ソフトウェア基礎論　3～7
（1982. 12. 10）

階層的関数型言語HFPの実現

石川　徹　　片山　卓也　　塩田　憲行
（東京工業大学・工学部）

1．まえがき

筆者等は、階層的関数型プログラミング(Hierarchical Functional Programming)　用言語 HFPL-2 を開発中である。HFP の概念は、Knuth の属性文法に基礎を置き、HFP におけるプログラムの記述は属性と呼ばれる入力、出力を伴なったモジュール及び、その分割から構成される。本報告では、HFP 記述用言語HFPL-2の仕様を紹介し、その実現方法、効率化手法を述べる。

プログラム言語の意味記述の手段として属性文法は優れた方法であり、とりわけコンパイラの自動生成を考える際には非常に有用な形式的意味記述法であることは良く知られている。HFP は属性文法を拡張することによって、一般の計算過程を表現できるようにした階層的関数型計算モデルであり、Prologなどとも深い関係がある。

HFP における計算は、計算木の構成と、その上での属性評価という２つの面を持つが、後者は属性文法のそれと同一である。

属性文法における効率的な属性評価方法の研究も色々と行なわれているが、現在までのところ実用に絶えるような評価機構は確立されていないのが実状である。　HFPL-2処理系ではHFPL-2によって記述された記号列を再起手続き群に変換することによって、属性の管理的作業を属性評価時以前に行ない、属性評価時には純粋に属性の評価のみを行なうことにより、属性評価の効率を高めるとともに、属性の同時評価、記憶大域化、値によるデータアクセスから部分的変更によるアクセスへの変換、再帰構造の繰り返し構造への変換などによって属性を効率的に評価する手続き群を生成する。

2．HFPの形式的定義

HFP は次の６つの項組として定義される。

$HFP=<\mathbb{M}, M_{init}, A, \mathbb{D}, V, E>$

ここで

(1) $\mathbb{M}$はモジュールの集合であり、分割を終了させるための特別なnullモジュールを含む。

(2) $M_{init} \in \mathbb{M}$は初期モジュールである。

(3) Aはモジュールの入出力属性集合であり、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]としたとき、

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

かつ　$IN[M] \cap OUT[M] = \Phi$

(4) $\mathbb{D}$はモジュール分割の集合であり、$d \in \mathbb{D}$は分割規則と呼ばれ、次のように表現される。

$$d : M_0 \rightarrow M_1, \cdots, M_n$$
$$\underline{when}\ \ C_d$$
$$\underline{with}\ \ EQ_d$$

ここで、$M_0, \cdots, M_n \in \mathbb{M}$であり、特に$n = 0$の場合

$$d : M_0\ \underline{when}\ C_d\ \underline{with}\ EQ_d$$

を終端分割規則と呼ぶ。C_dを分割条件と呼び、これが真のときモジュールM_0が$M_1, \cdots, M_n$に分割されるという。EQ_dはdに付随した属性定義式の集合である。$a \in A[M_k]$，$0 \leq k \leq n$のとき$M_k.a$を分割d中の属性生起と呼ぶ。

(5) Vは属性の値域の集合を表わす。

(6) Eは属性生起に関する定義式の集合である。分割$d : M_0 \rightarrow M_1, \cdots, M_n$中での親モジュール$M_0$の出力属性生起$M_0.a$，$a \in OUT[M_0]$と子モジュール$M_k$，$1 \leq k \leq n$の入力属性生起$M_k.a$，$a \in IN[M_k]$に対して、次の属性定義式$E_{d_N} \in E_d$が存在する。

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

ここで、

$v = M_0.a$, $a \in OUT[M_0]$, または

$v = M_k.a$, $a \in IN[M_k]$, $1 \leqq k \leqq n$

そして $v_1, \ldots, v_m$ は分割 d 中の他の属性生起であり、

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

を $f_{d,v}$ の依存集合と呼ぶ。

すなわち、$E_{d,v}$ は分割 d 中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定義式である。

〖簡単なHFP プログラム例〗

図1は、x番目のフィボナッチ数を求めるプログラムをHFP によって記述した例である。入力xと出力vを用意したモジュールfib を準備し、その入出力関係を図のように定義する。図中when節によりfib の分割条件が表現されており、たとえば入力xの値が2以上なら、fib は2つの子モジュールに分割される。with節は，属性定義部でありモジュール間の入出力の受け渡しを定義している。
け渡しを定義している。

