・パーザの記述

この章では，Bi-HFP によりボトム・アップ・パーザを記述する．ここで用いた方法では，扱う文脈自由文法の非終端記号をモジュールに，終端記号を初期モジュールに，生成規則を分割統合に対応させ，結合条件と意味規則内に構文解析の機構を組み込むようにしたものである．つまり，各分割記述はその中の子モジュールに対応する部分文字列が連続であるかどうかを結合条件に用いて，条件が満たされれば子モジュールを親モジュールに統合する．また，この方法では，frame 等の知識表現を用いた自然言語の意味に関する属性や意味規則を導入することにより，自然言語処理へと拡張することが可能である[10),11)]．対象とする文法としては簡単のために次のような文法を用い，1と3についての記述を示す．

1: SENT→PRN・VP
2: VP→V・N
3: PRN→'I'
4: V→'have'
5: N→'books'

(i) モジュール

SENT, …, N ↑position, ↑π

TERM ↑position, ↑symbol

各モジュールに対して次のような用途で属性を使用する：

↑position: この属性をもつモジュールで処理された部分文字列の入力文字列中での位置を示す．

↑π: モジュールで処理された入力文字列に対する部分パーズ木を示す．

↑symbol: モジュール TERM に対応する実際の終端記号を表す．

(ii) 初期モジュール

入力文字列を終端記号ごとに分解しその各終端記号に対してモジュール TERM を割り当てる．position にはそのモジュールが表す終端記号の入力文字列 'I have books' における位置を与える．

t_1: TERM ↑position, … t_3: TERM ↑position,
↑symbol ↑symbol
with **with**
symbol='I' … symbol='books'
position=1 … position=3

(iii) 最上位モジュール

スタート・シンボル SENT を指定する．

(iv) 分割記述

p1: SENT ↑position, ↑π
PRN VP
↑position, ↑position,
↑π ↑π
when
contiguous[position. PRN, position. VP]
with
π. SENT
=make-tree[$DATA_{p_1}$, π. PRN, π. VP]
position. SENT
=concat[position. PRN, position. VP]

結合条件として，モジュール PRN 以下で処理された文字列とモジュール VP 以下で処理された文字列の連続性が使用される．この分割記述が採用された場合，position. PRN と position. VP の連接が，モジュール SENT 以下で処理された文字列として position. SENT に与えられる．

p3: PRN ↑position, ↑π
TERM ↑position, ↑symbol
when
symbol. TERM='I'
with
π. PRN=make-leaf[$DATA_{p_3}$]
position. PRN=position. TERM

終端記号はすべて TERM というモジュールによって表され，その symbol という属性が実際の終端記号を示している．結合条件では属性 symbol の値が 'I' であるかどうかを調べている．

図4 にこの方式で処理した場合の計算過程を示す．

4. Bi-HFP の意味

この章では，Bi-HFP=⟨**M**, z_0, Top, A, **D**, V, E⟩ の操作的意味について述べる．Bi-HFP の計算過程が非決定的であるために，ここでは初期状態 x と計算過程

