計算機アーキテクチャ　63－14
（1986・11・21）

汎用パイプラインを簡約エンジンに用いたＦＰグラフリダクションマシン

A Graph Reduction Machine by Using a Pipelined Reduction Engine

高井　昌彰　Yoshiaki TAKAI
池部　優　Masaru IKEBE
伊波　通晴　Michiharu IHA
中村　維男　Tadao NAKAMURA
重井　芳治　Yoshiharu SHIGEI

東北大学　工学部
Tohoku Univ.

１．まえがき

近年、ソフトウェア生産性などの面から、関数型言語が注目されている。厳密な数学的基礎をもつ関数型言語は、プログラムの記述性が高く、簡潔な記述の内にアルゴリズムに内在する並列性を自然な形で表現できる。そのため関数型言語を指向した様々な並列処理計算機が研究されている(1)。関数型言語の代表的実行モデルとしては、データフローモデル、リダクションモデルが挙げられるが、言語モデルとの親和性の点からは、一般にリダクションモデルが秀れている。特にグラフリダクションでは、式がリスト構造で表現されるため、リダクションの進行に伴う式の多様な構造変化に柔軟に対応できる長所をもつ。

一方、リスト処理に内在する並列性の抽出を目的として、リスト処理指向汎用パイプラインを用いた構造データ並列処理の基本的手法が検討されている(2)(3)(4)。汎用パイプラインとは、パイプラインの各セグメントの処理機能を、理論的には、処理されるデータに付随するプログラムによって動的に決定し、価格性能比の良いパイプラインの汎用化を目指した処理システムである(5)(6)。リスト処理指向汎用パイプラインにおける構造データ処理の手法は、リスト操作が多用されるグラフリダクションシステムにも適用可能である。

そこで本論文では、関数型言語を直接実行するマシンの１つとして、先に述べたリスト処理指向汎用パイプラインを簡約エンジンに用いたFPグラフリダクションマシンを提案する。本処理システムのターゲット言語は、J.Backusによって提案された関数型言語FP(7)である。FPは構造データを１つのまとまったオブジェクトとして扱い、これに作用する関数および関数の組合せを表現する構成子(PFO)からプログラムが構成される。本処理システムでは、これらの関数とオブジェクトは全てグラフ（リスト構造）として表現され、FPの評価規則に従って、グラフリダクションが実行される。簡約可能な関数とオブジェクトは簡約エンジンに投入され、再帰的なリスト操作が汎用パイプラインで処理される。

本論文では、はじめにFPプログラムのグラフリダクションによる実行の基本原理について述べる。続いて、文献(3)(4)では明確にされていなかったパイプライン処理可能な再帰関数のクラス定義の一方法を検討し、再帰的リスト処理関数のパイプライン実行における基本手法を述べる。次に、FPにおける先行評価および遅延評価の実現を可能とする、簡約実行の具体的制御方法を明かにする。

２．関数型言語ＦＰのグラフリダクション

２．１　ＦＰシステム

FPは、J.Backusによって提案された合成型関数型言語の１つである(8)。FPシステムは、オブジェクトの集合、関数の集合、作用(application)、関数型式(functional form)の集合、および関数定義の集合から構成される。

オブジェクトは、与えられたアトムの集合Ａの要素から構成され、Lispのリストと同様に、再帰的な構造データを表現する。すなわち、オブジェクトの集合をＯとすれば、

$$A \subset O \qquad (1)$$

$$\langle x_1, x_2, \ldots, x_n \rangle \in O \quad \text{if} \quad x_i \in O,\ 1 \leq i \leq n \qquad (2)$$

関数は、これらのオブジェクトをオブジェクトに写像する。関数は、原始関数(primitive)であるか、関数形式であるか、あるいは定義されたものである。原始関数の集合には、オブジェクトに対する構造操作関数、算術関数、および述語関数が含まれる。FPの関数はすべて１引数である。

FPシステムの唯一の演算は作用である。ｆを関数、ｘをオブジェクトとするとき、ｆ：ｘは作用であり、その結果は新しい１つのオブジェクトである。

関数型式は、関数を表現するまとまった１つの式であり、いくつかの関数をプログラム構成子(PFO, Program Forming Operator)を用いて結合することで表わされる(付録参照)。すなわちPFOは、関数を関数に写像する高階関数の一種と考えられる。

２．２　ＦＰの評価規則

FPの形式的な意味は、FFP(Formal system for FP)によって与えられる(7)。FFPシステムでは、与えられたアトムの集合の要素から式(expression)の集合Ｅが次のように構成される。

$$A \subset E \qquad (3)$$

$$\langle x_1, x_2, \ldots, x_n \rangle \in E \quad \text{if } x_i \in E,\ 1 \leq i \leq n \qquad (4)$$

$$(x : y) \in E \quad \text{if } x, y \in E \qquad (5)$$

FFPでは、FPにおいて明確に区別されていたデータとしてのオブジェクトとプログラムとしての関数は、式という１つの集合で扱われる。したがって、FFPのアトムの集合には、FPにおけるアトムの集合の他に、原始関数やPFOを表わすアトムが含まれる。FFPにおけるオブジェクトの集合Ｏとは、作用(f:x)を部分式として持たない式の部分集合である。

FFPの任意の式ｅ∈Ｅは、意味μ(e)∈Ｏを持つ。この意味は、式ｅの最内側の作用を、それぞれの意味に置き換えることを繰返して見出される。最内側の作用(x : y) (x,y∈Ｏ)の意味は、ｘによって表現される関数をｙに作用させた結果である。オブジェクトと、それが表現する関数の対応関係は、表現関数ρによって与えられる。すなわち、FFPシステムは、次の意味関数μと表現関数ρによって記述される。

$$\mu \in [E \to O], \quad \rho \in [E \to [E \to E]] \qquad (6)$$

２．３　ＦＰグラフリダクション

関数型言語の実行モデルの１つとして、リダクションモデルがある。リダクションモデルでは、プログラムとデータを式の構成要素として同一視し、すべての計算が式の書き替えによって実現される。すなわち計算は、次の２つの基本ステップの繰返しによって実現される。

Step 1. 書き替え可能な式(リデックス)を見付ける。

Step 2. 書き替え規則に従って、式の書き替えを実行する。

このような計算のリダクションモデルは、一般に次の利点をもつことが指摘されている(9)。

(1)関数型言語とのセマンティックギャップが少ない。

関数型言語における計算の概念は、その数学的基礎となっているラムダ算法やコンビネータからも明らかなように、式の簡約(reduction)によって定義されている。そのため、言語モデルと実行モデルの間の意味的な隔りが少なく、言語に含まれる諸概念を容易に実現できる。