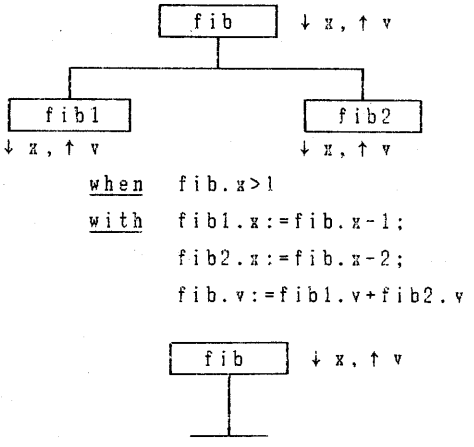


図1 HFP によるフィボナッチ数

3．決定性絶対非循環 HFP

〖決定性 HFP〗

次に示す制限を満足するものを決定性HFPと呼ぶ。

(1) 分割規則

$d : M_0 \rightarrow M_1, \ldots, M_n$
when C_d
with EQ_d

内の分割条件 C_d は親モジュール M_0 の入力属性生起 $M_0.a$, $a \in IN[M_0]$ によって記述される。ただし、そのような入力属性生起は任意の計算木においてその M_0 の出力属性生起に依存してはならない。

(2) 同一モジュールに対する分割規則を $d_1, \ldots, d_k$ とするとき、

$C_{d_i} \wedge C_{d_j} = false$
$1 \leqq i \neq j \leqq k$

(3) $\forall i \; C_{d_i} = true$

制限(1)はモジュール M_0 を分割する際に、その分割の適否を現在既知である属性値のよって決定できるための条件である。制限(2)は、モジュールの分割が計算木中で唯一に決定できるための条件である。制限(3)は同一モジュールに対する分割条件の記述が不完全なために分割が進行できなくならないための条件である。

〖絶対非循環HFP 〗

HFP はモジュールの属性が入力用、出力用と厳密に区別されているため、これを利用して事前にデータ依存解析を行なうことによって効率的な実現を計ることができる。以下に、いくつかの有用なデータ依存関係を与える。

• 属性依存グラフ DG_d ：分割 d 中における属性生起を節点とする有向グラフで、その枝は属性定義式意 EQ_d によって与えられる属性の依存関係を表現している。分割規則 $d : M_0 \rightarrow M_1, \ldots, M_n$ when C_d に対する属性依存グラフ DG_d は、次のように定義される。

$DG_d = (DV_d, DE_d)$

・構造体に関するオペレーション

HFPL-2ではPascalのように、構造体に対して、部分的な代入によってのデータへのアクセスは、許されていない。属性定義式はあくまでも、ある属性生起の値を他の属性生起によって定義するためのものである。そこでHFPL-2構造体の各型には、selector, update-constructor, constructor のオペレータが用意されている。たとえば配列にたいして、

selector a[i]：配列aのi番目の要素を取り出す。

update-constructor [a:i:x]:配列aのi番目の要素の値をxの値に変更した配列。

constructor [x,y,z, …]:要素の値がx，y，z，…からなる一次元配列。

が使用できる。

・リスト型

HFPL-2で扱うことのできるのは、Lisp型のリストで、要素は、関数型、ファイル型、集合型を除く型である。ただしすべての要素の型は一致いなければならない。

・ファイル型

ファイル型は大域化属性として実現されるため、後述する大域化可能な属性でなければならない。

・関数型

HFPL-2では属性の型として関数型を許している。これは一般の高階の関数を構成するのが目的ではなく、属性定義式や分割条件で用いられる関数を属性とするようなモジュールを構成するためである。従って関数に対する演算は用意されていない。

〖HFPL-2の構文〗

HFPL-2のプログラムは、大きく次の3つの部分から成る。

(1) プログラム先頭部：初期モジュールを指定するとともに、その初期モジュールに与える入力属性値を定義する。

(2) 宣言部：定数宣言，型宣言およびモジュールの宣言を行なう。型宣言、定数宣言は、Pascalの宣言とほぼ同じであり、モジュール宣言部ではプログラム中で使用されるすべてのモジュールとそれに伴なう入出力属性名、その型を宣言する。

