数理解析研究所講究録
902巻 1995年 49-63

応答型逐次プロセスの部分計算を用いた並列実行

Parallel Decomposition of Reactive Sequential Processes Using Partial Evaluation

村上 昌己
Masaki Murakami

岡山大学 工学部

Faculty of Engineering, Okayama University
murakami@momo.it.okayama-u.ac.jp

概 要

本稿では、通信動作からコンティニュエーションへの関数の形で定義された逐次的なプロセスについて、並列に実行可能ないくつかの部分プロセスによって構成された形に変換する方法の一般的な枠組みについて述べる．本稿で述べる方法の基本的な考え方は次のようなものである．通信動作の実行の間、すなわち外部からの入力の待ち合わせや出力の計算を行なっている間に、その通信動作によるコンティニュエーションの部分計算を並行して行なう．さらにコンティニュエーションの部分計算も並行に行なうことによって、パイプライン並列性を引き出すことが可能であることを示す．

1 はじめに

筆者は先に、CSP で記述された並行プロセスを部分評価する方法について報告した [Mu92]．そこで提案した方法は次のようなものである．プロセス P とその実行環境についての制約条件 p に対して、p を満足する環境のもとでは P と動作によって区別することができないプロセス P' とする．このとき P' を P と p から組織的に求めるために、構文的な変換規則系を用いる．[Mu92] では、逐次型プログラムの部分計算手法が与えられたとき、それを拡張して並行プロセスの変換規則系を構成するための一般的な方法

を与えた．すなわち並行プロセスの部分計算のための、逐次型プログラムの部分計算手法の応用の枠組みを示した．そこでは p を記述するために、時相論理式を用いた．

そこで提案された規則系には以下のような問題点が残されていた．第一に、並行合成に関する規則が適用できる場合が強く制限されている点である．すなわち P_1, P_2 という 2 つ以上のサブプロセスの並行合成によって定められるプロセス $P_1 \| P_2$ について、変換規則が適用できるためには、P_1 と P_2 がお互いに値を送り合うということがなく本質的に一方向にのみ値が流れることが必要であった．このため、そこで述べられた規則系では変換できない場合がみられた．そこで双方向に通信を行なう $P_1 \| P_2$ を部分計算するためには、[Ho85] で述べられた等式を用いて $P_1 \| P_2$ を逐次的な非決定性のプロセスに展開した後に、部分計算を行なわなければならなかった．

第二に、[Mu92] の規則で陽に与えられた変換規則は主に、応答型のプロセスの部分計算を、逐次型プログラムの部分計算に帰着するものであった．そこでは、逐次的なプロセスを新たに並列化するための具体的な手法は示されていない．したがって古典的な逐次型プログラムの部分計算法との組み合せでは、各 P_i 内部で逐次的に実行される計算の効率を向上させるのみであった．したがって上に述べたように $P_1 \| P_2$ を一旦逐次的なプロセスに変換してしまった後に部分計算を行なった結果では、もともと $P_1 \| P_2$ で実現されていた並列性が失われている可能性も考えられる． $P_1 \| P_2$ を逐次的な非決定性のプロセスに展開するために用いた等式を逆に用いることによって、$P_1 \| P_2$ を展開した形のプロセスをもとの形に戻すことは可能であるが、部分計算の結果が偶々そのような等式が適用できる形をしている場合は殆ど期待できない．