(2)並列処理向きである。

複数のリデックスの書き替えは、相互作用のない限り、独立に実行できる。また、与えられた式が正規型をもつならば、式の書き替え順序によらず同一の結果が得られることが数学的に保証されている(Church-Rosser性)。そのため、複数の処理要素が非同期に動作する並列処理システムに適している。

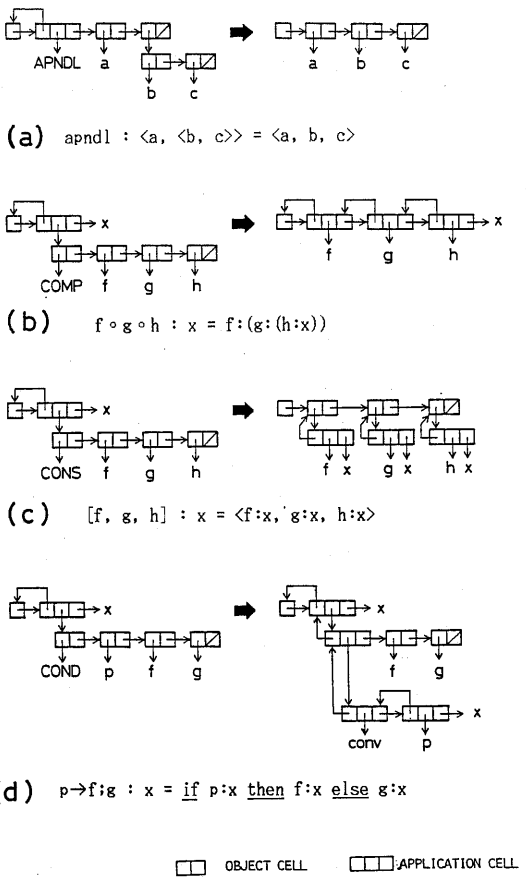

図１．基本的なＦＰのグラフ表現と書き替え規則

2.2節で述べたように、FPの形式的意味は、関数とオブジェクトをまったく同等に取り扱うFFPシステムにおける式の簡約によって与えられている。そこで、FPのプログラムをセルを用いたグラフ(リスト構造)の形(10)で表わし、さらにそのグラフに対する書き替え規則を定義することで、FPシステムをリダクションモデルによって実現する。

FPプログラムの構造を表現する基礎となるセルは、オブジェクトセルと作用セルの２種類に大別される(図６参照)。オブジェクトセルは、CARフィールドとCDRフィールドから成り、FPのオブジェクトまたは関数のリスト構造を構成する。一方、作用セルはオブジェクトと関数の作用部分を表わし、RET(return)フィールド、FUN(function)フィールド、OBJ(object)フィールドを基本的構成フィールドとする。FUNフィールドとOBJフィールドは、作用を構成する関数とオブジェクトへのポインタをそれぞれ含んでいる。また、RETフィールドは作用の結果を返すセルへのポインタを含んでい

る。グラフに対する書き替え操作の内容は、作用セルのFUNフィールドが示すオブジェクト(原始関数、またはPFO)に依存する。

図1に、最も基本的なFPプログラムのグラフ表現と、それぞれのグラフの書き替え規則を示す。任意のFPプログラムは、これらを繰返し適用することで、1つのグラフ構造に変換できる。グラフの書き替えを大別すると、原始関数に対する変換(図1.(a))と、PFOに対する変換(図1.(b)(c)(d))に分類される。これらは、グラフを書き替えて簡約を実行するという点では同じであるが、簡約の実行に伴う作用セルの増減が異なっている。原始関数を評価した場合、必ず作用セルが1個減少するのに対し、PFOを評価した場合では、一般に作用セルが増加する。これはPFOが高階関数であることによる。新に生成された作用セルによって示される関数とオブジェクトは再び評価され、最終的に作用セルがなくなったとき、プログラムの実行が停止する。例として、内積計算を行なうFPプログラム((7)式)のグラフリダクションの過程を図2に示す。

$$(/+)\circ(\alpha\times)\circ trans : \langle\langle 1,2,3\rangle,\langle 4,5,6\rangle\rangle \qquad (7)$$

ただし、+, ×, trans は原始関数、/, α, ∘はPFOである。ここで、グラフに含まれる全ての作用セルは、必ずしもリデックスを表わしていないことを注意しておく。作用セルの評価の順序および制御方法については、5章で述べる。

2.4 明示的な遅延評価と先行評価の導入

はじめに、自然数の無限リストを生成する、次のような線形再帰関数を考える。

$$Def \quad int \equiv apndl\circ[id,\ int\circ add1] \qquad (8)$$

この関数を含む作用対(int:1)を、2.2節で示したFPの通常の評価規則に従って評価することは、次の2つの理由で意味がない。

(1) 終了条件がないため、無限の作用対を生成し、簡約が停止しない。

(2) apndlが最後に評価される(最内側の作用から評価される)ため、無限に続く簡約の途中結果を参照することもできない。

したがって、(8)式に意味を持たせるためには、必要な時だけ作用対の評価を進める遅延評価[11]と、必要な関数を先に評価する先行評価を取り入れる必要がある。そこでFPに次の2つのPFOを追加する。これらは評価システムに対して、それぞれ特別な評価方法を明示的に指示するものである。すなわち、システムにおける評価順序の決定の一部は、直接プログラマに委ねられる[12]。

(a) β 〈遅延評価PFO〉

$\beta f:x = f:x$, ただし作用f:xの評価は、その結果が要求されるまで遅延される。(グラフの書き替えは、図3(d)(e)を参照)

