階層的関数型の並行計算モデル

宮地利雄　片山卓也
（東京工業大学・工学部）

AN HIERACHICAL FUNCTION MODEL FOR PARALLEL SYSTEMS

Toshio Miyachi , Takuya Katayama
(Tokyo Institute of Technology)

ABSTRACT

Although functional programming language is attractive in point of naturaity and provability, lack of synchronization mechanisms for managing shared resources sets limits to its extensive utilization. In this report we propose a hierachical functional computation model for describing parallel systems. The nucleus of the model is harmony of the hierachy of pure the functions and the mechanism for synchronizing and exchanging data through a pair of communication ports. The mechanism, which we call 'rendezvous', permits us to describe nondeterministic behaviours and synchronization on shared objects. Also as examples we describe some well-known synchronizing problems in our model.

1。 まえがき

関数型言語を用いたプログラミングは，データの流れを中心に計算が進むため，制御の流れに基いて実行が行われる従来型の言語と比較し 記述の自然さ，検証の容易さ等の有利な点を持ち，また並列に実行できる部分を検出しやすいことから並列計算機上で高速に実行できる可能性が注目されている[1]。しかしながら，データの到着を待合わせることが唯一の同期方法である純粋な関数型言語では，並行処理システム内で不可避的に現われる共有資源管理などに伴った同期を自然に記述する事ができないことなどから適用可能範囲が限定されてしまっている。

本報告では，並行処理システムの記述を目的として，属性文法を基礎とした階層的関数型言語として提案されたHFP[2]によるモジュール階層を中核とし，これに同期のための基本機構を付加して拡張した関数型計算モデルの提案を行う。 この計算モデルでは，並行プロセスは 通信ポートを経由して相互に同期しデータを交換しあう計算木として表現される。 計算木は，モジュールのインスタンスを節点として

構成される。　各モジュールは属性の集合を持っている。　また，展開規則が与えられていて，これにより計算木を成長させ属性値を評価する方法を定義する。　計算過程の中心は属性値の評価であって，副作用を持たない純関数のみによってデータ駆動原理に従って進められ，計算モデルとして望ましい性質を備えている。　そして，並行処理システムが持つ非決定的動作や同期を　関数システムの性質と調和させながら記述するために　同期条件および順序条件と呼んでいる規則を伴った通信ポート間のランデブーの概念を導入している。

以下，第2節で計算モデルを定義し，第3節では　これを用いて代表的なプロセス同期問題のいくつかを記述する。第4節では　他のモデルとの比較を通して　このモデルの位置づけを与える。

2.　計算モデル

2.1　定義

我々の計算モデルは，属性文法を基礎とした階層的関数型言語として提案されたHFP[2]を拡張したもので，次の4つ組として与えられる：

（ 𝕄，ℂ，ℙ，𝔼r ）。

ここで，𝕄はモジュールの集合，ℂは通信ポートの集合である。　ℂの各要素は対をなしており，$c \in \mathbb{C}$と対になっているものを$\bar{c}$ ($\in \mathbb{C}$)により表す。𝕄とℂの各要素nには，それぞれ入力属性の集合$I(n)$と出力属性の集合$O(n)$とが与えられている。（ただし$I(n) \cap O(n) = \phi$.）　また，通信ポート$c \in \mathbb{C}$については，$I(c) = O(\bar{c})$ かつ $O(c) = I(\bar{c})$ である。

ℙはプロセスの集合で，各プロセス$p \in \mathbb{P}$に対して初期モジュール $i_p \in \{ m \mid m \in \mathbb{M}, I(m) = \phi \}$ が1個ずつ決められている。　各プロセスは，計算の進行に伴ない𝔼rに従って，初期モジュール i_p を根とし，モジュールや通信ポートのインスタンスを節点とする計算木を構成する。