また与えられたプロセスのネットワーク・トポロジを大きく変更することによって効率的な並列アルゴリズムを得る等の変換も期待できず、変換能力にも不満が残るものである．

以上のような問題点は、逐次的なプロセスを複数の並列なサブプロセスに分割する方法を与えることによって解決されることが期待できる．

そこで本稿では、逐次的なプロセスの動作を複数の並列なプロセスに

よってシミュレートする方法の一般的な枠組みについて述べる．ここで述べる方法は再び古典的な逐次型プログラムの部分計算手法の応用のための枠組みである．すなわち具体的な部分計算の手法が与えられたとき、それを拡張して逐次型プロセスの並列化の手法を構成する方法について述べる．ここで対象とする逐次型プロセスは、通信動作からコンティニュエーションへの関数の形で定義されたものであるとする．本稿で述べる並列化手法の基本的な考え方は、以下のようなものである． 通信動作の実行の間、すなわち外部からの入力の待ち合わせやプロセスの内部変数の値を用いて出力値の計算を行なっている間に、その通信動作によるコンティニュエーションの部分計算を並行して行なう．このように実行前でなく実行の途中にプログラムの部分計算を行なうという考え方は、並列計算に固有のものである．さらにコンティニュエーションの部分計算も並行に行なうことによって、さらに並列に実行可能なプロセスに分割できることを示す．これは逐次プログラムのパイプライン並列実行を、陽に記述したものであると考えることができる．

2 準備

2.1 CSP

本稿では逐次型プロセスを記述するためにCSPを用いる． CSP[Ho85]では、プロセスの動作は通信動作からプロセスへの関数である、という定式化をしている．通信動作は入力命令 $c?x$ と出力命令 $c!v$ によって実行される．ここで c は通信チャネル名であり、入力命令と出力命令が揃って実行されることにより、変数 x に式 v を評価した結果の値が送られる．また同じ通信チャネル名の入力動作が2つ以上のプロセスに出現する場合は、それらは同時に実行される．

本稿ではこのような通信動作 m_i からそれを実行した後に得られるプロセス P_i に写像する関数を表記するため、以下のような guarded command 風表記を用いる．

$$\{m_1 \to P_1 [\!] \cdots [\!] m_n \to P_n\}$$

ここで $[\!]$ は、決定性の選択をあらわす記号であり、$j \neq k$ ならば $m_j \neq m_k$

である．または動作の集合 Mに対して

$$\{[\!]_{m_i \in M} m_i \rightarrow P_i\}$$

と表記も用いる．全く動作しないプロセスを、$nill$であらわす．

これらのプロセスのセマンティクスは失敗集合 (failures) によって定義される．

定義 2. 1: [失敗集合][Ho85] プロセス Pが実行することができる動作の集合を$\alpha(P)$、Pの実行する通信動作の系列の集合を $traces(P)(\subset (\alpha(P))^*)$であらわすことにする．

このときプロセス Pの失敗集合は、$failures(P)$ は以下のように定義される．

$$failures(P) = \{(\overline{m}, R) |\ \ \overline{m} \in traces(P), R \in refusals(P/\overline{m})\}$$

また

$refusals(Q) = \{R|R$はプロセス Q の環境が Q に実行
させようとしている動作の集合．
かつ Rのいずれの要素についても、
Q が直ちにデッドロックする可能性がある．$\}$

である．また $P/\overline{m}$は、プロセス Pが系列$\overline{m}$を実行した結果残るプロセス、すなわち Pの$\overline{m}$によるコンティニュエーションである．失敗集合が等しい二つのプロセスを等価なプロセスと呼ぶ．

Pが $\{[\!]_{i \in I}(m_i \rightarrow P_i)\}$ であるときの失敗集合は，P_i の失敗集合を用いて以下のように定められる．

$$\begin{aligned} failures(P) =& \{([m_i]^\wedge \overline{m}, R) | (\overline{m}, R) \in failures(P_i)\} \cup \\ & \{([\,], R) | \forall i \in I, m_i \notin R\} \end{aligned}$$

ここで$\overline{m_1}^\wedge\overline{m_2}$は有限系列$\overline{m_1}$と有限又は無限系列$\overline{m_2}$の連接を表わす．すな

わち,

$$\overline{m_1} = [m_1^1, m_1^2, \ldots, m_1^k], \quad \overline{m_2} = [m_2^1, m_2^2, \ldots, m_2^h]$$

(又は$\overline{m_2} = [m_2^1, m_2^2, \ldots]$) のとき,

$$\overline{m_1}^\wedge\overline{m_2} = [m_1^1, m_1^2, \ldots, m_1^k, m_2^1, m_2^2, \ldots, m_2^h]$$

(又は$= [m_1^1, m_1^2, \ldots, m_1^k, m_2^1, m_2^2, \ldots]$)

[ ] は空な系列を表す.