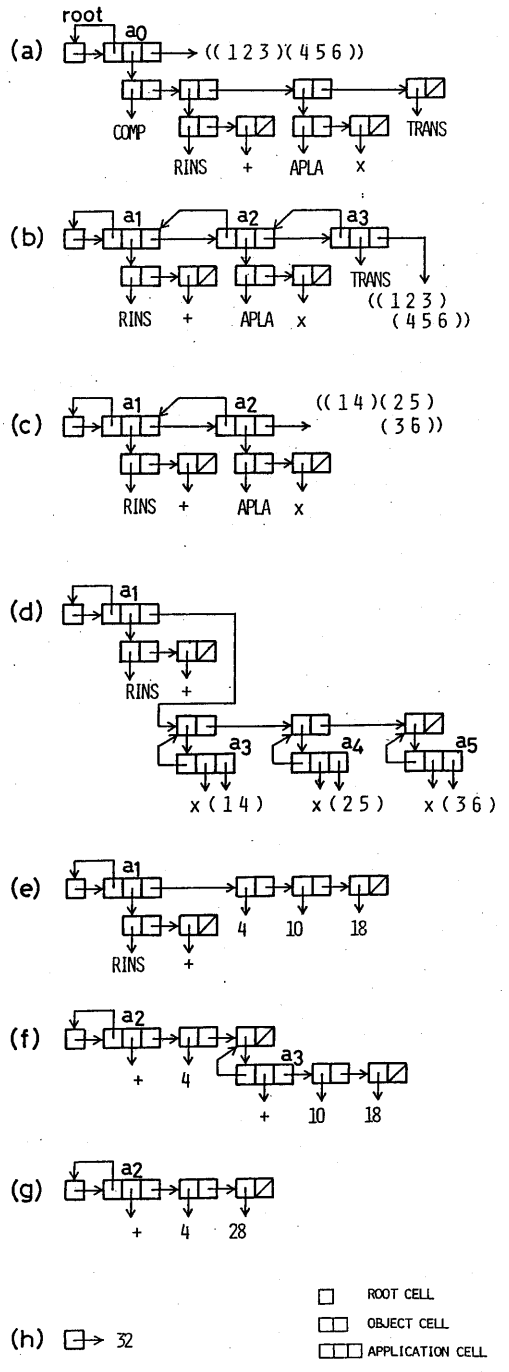

図2. 内積計算のグラフリダクションの過程

(b) • ＜先行Composition＞

f•g:x = f:(g:x), ただしfの評価はgの評価の結果を待たずに起動される。(グラフの書き替えは通常のCompositionと同じである。)

これらのPFOを用いると、先の関数intは次のように定義できる。

Def int ≡ apndl•[id, βint∘add1] (9)

図3は、こうして定義されたint関数を用いた作用対(int:1)のリスト構造と、その簡約の過程である。ここでアトムPCOMPとBETAはそれぞれ"•"、"β"を表わす。また、再帰的に定義された関数はポインタのループによって表現される。

先行Compositionに対するグラフの書き替えによって生じた新しい作用セルは、それらの間に依存関係がないものとして独立に評価されることをシステムに要求する。この結果、apndlが先行評価され、関数intが生成したオブジェクト(途中結果)は、他の関数によって直ちに参照することが可能となり(図3(c)(d))、容易にStreamを実現できる。またβの書き替えによって生じた作用セルは、その評価が強制的に遅延されることをシステムに要求する(図3(d)(e))。これらの評価の制御方法については、5章で述べる。

3. リスト処理再帰関数のパイプライン処理

3.1 リストのパイプライン処理の可能性

一般にリスト構造は、ポインタで結合されたセルから構成される。したがって、関数がリスト構造を操作する場合、各要素を参照するために、リストをたどるという逐次性が必然的に存在している。このリストのたどりは、通常再帰関数の関数呼出しによって表現され、再帰の各レベルでリストの各要素がそれぞれ参照され、処理が実行される。したがって、このような再帰関数をパイプライン上に展開し、リストのたどりと各要素に対する処理をオーバラップできれば、リスト処理の多重性を引き出すことができる[3][4]。この多重性の効果は、再帰の深さと、各再帰レベルでの処理に依存する。

しかしながら、処理要素が単純に縦続接続されたパイプラインアーキテクチャでは処理要素間の情報の流れが一方向であるという制約条件から、すべての再帰構造を直接パイプラインにマッピングして処理を行なうことは困難である。すなわち、パイプライン処理の対象となる再帰関数は、ある特定の条件を満たすクラスに限定される。

3.2 畳み込み可能線形再帰関数

パイプラインに直接マッピング可能な再帰関数のクラスを定義するために、まずFPシステムにおける次のような再帰定理を考える。Backusによるこの定理の証明は、文献(7)に見られる。

<再帰定理>

f ≡ p→g ; h∘[i, f∘j] (10)

ただし、p,g,h,i,jは任意の関数である。

このとき、関数fは次のように無限展開できる。

f≡p→g; p∘j→Q(g); ...; p∘j^n→Q^n(g);... (11)

ここで、

$Q^n(g) \equiv /h \circ [i,\ i\circ j,\ \ldots,\ i\circ j^{n-1},\ g\circ j^n]$

$\equiv h\circ[i,\ h\circ[i\circ j,\ h\circ[i\circ j^2,\ \ldots$

$\ldots,\ h\circ[i\circ j^{n-1},\ g\circ j^n]\ \ldots$

$\ldots]]]$ (12)

線形再帰関数fの評価は、関数i, jおよびgによる展開過程(unfolding)と、関数hによる縮退過程(folding)から構成される。(12)式から明らかなように、この2つの過程は逐次的に行なわれる必要がある。すなわち、縮退過程は展開過程が完了したのち、展開の方向と逆方向に進行する。

次に、新しいPFOとして、Left Insertを導入する。

Left Insert "\", (左挿入)

$\backslash f : \langle x_1,\ldots,x_n\rangle = f:\langle \backslash f:\langle x_1,\ldots,x_{n-1}\rangle, x_n\rangle$ (13)

この定義から明かに次の関係が成立する。

$\backslash f\circ[g_1,\ldots,g_n] \equiv f\circ[\backslash f\circ[g_1,\ldots g_{n-1}],\ g_n]$ (14)

ここで、(10)式の線形再帰関数fの縮退過程を構成する関数hに対して、

$/h \equiv \backslash h$ (15)

の関係が成立すると仮定する。このとき(12)式は、(14)式の関係を用いて、

$Q^n(g) \equiv \backslash h\circ[i,\ i\circ j,\ \ldots,\ i\circ j^{n-1},\ g\circ j^n]$

$\equiv h\circ[h\circ[h\circ[\ldots$

$\ldots\ h\circ[h\circ[i, i\circ j],\ i\circ j^2]\ \ldots$

$\ldots], i\circ j^{n-1}],\ g\circ j^n]$ (16)