𝔼rは展開規則の集合であり，個々の展開規則の一般形式は次のとおりである：

$X_0 \rightarrow X_1, \cdots, X_t$ ；　属性値定義
when 展開条件　*synch* 同期条件
order 順序条件。

ここで，X_0($\in \mathbb{M}$)を展開規則の左辺と呼び，$\mathbb{M} \cup \mathbb{C}$の要素からなるリスト（同じ要素の重複も，また空のリストも許される）$X_1, \cdots, X_t$を展開規則の右辺と呼ぶことにしよう。　また，$X_0, X_1, \cdots, X_t$は同じ名前の複数の出現を適宜に添字をつけて区別し，モジュールまたは通信ポートの出現（occurrence）と呼ぶ。

属性値定義は，$I(X_0)$と$O(X_i)$ ($1 \le i \le t$)と関数を用いて$O(X_0)$と$I(X_i)$ ($1 \le i \le t$)のそれぞれの値を決める定義式の集まりである。　我々のモデルでは，属性値は一度定義されると「代入」と異なりその後再定義されることがなく，また定義式中に現われる関数には副作用を持たない純粋な関数のみを許しており，従って属性値の評価順序が直接に計算結果に影響を及ぼすことがない。

展開条件は$I(X_0)$上の述語であって，これが真になる時に限って　その展開規則を適用することができる。

同期条件は $\{X_1, \cdots, X_t\} \cap \mathbb{C}$の部分集合であって，ここで指定された通信ポートがそれぞれ対をなす通信ポートと結合して属性値の交換を行うこと（ランデブー）が可能である時に限って　その展開規則を適用することができる。

順序条件は $\{X_1, \cdots, X_t\}$上の半順序関係であって，この展開規則の適用によって新しく作られる子節点を根とする部分木内の通信ポートのランデブーの時間的な順序を規定している。

2.2 動作

我々の計算モデルにおける計算の過程は，計算木の展開，属性値の評価，通信ポート間のランデブーの3種類の動作を軸として進行する．

（a）計算木の展開

計算木の形成は，モジュール，通信ポート，初期モジュールをそれぞれ非終端記号，終端記号，開始記号とみなし，展開規則の $X_0 \to X_1, \cdots, X_t$ を生成規則とみなした時の導出木の形成と同様の方法で行われる．　すなわち，計算木の葉の位置にあるモジュールは，左辺がそのモジュール名と一致しているような展開規則を捜し　その右辺を構成しているモジュールや通信ポートの新しいインスタンスを作り　自身の新しい子節点として計算木に加える．しかしながら，導出木の場合と異なり，適用する展開規則を任意に選ぶことができるのではなく，展開条件と同期条件をともに満足していることが必要とされる．　条件を満足する展開規則が複数個存在する場合には，そのうちの1つが非決定的に選ばれる．　展開条件が　その左辺のモジュールの入力属性の値により計算木の展開を制御するプロセス内的な条件であるのに対し，同期条件は　他のプロセスの通信ポートの状態により決定されるプロセス間的な条件となっている．

（b）属性値の評価

計算木の展開の際に作られたモジュールや通信ポートの新しいインスタンスの上の属性は，すべて初め値が未定義状態になっている．　そして展開後，展開規則の中の属性値定義を用いて各属性の値が評価されていく．　この評価過程は，副作用なしに，完全にデータ駆動的に進められる．

属性値として stream を許したモデルも以前に考えたが[3]，本報告においては属性は構造を持ったデータであるにせよ単一の塊であって，stream のような値の取扱いは考えていない．

簡単な例として，100 以下の自然数とそれに対する階乗の値を次々に計算するプロセスの記述を次に掲げ，計算木の展開と属性値の評価の様子を図1に示す．

［例 1］　Factorial

$\mathbb{M} = \{E, F\}$

$I(F) = \{n, f\}$

$\mathbb{C} = \phi$

$\mathbb{P} = \{Prod\}$　　initial-module(Prod) = E

$\mathbb{E}_r$:

$E \to F$;　$n.F = 1$,　$f.F = 1$.