(3) 分割記述部：分割記述部はさらに次の3つの部分から成る。

・分割規則記述部　モジュール間の分割の仕方を定義するもので、→の右辺のモジュールが左辺のサブモジュールに分割されることを示している。右辺がnullのとき、終端分割と呼び、分割が終了することを示す。分割規則には、モジュール名と共にそれに付随する属性生起名を付記する。属性生起名の並びは、モジュール宣言部での属性名の並びに、対応する。また親モジュールの出力属性、子モジュールの入力属性の所には、属性生起名の代わりに式を書くことが許されており、属性の受け渡しの関係が簡単な場合にはその属性方程式のかわりに、意味規則を分割規則中に記述することが可能である。

・分割条件記述部　その分割が行なわれる条件を示し、この条件が成立したときのみ、その分割が行なわれる。分割条件がotherwise のときには、そのモジュールに関する他のすべての分割条件が成り立たないときにその分割が行なわれる。図1中のwhen節にあたる部分である。

・属性定義式記述部　モジュール間の属性の受け渡しの関係を定義する。すなわち、親モジュールの入力属性生起と、子モジュールの出力属性生起の値を他の属性生起を使って記述する。図1中のwith節に対応する

分割記述部では、属性生起名は、宣言部の属性名の並びに、数の対応さえとれていればどんな識別名を使ってもよい。さらに、複数の属性生起に同じ名前を与えることによって同一名属性生起間の値の受け渡しが自動的に行なわれる。

$DV_d = \{M_k.a \mid 0 \leqq k \leqq n \wedge a \in A[M_k]\}$

$DE_d = \{\langle v_1, v_2\rangle \mid v_1 \in DS_{dv2}\}$

$\cup \{\langle v_1, v_2\rangle \mid v_1 \in CS_d \wedge v_2 = M_0.a \wedge a \in OUT[M_0]\}$

ここでCS_dは、分割条件C_d中に現れる属性生起からなる集合である。

・IOグラフIO[M,T]:モジュールMを根とする計算木Tにおいて

IO[M,T] = (v, E)

ただし、v = A[M], E ⊂ IN[M] × OUT[M]であり、

<a1,a2>∈E ←→ 計算木Tにもとでの属性の評価において、根Mの出力属性a2の計算には入力属性a1が必要である。

・OIグラフOI[M,T]:初期モジュールM_0を根とし、モジュールMをその葉に含む計算木Tにおいて、

OI[M,T] = (v, E)

ただし、v = A[M], E ⊂ OUT[M] × IN[M]であり、

<a1,a2>∈E ←→ 計算木Tのもとでの属性の評価において、葉Mの入力属性a2の計算には出力属性a1が必要である。

IO[M,T],OI[M,T]ともに、計算木Tによって複数個存在するが、それらの集合を各々IO[M],OI[M]によって表わす。

・拡張属性依存グラフDGd^*:分割dに対する拡張属性依存グラフDGd^*は次のように定義される。

$DGd^* = (DVd, DEd^*)$

ただし、ここで

ある (a1,a2)∈IO[Mk]に対して、

$e \in DEd^* \leftrightarrow e \in DEd$ または、

e=(a1,a2), a1∈IN[Mk], a2∈OUT[Mk]

・属性依存グラフDG[M]:モジュールMの属性を節点とする有向グラフであり、拡張属性依存グラフDG_d^*を用いて次のように定義できる。

DG[M] = (A[M], E)

E = {<a,b> | a∈IN[M] ∧ b∈OUT[M]

・predM(a):モジュールMとその出力属性a∈OUT[M]に対して、

predM(a) = {b | (b, a) ∈ DG[M]}

すなわち、predM(a)はDG[M]における出力属性aのpredecessorであるような入力属性の集合である。

・絶対非循環HFP:HFPの任意の分割規則d∈𝔻に対して、その拡張依存グラフDG_d^*が有向サイクルを含まない場合かつその場合に限りそのHFPは絶対非循環であるという。