となる。この場合、線形再帰関数fの縮退過程は展開過程の終了を待たずに、展開過程と同一方向に同時進行する。そこで、(15)式の関係を満足する線形再帰関数を"畳み込み可能"であると呼ぶ。(15)式の条件を満たすFPの原始関数としては、+, ×, and, or, catなどがある。これらの関数を、それが作用するオブジェクト内部の実質的な引数の集合Dにおける2項演算子"⊕"と見なした場合、(D,⊕)は半群(semigroup)を形成する。特にcatは、FPのオブジェクトを構築するための基本的かつ重要な構造操作関数である。

$cat : \langle\langle x_1,\ldots,x_n\rangle,\langle y_1,\ldots,y_m\rangle\rangle$

$= \langle x_1,\ldots,x_n,y_1,\ldots,y_m\rangle$ (17)

3.3 パイプラインへのマッピング

複数の処理要素が縦続接続されたパイプラインにおける処理の進行は、2入力-2出力の記憶をもたないプロセスノードをアークで結合したフローグラフ(図4)を用いて記述できる。同図で、ξ_1軸方向のアークは、パイプラインの同一処

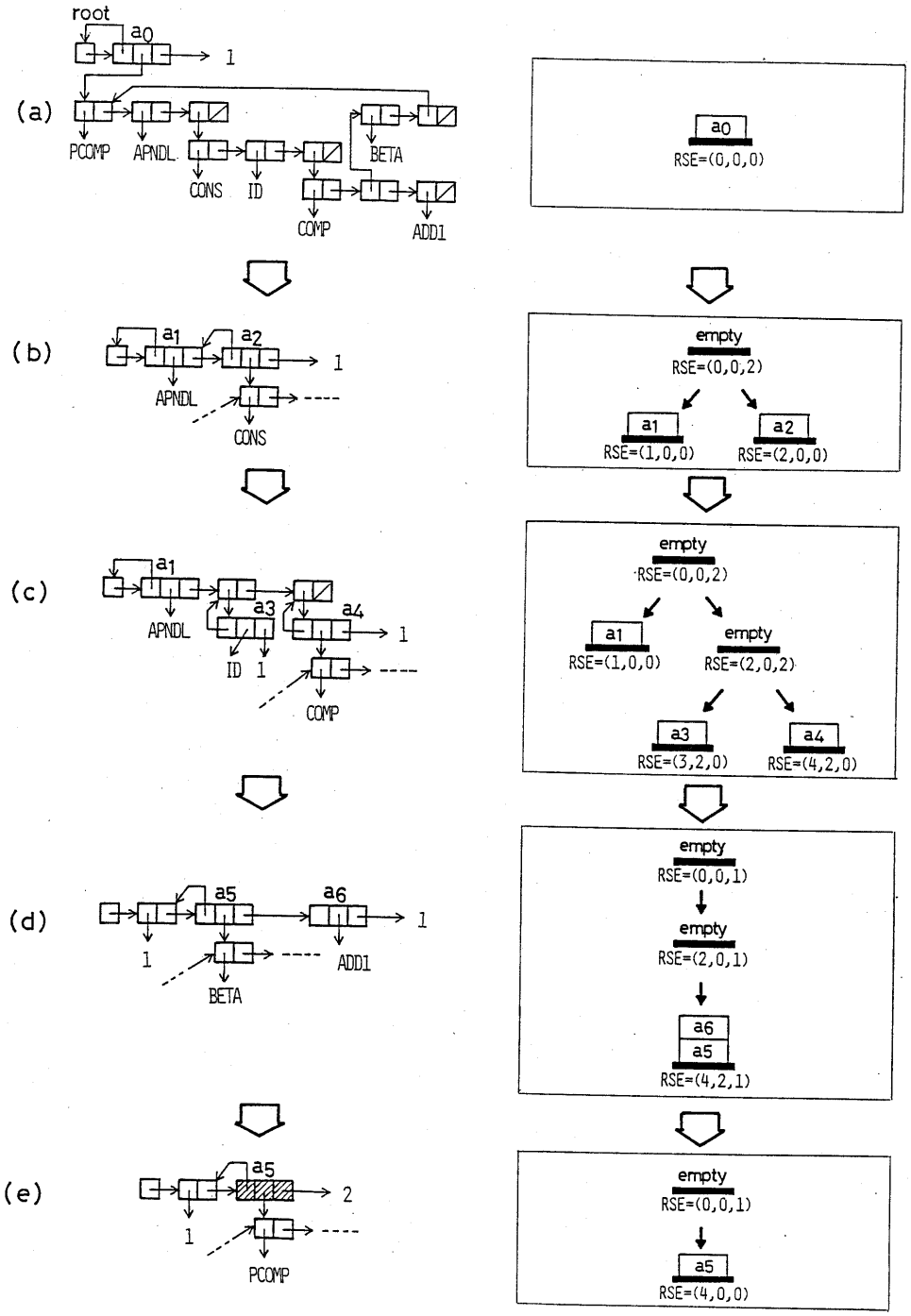

図3．先行評価および遅延評価の簡約実行制御過程

(左:グラフの書き替え、右:簡約スタックの内容)

理要素内のプロセスの、履歴を用いた通信を表わし、ξ_2軸方向のアークはパイプライン上で隣接する処理要素に存在するプロセス間の通信を表わす。したがって、各プロセスノードの処理が同期していると仮定すれば、このフローグラフ上における時間の進行は、$\xi_1+\xi_2$=一定の同時刻直線を用いて表わされる。

次に、畳み込み可能関数f≡p→g；h∘[i,f∘j]を考える。ただし、fを構成する部分関数p,g,h,i,jは、それぞれプロセスノードにあらかじめ組み込まれているものとする。このとき関数fは、図4に示すようにフローグラフ上にマッピングすることができる。フローグラフの自由度(独立な入力/出力のアーク数)は2であるから、再帰の展開と縮退の方向はξ_1軸方向(図4(b))またはξ_2軸方向(図4(a))のいづれかである。

一般には、畳み込み可能関数の部分関数が、また畳み込み可能である場合がある。そのような場合、上で述べたマッピングが部分関数に対して繰返し適用される。ただし、実際的な処理の効率を考慮した場合には、プロセスの粒度と並列性のトレードオフを考慮して、いくつかのプロセスノードがまとめられる。

3．4　畳み込み可能なリスト処理再帰関数

FPのグラフリダクションにおいて、オブジェクトに対する関数の作用は、すべてリストの構造操作に帰着する。このリスト操作は、原始関数またはPFOの性質に依存するが、次の2つの型に代表される線形再帰関数で表現できる場合が極めて多い。