$F_0 \to F_1$;　$n.F_1 = n.F_0 + 1$,

$f.F_1 = f.F_0 * (n.F_0 + 1)$

when　$n.F_0 < 100$

図中の四角形はモジュールを，破線はその階層関係を，点線は属性値が評価されていく様子を表わしている．なお，この段階では我々のモデルも HFP と全く同じ計算モデルに縮退している．

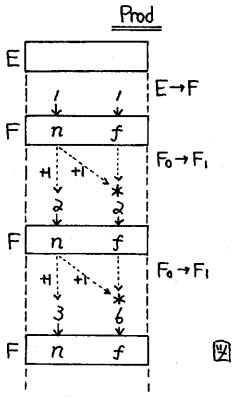

図1．例1のシステムに対する計算木．

（c）ランデブー（rendezvous）

異なるプロセスの計算木に属する1組の対をなす通信ポートのインスタンスが結合し，その属性値の交換を行う動作を「ランデブー」と呼ぶ．　ランデブーを一旦開始した通信ポートが，再び別の組合せを作ってランデブーすることはない．

通信ポートの各インスタンスは，そ

れを作り出した展開規則の同期条件に含まれていたか否かにより，条件つき同期待ち通信ポートと無条件同期待ち通信ポートに分類される。　前者では展開によりインスタンスが作られた時に，後者では適当な時間の後に対をなす通信ポートとの対応関係が成立した時にランデブーを開始する。　その後，属性名の対応により2つの通信ポートの間で値の受渡しが行われ，通信ポートの上の属性の値がすべて定義された時にランデブーが完了する。

属性値の評価がデータの依存関係のみにより制御されているのに対し，ランデブー動作は実行順序が展開規則の順序条件で与えられた半順序関係により拘束されている。　すなわち，展開規則 $e \in \mathbb{E}_r$ の順序条件で指定された半順序関係を $\ll_e$ で表わす時，計算木内の通信ポートのインスタンスの集合の上の半順序関係 $\ll$ が次式により定義される：

$$\text{for all } C_1, C_2 \in \mathbb{C}$$

$$C_1 \ll C_2 \overset{\text{def}}{\Longleftrightarrow} \exists e \in \mathbb{E}_r,\ \exists n_1, n_2 \in \mathbb{M} \cup \mathbb{C}$$

$$\begin{bmatrix} C_1 \in \underline{subtree}(n_1) & \& \\ C_2 \in \underline{subtree}(n_2) & \& \\ n_1 \ll_e n_2 \end{bmatrix}$$

ここで $\underline{subtree}(n)$ は節点 n を根とする部分木を表わす。　そして，$C_1 \ll C_2$ である時には，C_1 のランデブーの完了より前に C_2 のランデブーが開始されない事を要請している。

ここで前掲の例を拡張して，100以下の自然数とそれに対する階乗の値を順に生成し通信ポートを経由して他のプロセスにこれを供給するプロセスを例として次に掲げ　動作の様子を示すために計算木のスナップショットを図2に示す。

［例2］　Factorial Producer

$\mathbb{M} = \{ E, F, C \}$

$I(F) = \{ n, f \}$　…

$\mathbb{C} = \{ P, \overline{P} \}$

$I(P) = O(\overline{P}) = \{ n, f \}$

$\mathbb{P} = \{ Prod, Cons \}$

$\underline{initial\text{-}module}(Prod) = E$

$\underline{initial\text{-}module}(Cons) = C$

$\mathbb{E}_r$:

$E \to F$;　$n.F = 1$, $f.F = 1$

$F_0 \to P, F_1$;　$n.F_1 = n.F_0 + 1$,
$f.F_1 = f.F_0 * (n.F_0 + 1)$,
$n.P = n.F_0$,
$f.P = f.F_0$

$\underline{when}\ n.F_0 < 100$　$\underline{order}\ \{ P \ll F_1 \}$.