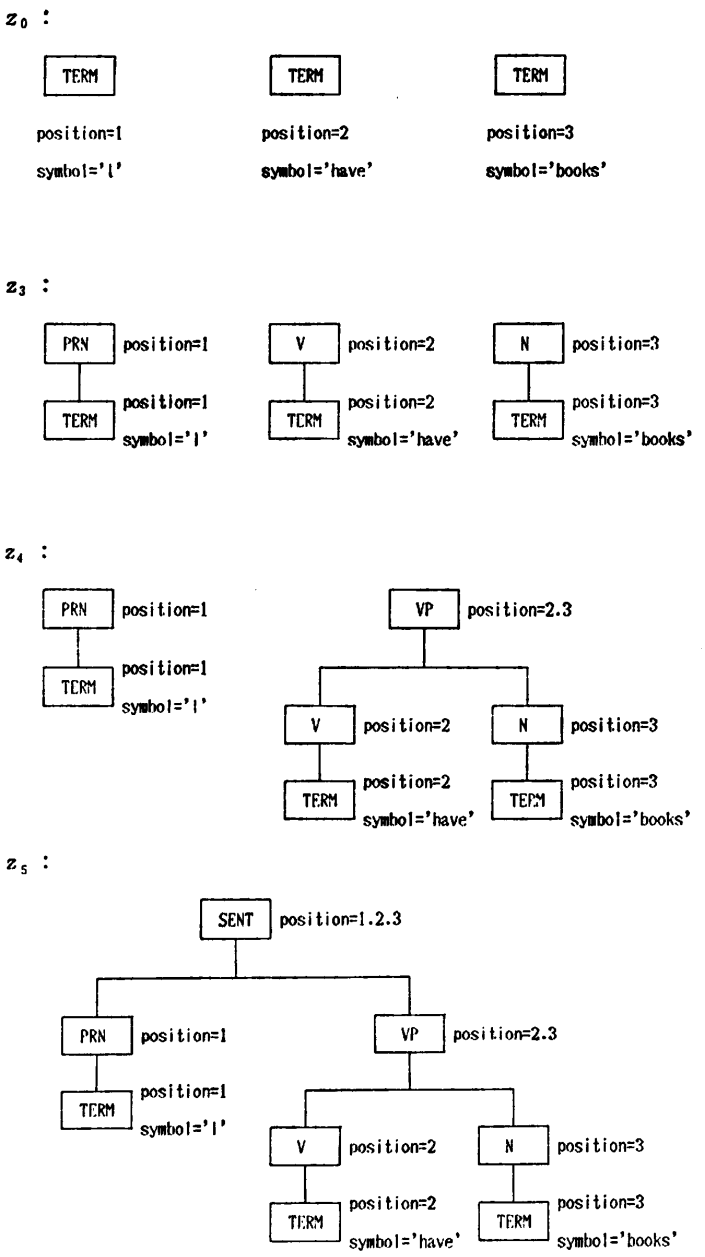


図 4 Bi-HFP 計算過程の例
Fig. 4 An example of computation process of Bi-HFP.

の最終状態 y との間の関係 xM(Bi-HFP)y を Bi-HFP の意味 M(Bi-HFP) とする.

まず, 計算過程の状態集合を定義する.

[定義 4.1] (状態空間)

Bi-HFP の計算過程における状態の集合とは, 計算木集合を要素とする集合 Q である.

次に, 結合についての関係, および計算過程の終了を判定するための述語を与える. x, y ∈ Q に対して, 2 項関係 xCy は, 状態 x におけるある計算木において結合が行われて新しい状態 y が作られたことを表す. また, 述語 Terminate (Top, x) は, 状態 x に最上位モジュールを Top とするときの計算結果が得られていることを示す. 2 項関係 C と述語 Terminate を定義するために定義中で使用される関数, 述語とその定義域を次のように定める.

〔領域〕

T_c : 計算木の集合
T_d : 分割統合木の集合
S : 計算木中の一つの葉を選択するセレクタの集合
N : 計算木の節点の集合
M : モジュールの集合

〔関数, 述語〕

Exist : $T_d \rightarrow$ Boolean
分割統合木に対して対応する計算木が存在することを示す述語

assign : $T_c \times S \times T_c \rightarrow T_d$
セレクタで指定される計算木の葉を他の計算木で置換えた分割統合木を求める関数

evaluate : $T_d \rightarrow T_c$
与えられた分割統合木において, 評価可能な属性をすべて評価して計算木を求める関数

root : $T_c \rightarrow$ N
計算木の根となる節点を求める関数

module : N→M
節点からモジュール名を求める関数

Defined : N→Boolean
与えられた節点の出力属性がすべて評価されていることを示す述語

〔関係C〕

C≡
{⟨z, z'⟩|
∃ $t_l, t_u \in z \cup \mathbf{D}$, ∃s ∈ S:
z'=z ∪ {t}
and t=evaluate[assign[t_u, s, t_l]]
and Exist[assign[t_u, s, t_l]]}

〔述語 Terminate〕

Terminate(Top, z)
≡∃t∈z: [module[root[t]]=Top
and Defined[root[t]]]

ある状態xに対して，一般には分割，統合，結合のうちのいくつかの操作が可能である．これらのうちには計算過程を停止させなくする操作も存在するし，また，異なる操作でも同一の計算結果をもつものもある．このように，計算過程は非決定的であるが，状態の遷移を関係についての列としてとらえることによって計算過程を形式的に定義する．

[定義 4.2] (計算過程)

Bi-HFP=⟨**M**, z_0, Top, A, **D**, V, E⟩ の計算過程 z_0, z_1, … は，2項関係Cを満たす状態の列として定義される．すなわち，

$\langle z_i, z_{i+1}\rangle \in C \quad (i \geqq 0)$

計算過程は次のように，(i)停止して結果が得られる，(ii)停止するが結果は得られない，(iii)停止しない，の3種のうちのいずれかの形をしている．

(i) (proper termination)

$z_0 z_1 \cdots z_n$

すなわち，

$\forall i \in [1, n]$, $\exists z_i \in Q$:
$z_{i-1} C z_i$,
かつ, ～Terminate[Top, z_{i-1}],
Terminate[Top, z_n]