Type 1.　f_1=null→[]；apndl∘[g∘1, f_1∘tail]　(18)

Type 2.　f_2=null→[]；apndr∘[f_2∘tail, g∘1]　(19)

これらの再帰関数の縮退過程は、apndlまたはapndrによる線形リストの構築であるが、これらの関数は3.2節で述べたcatと次の関係にある。

apndl∘[f, g]≡cat∘[[f], g]　(20)

apndr∘[f, g]≡cat∘[f, [g]]　(21)

したがって、Type 1.、Type 2.ともに畳み込み可能であり、パイプラインにマッピングすることが論理的に可能である。

関数apndl、apndrによるリストの構築は、実際のシステムにおいては、CARおよびCDRフィールドをもつセルの集合に対する次の3つの原始的操作の組合せによって実現される。(これらの操作は、作用セルの場合においても、原理的には同じである。)

(1) get_cell()
　新しくセルを用意し、そのアドレスを返す。

(2) write_car(address, value)
　指定されたセル(address)のCARフィールドに、指定された値(value)を書き込む。

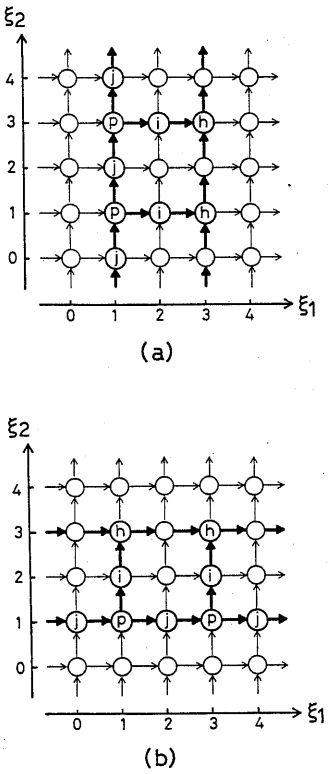

図4．パイプラインのプロセスフローグラフへの畳み込み可能線形再帰関数のマッピング

(3) write_cdr(address, value)
　指定されたセル(address)のCDRフィールドに、指定された値(value)を書き込む。

通常、Lisp等におけるcons命令は、これらの原始的操作を次のように組合せて実現される。

```
normal_cons(x, y)
{
    c=get_cell();
    write_car(c, x); write_cdr(c, y);
    return(c);
}
```

この方式は、Type 2.の再帰関数におけるapndrの実現に適用できる。しかし、Type 1.の場合では、セルのCDRフィールドの値が次の再帰レベルで決定されるため、上の方式でリストを構築することは不可能である。　このような場合、　CDRフィールドへの書き込みを次の再帰レベルで実行させる、"Pipeline cons"を用いることによって、　Type 1.の再帰におけるapndlを実現することができる(具体的なマッピングの例は、図8，9を参照)。

```
pipeline_cons(x, adr)
{
        c=get_cell();
        write_car(c, x); write_cdr(adr, c);
        return(c);
}
```

このPipeline consは、本来データフローマシンにおいてリストの生成側の関数と消費側の関数の間での並行処理を実現する"Lenient cons"[13]を、一方向の情報の流れしか持たないパイプライン上で実現したものである。Type 1.のapndlの実現方法としては、この他に、連想記憶(CAM)を用いた方法もある[14]

4．ＦＰグラフリダクションマシンのシステム構成

次に、本論文で提案するFPグラフリダクションマシンのシステム構成について述べる。図５は本システムの基本アーキテクチャである。本システムは大きく３つの部分、すなわちSM(構造体メモリ)，RE(簡約エンジン)，およびRC(簡約コントローラ)から構成される。以下、各部の機能を簡単に説明する。

構造体メモリ（SM）は、簡約の対象であるFPプログラムのグラフ表現をリスト構造の形で保持するメモリである。さらにリスト構造を効率的に処理するために、REから転送されるリスト操作命令を解釈し、構造データに対して原始的な操作を実行する。SMのセル領域は大きくオブジェクトセル領域と作用セル領域に分けられる。各セルの構成を図６に示す。ここで、セルの各フィールドの内容は、2.3節で述べた通りである。ただし、各フィールドは属性tagおよびセルで同期をとるためのReady_tagを持つ。あるフィールドのポインタがオブジェクトセルを指す場合、Ready_tagはONである。それ以外の場合には、Ready_tagはOFFである。また、作用セルのLAZフィールドは遅延評価のために使用される。SMは論理的には１つであるが、REからのアクセス競合を軽減するために多バンク構成されている。

簡約エンジン（RE）は、RCから作用セルのポインタを受け取り、それによって示されるグラフに対して書き替え規則を適用し、簡約を実行する。REは、多段に縦続接続された複数のPEからなる汎用パイプラインによって構成される。また、PSCはパイプラインのスケジューラである。すべてのPEはネットワークを介してSMの各バンクと結合されている。

PEの内部構成を図７に示す。PEはインタプリティングユニット(IU)、メモリアクセスユニット(MAU)、リンケージユニット(LU)、演算ユニット(FU)、入力コントローラ(IC)、出力コントローラ(OC)から構成される[3]。IUは、ICからの入力とLU内の履歴から、プロセスを決定(デコード処理)し、セグメントの他の構成要素を制御して処理を実行する。原始関数

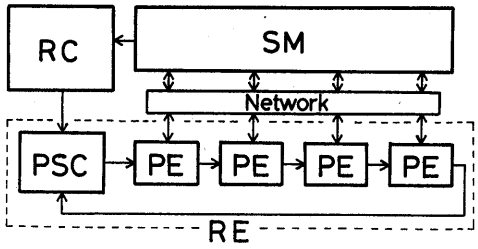

図５．ＦＰグラフリダクションシステムの構成

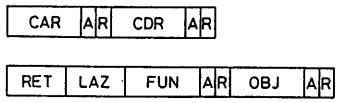

図６．セルのフィールド構成
（上：オブジェクトセル、下：作用セル）

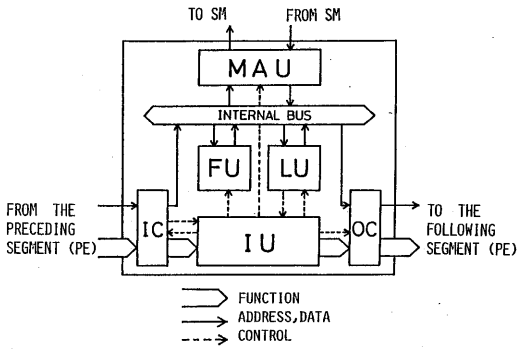

図７．ＰＥの内部構成

およびPFOの書き替え規則は、すべてマイクロプログラムとしてIUに組み込まれている。IUは１つのプロセスの処理の最後に、同一PE内の次のプロセスに処理を引き継ぐため、部分処理された構造データのポインタまたはデータをLUに履歴として残し、また、次段のPEに対しても、同様な情報をOCから転送する。