本稿では複数のプロセス P_1, P_2を並行に実行した場合のように動作するプロセスを $P_1 \| P_2$のように表記する. 並行に走るプロセスの間の通信は、出力側と入力側に共通なチャネルを用いて、先に述べた通信動作を使って行なわれる.

Pが $P_1 \| P_2$のときの失敗集合は以下のようになる.

$$\begin{aligned} &\mathit{failures}(P_1 \| P_2) = \\ &\{(\overline{m}, R_1 \cup R_2) \mid \ \overline{m} \in \mathit{traces}(P_1 \| P_2), \\ &\qquad\qquad (\overline{m} \uparrow \alpha(P_1), R_1) \in \mathit{failures}(P_1), \\ &\qquad\qquad (\overline{m} \uparrow \alpha(P_2), R_2) \in \mathit{failures}(P_2)\} \end{aligned}$$

ここで

$$\begin{aligned} \mathit{traces}(P_1 \| P_2) \ = \{t \mid \ & t \uparrow \alpha(P_1) \in \mathit{traces}(P_1), \\ & t \uparrow \alpha(P_2) \in \mathit{traces}(P_2), \\ & t \in (\alpha(P_1) \cup \alpha(P_2))^*\} \end{aligned}$$

である. また Mを通信動作の集合とするとき, $\overline{m} \uparrow M$ は, $\overline{m}$から M に含まれる動作を残し他はすべて取り去ることによって得られる系列である.

またプロセス Pと動作の集合 $S(\subset \alpha(P))$ について、$P \backslash S$ は Pの動作のうち Sに含まれるものを外部から隠した (conceal した) プロセスをあらわす.

2.2 プロセスの実行環境

本稿では [Mu92] と同様に、プロセスの環境を表現するために時相論理 (Temporal Logic) を用いる．時相論理式の真/偽は，時間的の経過を意味する系列を与えることによって定まる．ここで時間の経過を応答型の並行のプロセスにとっての時間の経過と考えれば，各時点において実行した通信動作の系列によって表わされる．すなわち時相論理式の真/偽は，プロセスの動作系列を与えることによって定まる．したがって，逆に動作系列が与えられたときそれが充たす/充たさない制約を記述するために好都合である．

時相論理では，論理式に G, X のような時相記号を含むことが許される．(F (eventually) 記号や U (until) 記号等もよく使われるが，本稿ではとりあえずこれらの記号は含まないものとする．) これは変換型のパラダイムにおいて，入力についての制約を述語論理で記述するような方法を，応答型のパラダイムに自然に拡張したものといえる．各記号の意味は，直観的にはよく知られているように以下の通りである．

- Xp: 現在の次の時点で p が成りたつ．
- Gp: 現在から先のすべての時点で、p が常に成りたつ．

以下では，与えられた動作系列についての論理式の真/偽を形式的に定める．

通常は時相記号を含まない述語の真偽についてプロセスの内部変数の値から定まる状態の概念を用いて定義することが多い．しかしながら本稿では，時相記号を含まない述語の真/偽は本質的ではないため，議論を簡単にするため状態についての詳細な議論はしない．すなわち時相記号を含まない述語についてはそれを真とする動作系列の集合が定められているものとする．すなわち，与えられた時相記号を含まない述語 p について，系列の集合 $models(p)$ がそれぞれ定まっているものとする．

以下では$\overline{m}$を通信動作の有限系列 $[m_1, m_2, \ldots, m_n]$
(又は無限系列 $[m_1, m_2, \ldots, m_i, \ldots]$) とするとき，

$$head(\overline{m}) = m_1,$$

$$tail_i(\overline{m}) = [m_i, \ldots, m_n]$$

(又は $[m_i, \ldots]$) とする.