$C_0 \to \overline{P}, C_1$;　$\underline{synch}\ \{ \overline{P} \}$

図の中では，通信ポートを二重の四角形で，ランデブーの組合せを二重線の矢印により示す。

なお，2番めの展開規則中の順序条件を省略した場合には生成された値を供給する順序が定義されないことになり，また　同期条件 $\underline{synch}\ \{P\}$ に置換えた時にはプロセス Prod の計算木の生長がランデブーと歩調を合せ進行するようになることを注意しておこう。

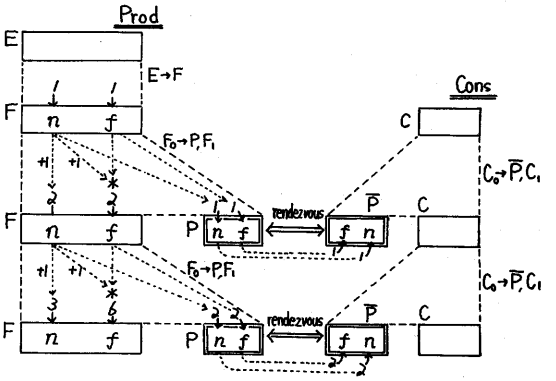

図2. 例2のシステムに対する計算木。

3.　応用記述例

本節では，プロセス同期に関する代表的な問題のいくつかについて我々の計算モデルを用いて記述を試る。

3.1　Bounded Buffer

最初の例題は　容量制限つきバッフ

ァの問題である。　次に記述例を掲げ，図3に計算木のスナップショットを示す。

［例3］　Bounded Buffer

$\mathbb{M} = \{ IB, B, \cdots \}$

$I(B) = \{ a, dp, dc \}$　…

$\mathbb{C} = \{ GET, \overline{GET}, PUT, \overline{PUT} \}$

$I(\overline{GET}) = O(GET) = \{d\}$

$O(\overline{PUT}) = I(PUT) = \{d\}$

$\mathbb{P} = \{ Buffer, \cdots \}$

$\underline{initial\text{-}module}(Buffer) = IB$

…

$\mathbb{E}r$：

$IB \to B$；　$dp.B = 0$, $dc.B = 0$, $a.B = \#$.

$B_0 \to \overline{GET}, B_1$; $d.\overline{GET} = a.B_0[dp.B_0]$,

$dp.B_1 = dp.B_0 + 1 \ \underline{mod}\ n$,

$dc.B_1 = dc.B_0 - 1$,

$a.B_1 = a.B_0$

$\underline{when}\ dc.B_0 > 0$　$\underline{synch}\ \{\overline{GET}\}$.

$B_0 \to \overline{PUT}, B_1$; $dc.B_1 = dc.B_0 + 1$, $dp.B_1 = dp.B_0$,

$a.B_1 = a.B_0[(dp.B_0 + dc.B_0 + 1 \ \underline{mod}\ n) \setminus d.\overline{PUT}]$

$\underline{when}\ dc.B_0 < n$　$\underline{synch}\ \{\overline{PUT}\}$.

… …

バッファ，リーダ，ライタのそれぞれにプロセスを割当て，ここでは　そのうちバッファ関連以外については省略した。　また属性 a は配列 $\underline{array}[0..n-1]$ であり，$a[i]$ で i 番めの要素を，$a[i \setminus x]$ で配列 a の i 番めの要素の値だけを x で置きかえて得られる配列を表す。