簡約コントローラ（RC）は、REによるPFOの簡約によって新にSM内部に生じた作用セルを管理し、簡約可能となった作用セルのポインタをREに供給する。簡約コントローラの動作の詳細については次章で述べる。

また、本リダクションシステムの評価用として、68000密結合マルチプロセッサシステムによるハードウェアシミュレータを現在開発中である[15]

5．グラフリダクションの実行制御

5．1　作用セルの管理方法

2.3節で述べたように、FPプログラムのグラフリダクション実行過程において、原始関数の評価はグラフに含まれる作用セルの数を減少させ、PFOの評価は作用セルを逆に増加させる。したがって、SM内部での作用セルの動的な生成・消滅を集中的に監視することによって、グラフリダクションの進行を容易に制御することが可能である。

簡約の結果生成される作用セル間の依存関係に基づいて、PFO(ただし、遅延評価用のβを除く)を次の2つに分類することができる。

(1) 逐次型PFO：

Composition, Right Insert,
Left Insert, Condition, etc.

生成された作用セルの間に依存関係があり、新しい作用セルの簡約が逐次的にしか実行できないもの。

(2) 並列型PFO：

先行Compositon, Construction,
Apply_to_all etc.

生成された作用セルの間に依存関係がなく、新しい作用セルの簡約を並列に実行できるもの。

したがって、次に簡約する作用セルを決定するためには、SM内部で生成される作用セルが、どの型のPFOによって生じたものかを判断して、作用セル間の依存関係に基づく動的なチェーンを構築すればよい。本システムでは、逐次型PFOの簡約に伴って生じる作用セルはスタックを用いて、また並列型PFOの簡約では、複数のスタックを動的に生成し、これらをポインタで結合することによって作用セルを管理する。

5.2　簡約コントローラ

簡約コントローラによるグラフリダクションの実行制御は、動的に生成・消滅する複数の簡約スタックとそれらを統一的に管理するためのテーブルを用いて行なわれる。簡約スタックは、その生成時に割当てられた番号(id)によって識別・管理される。簡約スタックには、SM内部で生成された作用セルのポインタが格納される。プログラムの実行中に動的に生成されるすべての作用セルは必ず1つの簡約スタックに属する。簡約スタックから作用セルのアドレスがpopされる場合、遅延評価に備えて、スタックトップはテンポラリレジスタに退避される。また、簡約スタック管理テーブルのエントリは、簡約スタックのidと、その簡約スタックを生成した親の簡約スタックのid (pid)、および簡約スタックの状態を表わすウエイトカウンタ(wc)の3項組

RSE=(id, pid, wc)　　(22)

から構成される。

RCからREへの出力は、次の2項組RCOである。

RCO = (RSID, ACADR)　　(23)

ここでACADRは簡約する作用セルのアドレスであり、RSIDはその作用セルのポインタが属している簡約スタックのidである。一方、SMからRCへの入力は、次の4項組RCIである。

RCI = (RSID, RST, FTYP, ACADR)　　(24)

ここでRSIDは簡約スタックのid, ACADRは簡約の過程でSM内部に生じた作用セルのアドレスである。また、RSTはREにおける現在の簡約状態(次のいずれか)である。

fin ： 簡約が完了した
con ： 簡約を継続中である
laz ： 評価の遅延された作用セルに到達した

FTYPは簡約している関数の型(次のいずれか)である。

prim ： 原始関数
s_pfo ： 逐次型PFO
p_pfo ： 並列型PFO
laz ： 遅延評価PFO(β)

5.3　簡約スタックのチェーンによる簡約実行制御

次に、簡約スタックのチェーンによるグラフリダクションの実行制御の詳細を述べる(適時、図3を参照)。

(1)簡約の開始

簡約を開始する準備として、id=0の簡約スタックを生成する。次に、これをエントリRSE=(#0,#0,#0)としてスタック管理テーブルに登録し、初めの唯一の作用セルのアドレス(a_0)をこのスタックにpushする(図3(a))。簡約は、id=0の簡約スタックをpopし、RCO=(#0, a_0)をREに供給することで開始する。

(2)原始関数およびPFOの簡約制御

SMからの入力RCIに依存して、次のような動作を行なう。

(a) RCI=(#k, fin, prim, ---)の場合
id=kの簡約スタックトップa_{top}をpopして、RCO=(#k, a_{top})をREへ出力する。

(b-1) RCI=(#k, con, s_pfo, a_i)の場合
id=kの簡約スタックにa_iをpushする。

(b-2) RCI=(#k, fin, s_pfo, ---)の場合
id=kの簡約スタックトップa_{top}をpopして、RCO=(#k, a_{top})をREへ出力する。

(c-1) RCI=(#k, con, p_pfo, a_i)の場合
新しい簡約スタック(id=n)を生成し、エントリRSE=(#n, #k, #0)としてテーブルに登録する。その後、RCO=(#n, a_i)をREに出力する。一方、id=kの親スタックは、RSEのウエイトカウンタ(wc)をインクリメント(+1)する(図3(b),(c))。

(c-2) RCI=(#k, fin, p_pfo, ---)の場合
id=kの簡約スタックは、wc=0になるまで待機状態(子スタックからの復帰待ち状態)を維持する。

(d-1) RCI=(#k, fin, laz, ---)の場合

id=kの簡約スタックは、テンポラリレジスタに退避していた作用セルのアドレスをpushして、要求があるまで評価遅延状態を維持する。

(d-2) RCI=(#m, laz, ---, ---)の場合

評価遅延状態にあるid=mの簡約スタックで、遅延されていた簡約を再開する。すなわちid=mの簡約スタックトップa_{top}をpopして、RCO=(#m, a_{top})をREへ出力する。

(3)子スタックの消去

簡約スタックトップが空になると、テーブルを参照して、親スタックのwcをデクリメント(-1)する。スタックの空になった子スタックのエントリはテーブルから除去される。