(ii) (improper termination)

$z_0 z_1 \cdots z_m$

すなわち，

$\forall i \in [1, m]$, $\exists z_i \in Q$:
$z_{i-1} C z_i$,
かつ, ～Terminate[Top, z_{i-1}],
～Terminate[Top, z_m],
$\forall z \in Q$: ～$z_m C z$

(iii) (nontermination)

$z_0 z_1 \cdots z_k \cdots$

すなわち，

$\forall i \geqq 0$, $\exists z_i, z_{i+1} \in Q$:
$z_i C z_{i+1}$,
かつ, ～Terminate[Top, z_i].

このうち，(i)の計算過程をQ上の関係 "⇒" を用いて，

$z_0 \Rightarrow z_n$

と表す．これにより，Bi-HFP の意味は次のようになる．

[定義 4.3] (Bi-HFP の意味)

Bi-HFP=⟨**M**, z_0, Top, A, **D**, V, E⟩ が与えられたとき，Bi-HFP の意味 M(Bi-HFP) は次のような関係として定義される．

$M(\text{Bi-HFP}) = \{\langle z_0, z\rangle \mid z_0 \Rightarrow z, z \in Q\}$

5. むすび

本論文では，プログラムの仕様記述法 Bi-HFP を提案し，その意味を定義し，構文解析への応用について述べた．プログラムの作成は純粋なトップ・ダウン，ボトム・アップの手段によってのみならず，それらの混合的手段を使って行われるほうがむしろ普通であると考えられる．Bi-HFP は，一つの処理を表すモジュールの階層的分割関係を記述単位としてプログラムするものである．この記述内では，各属性の値が満足すべき述語，方程式のみが示されていて，利用者はモジュールの階層関係がトップ・ダウン的であるかボトム・アップ的であるかを意識しないでよい．このようなプログラム形態は，記述的であるという点で Prolog と似ているが，Prolog はプログラムの非決定的要素をトップ・ダウン的な探索により解決している．

本論文で定義した Bi-HFP の計算過程は非決定的である．これに対して，関係を導入することにより非決定的な計算過程の意味を形式的に扱うことができた．また，本論文で述べたパーザの記述は，生成規則ごとに行われるが，各記述中に frame 等の知識表現を用いた属性や意味規則を導入することによって，自然言語処理の記述へと拡張することが可能である．さらに，このような記述から成る Bi-HFP のサブ・クラスに対して，通常のボトム・アップ・パーザをもとにして計算木を作らずにスタック上で評価を行う Bi-HFP 評価器を作ることができる．

参考文献

1) de Bakker, J. W.: Automata Languages and Programming, in Michaelson, S. and Milner, R. (eds.), *Semantics and Termination of Nondeterministic Recursive Program*, pp. 435-477, Edinburgh University Press, Edinburgh, (1976).
2) Guttag, J. V., Horowits, E. and Musser, D. R.: The Design of Data Type Specification, in Yeh, R. T. (ed.), *Current Trends in Programming Methodology*, Prentice-Hall, Englewood Cliffs (1978).
3) Harrison, M. A.: *Introduction to Formal*

Language Theory, Addison-Wesley, Reading (1978).
4) Katayama, T.: HFP: A Hierarchical and Functional Programming Based on Attribute Grammar, Proc. 51 CSE (1980).
5) Knuth, D. E.: Semantics of Context-Free Language, *Math. Syst. Theory*, Vol. 2, No. 2, pp. 127-145 (1968).
6) Kowalski, R.: And-Or Graphs, Theorem-Proving Graphs and Bi-Directional Search, *Mach. Intell.*, Vol. 7, Edinburgh University Press, Edinburgh (1972).
7) Pereira, F. and Warren, D.: Definite Clause Grammar for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artif. Intell.*, Vol. 13, pp. 231-278 (1980).
8) van Emden, M. H. and Kowalski, R. A.: The Semantics of Predicate Logic as a Programming Language, *JACM*, Vol. 23, No. 4, pp. 733-743 (1976).
9) VanderBrug, G. J. and Minker, J.: State-Space, Problem-Reduction, and Theorem Proving—Some Relationships, *CACM*, Vol. 18, No. 2, pp. 107-115 (1975).
10) 田村, 片山: 双方向性階層的関数型言語 Bi-HFP とその自然言語処理への応用, 情報処理学会第25回全国大会 (1982).
11) 田村, 片山: 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, 自然言語処理研究会, 34-5 (1982. 12).

(昭和58年5月16日受付)
(昭和58年9月13日採録)