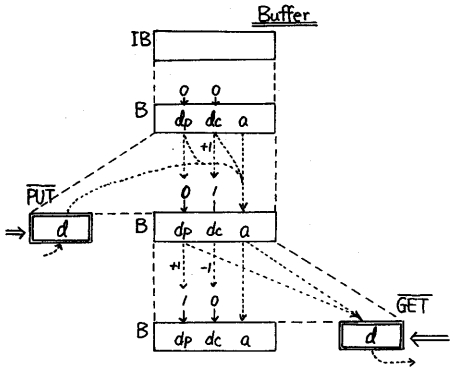

図3. Bounded Bufferに対する計算木。

3.2　セマフォ

2番めの例題では　セマフォの PV 操作のシミュレートを試る。

［例4］　Semaphore

$\mathbb{M} = \{ IS, S, \cdots \}$

$I(S) = \{sc\}$

$\mathbb{C} = \{ P, \overline{P}, V, \overline{V} \}$

$\mathbb{P} = \{ Sem, \cdots \}$

$\underline{initial\text{-}module}(Sem) = IS$

…

$\mathbb{E}r$：

$IS \to S$；　$sc.S =$ セマフォ初期値．

$S_0 \to \overline{P}, S_1$；　$sc.S_1 = sc.S_0 - 1$

$\underline{when}\ sc.S_0 > 0$　$\underline{synch}\ \{\overline{P}\}$.

$S_0 \to \overline{V}, S_1$；　$sc.S_1 = sc.S_0 + 1$

$\underline{synch}\ \{\overline{V}\}$.

ただし，我々のモデルによる記述では　P操作において待たされているプロセスが複数個存在する時に　必ずしも先着順に起動されるとは限らない。図4には　この例題の計算木を示す。

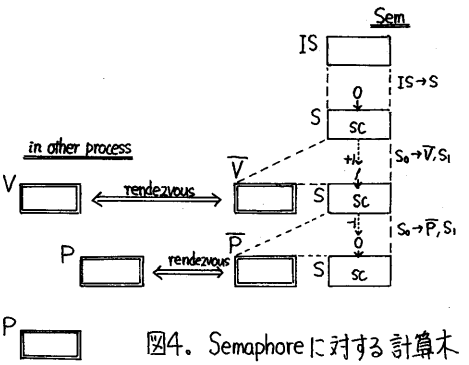

図4. Semaphoreに対する計算木

3.3　Dining Philosophers

3番めの例題は dining philosophers 問題である。　ここでは「M（in P）」の記法でプロセスP中のモジュールMを表して記述量の圧縮をはかっているが，本質的なモデルの拡張ではない。

［例5］　Dining Philosophers

$\mathbb{M} = \{ P, Eat, Think, F \}$

$\mathbb{C} = \{ U^i, \overline{U^i}, D^i, \overline{D^i} \mid 0 \le i \le 4 \}$

$\mathbb{P} = \{ phil_i \mid 0 \le i \le 4 \} \cup \{ fork_i \mid 0 \le i \le 4 \}$

$\underline{initial\text{-}module}(phil_i) = P \quad (0 \le i \le 4)$

initial-module ($fork_i$) = F　($0 \le i \le 4$)

Er:　ただし i+1 は 5 を法とした加算とする

P_0 (in $phil_i$) ⟶
　U^i, U^{i+1}, Eat, D^i, D^{i+1}, Think, P_1;
　synch {U^i, U^{i+1}}
　order {Eat ≪ D^i, Eat ≪ D^{i+1},
　　D^i ≪ Think, D^{i+1} ≪ Think, Think ≪ P_1}.
ただし　$i = 0,1,2,3,4$

Eat ⟶ ;

Think ⟶ ;

F_0 (in $fork_i$) ⟶ $\overline{U^i}$, $\overline{D^i}$, F_1 ;
　order {$\overline{U^i}$ ≪ $\overline{D^i}$ ≪ F_1}.
ただし　$i = 0,1,2,3,4$

5人の哲学者と5本のフォークに対して それぞれプロセス $phil_i$, $fork_i$ が割当てられる。 通信ポート U^i のランデブーは哲学者が i 番めのフォークを持ち上げることに，D^i のそれは i 番めのフォークをもどすことに対応している。 計算木の様子を図5に示す。

3.4 Petrinet

最後の例題として，一般の Petrinet を我々のモデルを用いてシミュレートすることを考えよう。 ただし簡単のため，始点と終点がそれぞれ同じ2本の枝は存在しないものとする。