(4)親スタックの再起動

待機状態の簡約スタック(id=k)でwc=0になると、スタックトップa_{top}をpopして、RCO=(#k, a_{top})をREへ出力する。

(5)簡約の終了

id=0の簡約スタックのスタックトップが空であれば、簡約は終了である。

(6)意味のない簡約スタックの除去

待機状態の簡約スタック(id=a≠0)で、スタックトップが空である場合、その簡約スタックのエントリは、次の手順に従ってテーブルから除去できる。ただしid=xのエントリのpidとwcを、それぞれ、pid(x),wc(x)で表わす。

Step 1. ∀x : pid(x)=aに対して、pid(x) ← pid(a)

Step 2. wc(pid(a)) ← wc(a) + wc(pid(a)) - 1

Step 3. id=aのエントリを消去

この管理テーブルの操作により、無限のデータ構造を先行評価で生成する場合に生じる空の簡約スタックのチェーンを容易に除去することができる(図3(d)のRSE=(2,0,1)のスタックは除去可能)。

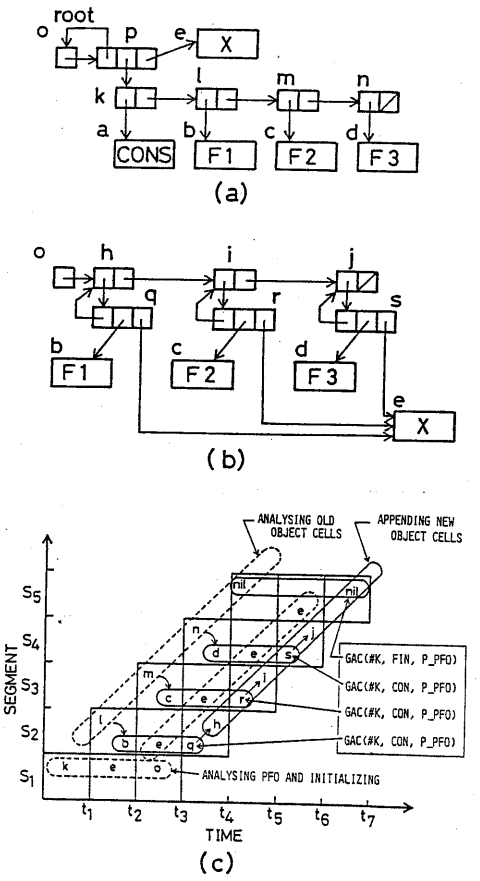

図8．並列型PFO, Constructionの簡約実行
((a):簡約前のグラフ、(b):簡約後のグラフ、
(c):パイプラインの時間空間図へのマッピング)

6．簡約エンジンにおけるリダクションの実行

6．1 簡約エンジン

REはSMに対して原始的なリスト操作命令を出力し、グラフの書き替えを実行する。このグラフ書き替えは、3.4節で述べたように、ほとんどの場合、線形再帰的なリスト操作であり、多段に縦続接続されたPEから構成される汎用パイプライン上に、動的に展開されて実行される。

簡約の途中で新たに生じた作用セルは、すべて簡約コントローラによって管理される必要がある。また、簡約の終了を簡約コントローラに伝えることも必要である。これらはすべてSMに対する次のようなgac(Get Application Cell)命令によって達成される。

gac(RSID, RST, FTYP) (25)

gac命令の3つの引数は、それぞれ、簡約スタックのid、簡約の状態、および関数の型を表わしている。この命令がREからSMに送られ、SM内部で実行されると、RST=conの場合には新しい作用セルがSM内部に生成され、そのアドレスがREに返される。RSTがそれ以外の場合には作用セルは生成されない。一方、SMからRCには、gac命令の3引数に、新しく生成された作用セルのアドレスを加えてRCI=(RSID, RST, FTYP, ACADR)を構成し、RCに供給する。

６．２　簡約の実行

次に、簡約される関数の型(FTYP)別に、簡約エンジンの動作を簡単に述べる。

(1)原始関数が簡約された場合

原始関数の簡約に伴うgac命令の発生は、作用セルのRETフィールドが示すセルへの書き込みが行なわれた時点で、gac(#k, fin, prim)が発生する。これにより、Implicitな先行評価が実現される。この際、簡約実行の同期はセルのReady_tagを参照することで行なわれる。すなわち、ある作用セルの簡約実行中に、Ready_tagがOFFのオブジェクトセルを参照したならば、その簡約は一時中断され、RE内部で待ち状態となる。Ready_tagがONになり次第、簡約が再開される。これらの制御は、すべてRE内部のスケジューラ(PSC)によってなされる。本論文では、PSCによる簡約実行のスケジューリングの詳細については言及しない。

(2)PFOが簡約された場合

新しい作用セルの生成が要求される時点で、gac(#k, con, p/s_pfo)が発生する。また、新しい作用セルが必要でないと判定された時点で、gac(#k, fin, p/s_pfo)が発生する。例として、並列型PFOの一つであるConstructionの簡約に伴う、gac命令の発生を、パイプラインの時間空間図を用いて図８(c)に示す。ここで、図中のアルファベットは、図８(a),(b)のリスト構造を構成するセルのアドレスである。また、逐次型PFOの一つであるCompositionの簡約に伴うgac命令の発生を、同様に、図９に示す。本システムでは、再帰的なグラフの書き替え操作がパイプライン上に展開されて処理されるため、PFOの簡約に伴うgac命令は、PFOの型によらず、逐次的に発生する。このことは、RCにおける作用セルの管理に、基本的にスタックを用いる理由の１つである。

(3)βが簡約された場合

作用セルのLAZフィールドに簡約スタックのid=kを書き込み、RETフィールドの示すセルのReady_tagを強制的にONにした後、gac(#k, fin, laz)を発生する。

(4)評価の遅延された作用セルに遭遇した場合

評価の遅延されている作用セルのLAZフィールドの値(#m)を参照して、gac(#m, laz, xxx)を発生する。この結果、評価の遅延されていた簡約が再開され、作用セルに代わって、新しいオブジェクトセルが生成される。この間、簡約実行の要求側は、RE内部で待ち状態となる。

７．むすび

リダクションマシンは、言語モデルとの親和性や並列処理の容易性などの点から、関数型言語の実行マシンとして有望である。本論文では、BackusのFPをターゲット言語とするリダクションマシンを提案し、その基本原理と実行制御の方法について述べた。本システムは、FPプログラムのグラフリダクションと、そのパイプライン実行によって特徴付けられる。