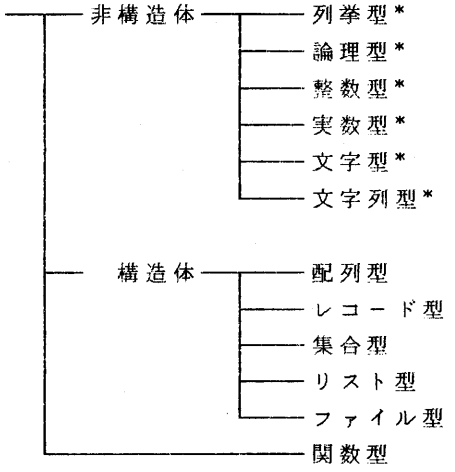
絶対非循環HFPは、実用上十分に広いHFPのクラスであると同時に、このクラスのHFPに対して、データ依存解析を事前に行うことにより、効率のよい実現を行うことができる。またこれは、HFPの記述を手続きに変換する場合に必要な条件でもある。

以下では単に、HFPといったときは、絶対非循環なHFPを指す。

4. HFPL-2の言語仕様の概略

HFPL-2はHFPモデルによって計算を記述するための関数型プログラム言語である。プログラムはモジュールとモジュール分割に関する記述から成るが、扱うことのできるデータ構造はPascalのそれ準拠しており、次のようなものがある。

〖HFPL-2のデータ型〗



*のついた型はPascalの相当する型と同じであり、宣言、使用もほぼ同様に行なえる。以下では、その相違についてのみ述べる。

〖HFPLプログラムの簡単な例〗

図２は、図１のHFP 記述をHFPL-2で記述したものである。

５．HFPL-2処理系の実現

この章では、HFPL-2の処理系の実現について述べる。まず最初に、HFPL-2の記述を手続きに変換するアルゴリズムを与える。これにより、属性の依存関係の解析などの管理的作業を属性の評価時以前に行なうことでき、効率的に属性を評価できる。次に、さらに評価効率を高めるためのいくつかの手法について、概説する。これによって、属性を効率的に評価するための手続きを生成する処理系を実現できる。

〖HFPL-2記述から手続き群への変換〗

モジュールＭの出力属性ａを評価するための手続きＰＭ,a は一般に次のような形式で定義される。

```
procedure PM,a(b1,...bm,a);
  if Cd1 then Bd1 else
  if Cd2 then Bd2 else
        ..............
  if Cdk then Bdk
```

ここで、{ b1, … ,bm }∈predM(a)、d1,…,dk はモジュールＭを左辺に持つ分割規則であり、Cd1,…,Cdkはそれらの分割条件である。Bdi,1<i<k は各分割規則に対応した手続き本体であり、意味規則EQdiを代入文により、子モジュールへの分割を手続き呼び出しにより表現した実行文から成るブロックである。これは拡張依存グラフDGdi* に基づいて次のアルゴリズムによって構成される。

```
program  fib ( );
module fib( input x:integer;
            output v:integer);
decomposition fib(x1,v1)=>
        fib(x2,v2),fib(x3,v3);
  when x1>1;
  with x2=x1-1;
       x3=x1-2;
       v1=v2+v3;
decomposition fib(x,v)=>null;
  when otherwise;
  with v=1;
end.
```

図2 HFPL-2によるフィボナッチ数を求めるプログラム

(1) 拡張依存グラフDGdi* において、親モジュールの出力属性生起M0.aはの依存関係がない属性生起と枝をDGdi* から除き、結果のグラフをDGdi* [M0.a]で表わす。

(2) DGdi* [M0.a]中の各属性生起ｖに対して、次のように属性評価のための実行文stm[v]を割当てる。

(2-1) v=M0.a または
v=Mk.b,1≦k≦n ,b IN[Mk] の場合
stm[v] : v:=fd,v(v1,...vr);

ここでfd,vは属性生起ｖに対する属性定義式であり、

DSd,v = { v1,....,vr }

である。

(2-2)v=Mk.b,1<k<m,b OUT[Mk]の場合
stm[v] : ＰMk,b(Mk.c1,..,Mk.cs,v);

ここで、predMk(b)={c1,...,cs} であり、ＰMk,bはモジュールMkの出力属性生起ｂを評価するための手続き呼び出しである。

(3) DGdi* [M0.a]中の属性生起をその依存関係に従ってトポロジカルソートする。その結果として

v1,v2,....,vq

なる順序関係が得られたとすると、手続き本体は

stm[v1],stm[v2],...,stm[vq]

と定義される。

以上のアルゴリズムに基づいて手続きＰM,a を構成することができるが、このアルゴリズムの適用できるクラスは手続きに与える引数を属性評価以前の段階で決定するという理由から、絶対非循環HFP に限られる。