シミュレートすべき Petri net の place 節点の集合を $\mathbb{P}l$，transition 節点の集合を $\mathbb{T}r$，有向枝の集合を $\mathbb{A}$ ($= \mathbb{T}r \times \mathbb{P}l \cup \mathbb{P}l \times \mathbb{T}r$) を書くことにしよう。 また節点 n を終点とする有向枝の集合を IN(n) ($\subset \mathbb{A}$) により，節点 n を始点とする有向枝の集合を OUT(n) ($\subset \mathbb{A}$) により表す。 さらに，place 節点 n に対する初期配置を $token_0(n)$ と書こう。 この時 Petrinet は我々のモデルの上で以下のように表現される。

［例6］　Petrinet

$\mathbb{M} = \{ i(n) \mid n \in \mathbb{P}l \cup \mathbb{T}r \}$
　$\cup \{ m(n) \mid n \in \mathbb{P}l \cup \mathbb{T}r \}$
　$I(m(n)) = \{tc\}$　for all $n \in \mathbb{P}l$

$\mathbb{C} = \mathbb{A} \cup \{ \overline{a} \mid a \in \mathbb{A} \}$

$\mathbb{P} = \mathbb{P}l \cup \mathbb{T}r$

initial-module (n) = i(n)　for all $n \in \mathbb{P}$

Er:

- for all $n \in \mathbb{P}$
  $i(n) \to m(n)$;　$tc.m(n) = token_0(n)$.
- for all $pl \in \mathbb{P}l$, $a \in IN(pl)$
  $m(pl)_0 \to \overline{a}$, $m(pl)_1$;
  $tc.m(pl)_1 = tc.m(pl)_0 + 1$
  synch {$\overline{a}$}.
- for all $pl \in \mathbb{P}l$, $e \in OUT(pl)$
  $m(pl)_0 \to \overline{e}$, $m(pl)_1$;
  $tc.m(pl)_1 = tc.m(pl)_0 - 1$
  when $tc.m(pl)_0 > 0$　synch {$\overline{e}$}.
- for all $tr \in \mathbb{T}r$
  $m(tr)_0 \to e^1, e^2, \cdots, e^s, a^1, a^2, \cdots, a^t, m(tr)_1$
  synch {$e^1, e^2, \cdots, e^s$}
  ただし {$a^1, a^2, \cdots, a^t$} = OUT(tr),
  {$e^1, e^2, \cdots, e^s$} = IN(tr).

4. 他のモデルとの比較

本節では，他のモデルとの比較を通して 我々の計算モデルの位置づけを行う。

並行プロセスにおける同期の方式は，共有記憶によるものとメッセージ授受によるものとに大別される。 前者と

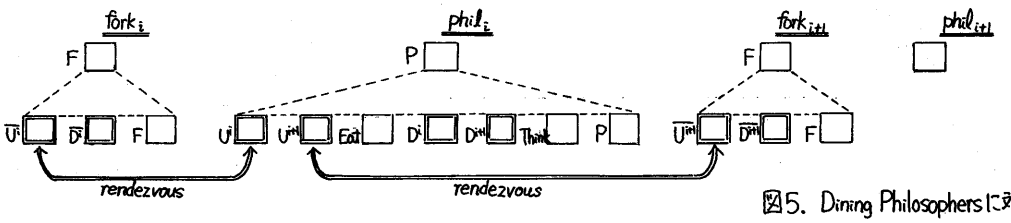

図5. Dining Philosophers に対する計算木

しては，セマフォ，条件つき臨界領域（conditional critical region）[4]，順路式（path expression）などがよく知られており，後者では アクタ・モデル（actor model）[6]，CSP（communicating sequential process）[7]，Adaのランデブー（rendezvous）[8]などを代表的なものとして挙げることができる。