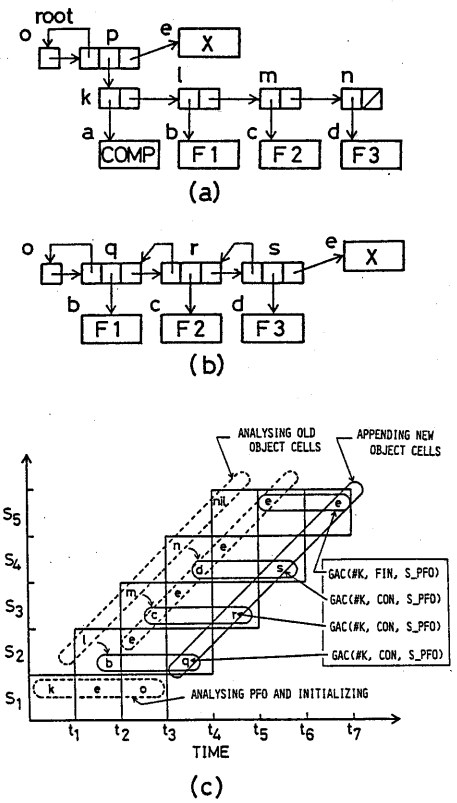

図９．逐次型PFO, Compositionの簡約実行
((a):簡約前のグラフ、(b):簡約後のグラフ、
(c):パイプラインの時間空間図へのマッピング)

本システムにおけるグラフリダクションの実行制御は、動的に生成・消滅する複数の簡約スタックとそれを管理するテーブルを用いて構成される作用セルの動的なチェーンによって達成される。これにより、FPに明示的な先行評価や遅延評価を導入することができ、無限のデータ構造やストリームをFPの中で容易に取り扱うことが可能となる。一方、パイプラインは一般に価格性能比の秀れたアーキテクチャであるが、その構造上、直接処理できる関数は、畳み込み可能な線形再帰関数に限定される。簡約エンジンの性能を十分に引き出すためには、直接パイプライン処理できるリデックスの粒度を大きくするためのプログラム変換が必要である。

今後、プログラムの最適化、ならびに簡約エンジン内部における簡約実行のスケジューリング等を考慮した上で、システム全体の性能評価を行う予定である。

参　考　文　献

(1) S.R.Vegdahl : "A Survey of Proposed Architectures for Execution of Functional Languages", IEEE Trans. Comput., vol. c-33, No.12, (1984).

(2) T.Nakamura, K.Sakai and Y.Mishina : " Function Level Computing on the Brain Structured Computer, " Proceedings of The IEEE Computer Society's 9th International Computer Software & Application Conference (Compsac85) 1985.

(3) 坂井、三科、中村、重井:"リスト処理指向パイプライン処理システムに関する一検討",信学技報,EC85-43(1985).

(4) 三科、坂井、中村、重井:"パイプライン処理システムにおける関数型言語の実行方式",信学技報,EC85-42(1985).

(5) 遠藤,中村,重井:"階層化汎用パイプラインシステム", 信学論(D),J68-D,7,pp.1376-1383(昭60-07).

(6) 小林,遠藤,中村,重井:"汎用パイプライン処理システムの性能評価", 信学論(D), J68-D, 10, pp.1744-1752 (昭60-10).

(7) J.Backus: " Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs,"CACM, vol.21, No.8, pp.613-614,(1978).

(8) 木村,米沢:"算法表現論",岩波講座, 情報科学-12

(9) 小長谷,山本:"関数型言語とリダクションマシン",情報処理, vol.26, No.7,(1985).

(10) D.A.Turner: "A New Implementation Technique for Applicative Languages, Software practice and experience",Vol.9,No.1,(1979).

(11) P.Henderson, "Functional Programming Application and Implementation", Prentice-Hall International Series in Computer Science (1980).

(12) K.J.Berkling :"Reduction Languages for Reduction Machines", in Proc. IEEE Int. Symp. Comput. Arch. pp.133-140, (1975).

(13) 長谷川,雨宮:"データフローマシンによる並列リスト処理", 信学論(D),J66-D,12, pp.1400-1407(昭58-12).

(14) 伊波,高井,池部,中村,重井:"リスト処理指向パイプライン処理システムの構造体メモリについての一検討", 昭61東北支部連大, 2J27(1986).

(15) 池部,高井,伊波,中村,重井:"リスト処理指向パイプライン処理システムのハードウェアシュミュレータ", 昭61東北支部連大, 2J24(1986).

付　　録

ＦＰの主な原始関数

(1) Selector function
$s:\langle x_1,x_2,\ldots,x_n\rangle = x_s$, $s\in N,\ 1\le s\le n$

(2) Tail
$tl:\langle x_1,x_2,\ldots,x_n\rangle = \langle x_2,\ldots,x_n\rangle$

(3) Identity
$id:x = x$

(4) Append left
$apndl:\langle y,\langle x_1,x_2,\ldots,x_n\rangle\rangle=\langle y,x_1,x_2,\ldots,x_n\rangle$

(5) Append right
$apndr:\langle\langle x_1,x_2,\ldots,x_n\rangle,y\rangle=\langle x_1,x_2,\ldots,x_n,y\rangle$

(6) Transpose
$trans:\langle\langle a,b,c\rangle,\langle d,e,f\rangle,\langle g,h,i\rangle\rangle$
$= \langle\langle a,d,g\rangle,\langle b,e,h\rangle,\langle c,f,i\rangle\rangle$

ＦＰの主なＰＦＯ

(1) Composition "$\circ$", (合成)
$f\circ g : x = f:(g:x)$

(2) Construction "[....]", (組立て)
$[f_1,f_2,\ldots,f_n]:x = \langle f_1:x,f_2:x,\ldots,f_n:x\rangle$

(3) Condition "$\rightarrow$;", (条件)
$p\rightarrow f;g : x =$ if $p:x$ then $f:x$ else $g:x$

(4) Constant " $\bar{}$ ", (定数化)
$\bar{y}:x = y$

(5) Right Insert " / ", (右挿入)
$/f:\langle x_1,x_2,\ldots,x_n\rangle = f:\langle x_1, /f:\langle x_2,\ldots,x_n\rangle\rangle$

(6) Apply-to-all "α ", (並列作用)
$\alpha f:\langle x_1,x_2,\ldots,x_n\rangle = \langle f:x_1,f:x_2,\ldots,f:x_n\rangle$