これによって基本的には、HFPL-2で記

述されたプログラムをPascal等の手続き型プログラムへ変換することができるが、変換されたプログラムは、記憶域、実行時間の点からみて、十分に効率的であるとは、言えない。そこで次に挙げる効率化手法によって生成される手続きの効率化を計る。

〖属性の同時評価〗

前記のアルゴリズムでは、モジュールの出力属性生起ごとに1つの手続きを定義して、その手続き呼び出しにより、属性の評価を実現している。しかし、一般の意味規則においては、たとえば親モジュールの異なる出力属性が同一の子モジュールの出力属性に依存する場合があり、このようなときには冗長な手続き呼び出しを行なうことになる。したがって任意の計算木において同一モジューの出力属性が同時に評価できる場合には、それを1回の手続き呼び出しで求めることにより時間的な効率化を行なうことができる。

IO[M],OI[M] を使って属性が同時評価可能であるとは次のように定義できる。

(1) IO[M] とOI[M] の枝集合を合併して得られるグラフをC[M]とする。

C[M] = (v ,E)

ただし, v∈A[M], <a,b> ∈E ←→ <a,b> はIO[M] の枝またはOI[M] の枝

(2) $\mathbf{a} \subset OUT[M]$が同時評価可能

←→ ∀ v_1 , $v_2 \in \mathbf{a}$に対してv_1 と v_2 を結ぶC[M]中の道が存在しない。

モジュールM_0 における同時評価可能な出力属性集合$\mathbf{a}$を評価するための手続きP_0 $\mathbf{a}$は次のような形式で定義される。

```
procedure P0 a(b1 , … ,bm ,
                    a1 , … ,an );
  if Cd 1 then Bd 1 else
  if Cd 2 then Bd 2 else
        ……
  if Cd k then Bd k
```

ここで、 $a_i \in \mathbf{a}$, $1 \leqq i \leqq n$ 、{b1,… ,b_m }=$\cup_i$ predM(a_i) である。

手続き本体のブロックB_d i は、拡張依存グラフDGdi* を予め縮約しておくことにより、同様に構成することができる。

〖属性の大域化〗

上記の変換においては、各属性は手続き内の変数として実現されるため、手続き呼び出しが行なわれる度に、その値がスタックに積まれる。しかしたとえば、構文解析処理のためのHFP プログラムを考えてみてもsymbol tableの管理などのように計算木中を流れる属性は多数存在し、そのような属性は大域的な変数として実現する方が自然であり、処理効率の点からもそのほうが有利である。従ってHFPL-2では、記述の上からは大域的属性を許してはいないが、処理系において属性を変数に写像する時点で、大域的な変数として実現しても副作用が発生しないと保証されるものに限り、その変数を手続きの外にだすことによって処理の効率化を行なう。 一般に属性が大域化可能であるための条件は次の2つである。

1.大域化する属性集合のデータタイプは同一である。

2.大域化する属性の生存期間は重ならない。

この条件はデータフロー解析を用いてしらべることが可能であることが示されている が、HFPL-2の処理系では、分割記述のみから判定でき、かつ属性の評価順序に制約を加えることがないものをのみ大域化する。すなわち、属性集合$\mathbf{a}$が次の条件を満たすとき$\mathbf{a}$の各属性は共通の大域的な変数に写像することができる。

任意の分割 d に対して、

(1)DGd * ／$\mathbf{a}$が枝別れを含まない。

または、

(2)DGd * ／$\mathbf{a}$の枝に相当する依存関係が ' = ' である。

ただし、DGd＊／aは、拡張属性依存グラフDGd＊からのa中の属性の依存関係のない属性と枝とを除いたものである。(1)の条件は、属性の集合a中の属性の依存関係が任意の計算木において1本道であることを、(2)の条件は、a中の属性に変更が施されないことを示している。

〖 transnfer属性の記憶領域の共有化〗

分割d中の属性vについて、$DS_{d,v}$の属性の数が1つであり、かつ$f_{d,v}$がequal('=')であるとき、属性vをtransfer属性と呼ぶことにする。属性文法によって、コンパイラなどを記述した場合、このtransfer属性が、属性の半分以上を占めることが経験的に知られている。従ってこのような属性に対して、割り当てる記憶領域を共有化することは有用である。これは、次のような方法で実現することができる。