本報告のような関数型の計算モデルにおいても同様に2通りのアプローチが可能である。我々は先に共有記憶によるモデルとして ペトリネットを基礎とした 関数と共有データから構成されるネットワークによる関数起動モデル（Function activation model）[9]を提案しているが，これと異なり本論文ではメッセージ授受による同期方式を採用したモデルの構築を試た。前者の場合にはデータと関数というように受動的な要素と能動的な要素とが存在して二部グラフを構成しているが，このモデルでは すべてのシステム要素をプロセスとして記述することになる。

メッセージの交換は，我々の計算モデルでは，通信ポート間のランデブーとして行われ，これはプロセス間にバッファを置かず両プロセスが同時に通信動作を行うことにより同期的に相互にメッセージの授受を行う強結合通信方式（tightly coupled communication）に分類されるものである。Adaのランデブーと非常に似ているが，Adaではエントリ呼出しを accept 文が受取ることによりランデブーが成立するのに対して，我々のモデルにおけるランデブーは完全な対称性を有する動作となっている。

また 並列計算モデルの重要なものとしてデータフロー（dataflow）モデルが知られているが，これには元来は共用資源や共用対象（shared object）の管理のような平行プロセスの同期を記述する機構がなく，現在研究されており[10]我々のモデルのランデブー機構もその解のひとつとなり得ると考えられる。

5. まとめ

階層的関数型の並行計算モデルを提案し，これを用いた平行プロセス系の記述を試た。

残された問題として現在研究をしているのは次の各点である：

① 動作の形式化，
② 検証技法，
③ 実現の手法。

これらのうち，検証技法については，プロセスごとの属性値に関する側面では 属性文法の検証技法[11]を適用できることが期待される。ランデブー動作に関する側面は，我々のモデルから対応するPetrinetに変換する規則を与え，得られたPetrinet上での検証を行うことにより 現実的なほとんどの場合をカバーできる程度の安全性を保証することができるものと予想している。

実現に関する問題としては，HFPの場合と共通した問題[12]のほかに，条件つき同期待ち通信ポート相互間での可能なランデブーの組合せを見つける方法が 一般にはパターン・マッチングと同じ複雑さを持ち，取扱いにくい問題として挙げられる。

［参考文献］

1. Backus, J：1977 ACM Turing Award Lecture：Can Programming Be Liberated from the Von Neumann Style? A Functional Style and its Algebra of Programs. C. ACM Vol. 21 No. 8 (1978.8)
2. Katayama, T.：HFP：A Hierachical and Functional Programming based on Attribute Grammar. Proc. of 5th International Conference on Software Engineering (1981.3)

3. 宮地，片山： 並列処理の関数的な記述の試み.
情報処理学会第23回全国大会予稿集 (1981. 10)
4. Brinch Hansen, P. : Operating System Principles. Prentice-Hall (1973)
5. Cambell, R.H. & Habermann, A.N. : The Specification of Process Synchronization by Path Expression.
Lecture Notes in Computer Science Vol.16
Springer-Verlag (1974)
6. Hewitt, C. & Baker, H. Jr. : Actors and Continuous Functionals.
MIT/LCS/TR-194 (1977.12)
7. Hoare, C.A.R. : Communicating Sequential Processes. C.ACM Vol.21, No.8 (1978.8)
8. United States Department of Defence : Reference Manual for the Ada Programming Language (1980.7)
9. 片山，宮地： 関数＋共有データ＝関数型並行プログラム.
情報処理学会 ソフトウェア基礎論研究会資料 1-2 (1982.6)
10. Arvind, Gostelow, K.P. & Plouffe, W. : The (Preliminary) Id Report : An Asynchronous Programming Language and Computing Machine.
University of California, Irvine, Dept. of Information and Comp. Sci. T.R. #114 (1978)
11. Katayama, T. & Hoshino, Y. : Verification of Attribute Grammars.
Conf. 8th ACM Symp. on Principles of Programming Languages (1981.1)
12. Katayama, T. : Treatment of Big Values in an Applicative Language HFP — Translation of By-value Access to By-update Access.
Tokyo Inst. of Tech. Dept. of Comp. Sci. TR CS-K8202
(1982)