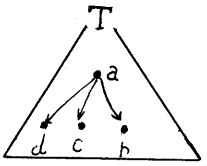
モジュールMoの出力属性aを求める手続きPMOaにおいて

(1) 手続きの仮引数はすべて call by refernce で行なう。

(2) 子モジュールの入力属性のうち、transfer属性については、手続き内のローカルな変数を割り当てず、仮引数で参照を行なう。

〖値によるデータアクセスから部分的変更によるアクセスへの変換〗

HFPでは、モジュール間の入出力の受け渡しはすべて属性定義式によって与えられ、手続きに変換したときには、一般には、手続き内のローカルな変数への値の代入として実現される。配列、リスト、ファイルなどの構造体に関しては、属性の一部分に対する変換が大部分であるにもかかわらず、評価時には新しい値を割り当てねばならない。しかし、属性がある条件を満たしているときは、その値をcall by reference で実現し、部分的な変更によるアクセスに変換することができる。すなわち、計算木T中で、

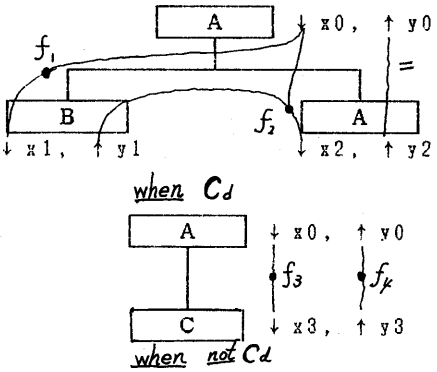


のように、1つの属性から複数個の属性が誕生している場合でも、トポロジカルソートの後に、属性bに関する実行文を最後に行うことができれば、属性aは、bとその記憶領域を共有しても、それによる副作用はおこらない。

［再帰構造の繰り返し構造への変換］

現在のHFPL-2では、子モジュールを繰り返し分割する機能を持っていない。そこで、繰り返して子モジュールを分割する場合には、再帰的に分割を行うことによって実現する。しかし処理効率の点からは、属性評価の時点では繰り返しによって評価した方が効率的である。そこでHFPL-2処理系では、ある種の、複雑な変換操作を行なわずに繰り返し構造に変換される右再帰的な分割は、手続きを生成する際に繰り返し構造に置き換えることにより、処理効率を高める。

たとえば、次のような構造の分割について、



ただし、 y i ,i=0..4 は同時評価可能な出力属性の集合であり、

x i = ∪k predM[v k], vk ∈ y i

このような分割に対しては、次のよう

な等価な繰り返し構造を持つ手続きを生成する。

```
procedure PAy0(x0,y0);
  var x1,y1,x3,y3,wx;
  begin wx:=x0; x1:=f1(x);
        PBy1(x1,y1);
        while Cd(wx) do
          begin wx:=f2(wx,y1);
                x1:=f1(wx);
                PBy1(x1,y1);
          end
        y3:=f3(wx); PCy(x,y3);
        y:=f4(y3);
  end
```

6．あとがき

本報告では、属性文法に基く階層的関数型計算モデルHFP の定義を与え、HFP の記述用言語HFPL-2の仕様を紹介した。HFPL-2は、実用上の立場からデータ構造、記述を考慮したものであり、汎用プログラミング言語として十分実用に絶えるものと信じる。さらに、処理効率を高めるためのいくつかの技法を示したが、これによって属性の評価をに効率的に行う処理系を作成することができる。

現在、我々はこれらの効率化を考慮したHFPL-2処理系を設計中である。この処理系自身も、HFPL-2を使って記述される。それによって、HFP の記述能力の確認、HFPL-2の拡張なども行なわれると考えている。

一方HFP には、非決定性HFP への拡張や、繰り返し分割記述の導入といった課題も残されている。

参考文献

[1] Knuth,D.E.: Semantics of Context free Languages, Math. syst. Th., 2, 127-145, 1968.

[2] Katayama, T.: HFP:A hierarchical functional programming based on Attribute Grammer, Proc. 5th Int. Conf. on Software Engineering, 1981.

[3] Katayama, T.: Translation of Attribute Grammer into procedures, Tech Rep. CS=K8001, Dept of Comp Sci., Tokyo Inst. of Tech., 1980.

[4] 武田　正之：プログラム言語の形式的意味記述と検証の研究，　東工大昭和５６年度博士論文．