定義 2. 2:

$\overline{m}$を通信動作の系列とするとき:

1. p が時相記号を含まない述語論理式であるとき,

$$\overline{m} \models p \text{ iff } \overline{m} \in models(p)$$

2. $\overline{m} \models \mathsf{X}p$ iff $tail_2(\overline{m}) \models p$

3. $\overline{m} \models \mathsf{G}p$ iff $\forall i(1 \leq i), tail_i(\overline{m}) \models p$

4. $\overline{m} \models p_1 \wedge p_2$ iff $\overline{m} \models p_1$ かつ $\overline{m} \models p_2$

5. $\overline{m} \models p_1 \supset p_2$ iff $\overline{m} \models p_1$ ならば $\overline{m} \models p_2$

6. $\overline{m} \models \forall x p(x)$ iff すべての x について $\overline{m} \models p(x)$.

プロセス Pの制約 qを充たす動作系列とは, $\overline{m} \in traces(P), \overline{m} \models q$ となる $\overline{m}$のことである.

このような時相記号を含む論理式の真偽は、時間的の経過を意味する系列を与えることによって定まる. 本稿では、時間の経過とは応答型の並行のプロセスにとっての時間の経過であり、それは各時点において実行した通信動作の系列によって表わされる. 詳しくは [Mu92] を参照されたい.

2.3 限定失敗集合等価

本稿では、部分計算の結果が正しいものであること、すなわち与えられた環境のもとで部分計算前のプロセスと区別のできない動作をすることを表現するために、限定失敗集合等価の概念を導入する.

定義 2. 3: [失敗集合の qによる制限]

Fを動作の有限系列と動作の集合の対の集合、qを動作系列についての時相論理式による制約とする．Fのqによる制限$F \downarrow q$とは、以下のように定まる集合である．

$$F \downarrow q = \{(\overline{m}, R) |\ (\overline{m}, R) \in F, \exists r, head(r) \notin R, \exists R'(\overline{m}^\wedge r, R') \in F, \overline{m}^\wedge r \models q\}$$

定義 2. 4: [プロセスの限定失敗集合等価]

qを動作系列についての時相論理式による制約とする．$\alpha(P_1) = \alpha(P_2)$なるプロセス$P_1, P_2$が$q$のもとで限定等価であるとは、$failures(P_1) \downarrow q = failures(P_2) \downarrow q$ を充たすことであり、

$$P_1 \sim P_2 \ wrt \ q$$

と表記する．

$P_1 \sim P_2 \ wrt \ q$であるようなプロセスP_1, P_2については、qを満足する動作系列のみが起こりうる環境で動作している限り、両者が同じプロセスとみなすことができると考えられる．この関係が同値関係であることは、容易に示せる．

3 部分計算と並列分割

本稿で述べる方法の基本的な考え方は以下のようなものである．プロセス$P \stackrel{\text{def}}{=} \{m \rightarrow P_m\}$ と Pが呼び出されたときの初期条件qを考えたとき、mをシミュレートするプロセスの実行が完了するのを待っている間に、「qでmが実行された後の条件」すなわち$q \supset \mathsf{X}r$ となるようなrを用いてP_mを部分計算する．mの実行が完了したところで、P_mの部分計算を打ち切り実行に移る．さらにP_mが$P_m \stackrel{\text{def}}{=} \{m' \rightarrow P'_m\}$という逐次型のプロセスであった場合、$P_m$の$q'$による部分計算は、$m'$の$q'$による部分計算と、$P'_m$の部分計算に分割される．

定義 3. 1: [動作のシミュレーション]

プロセス $\{m \rightarrow P\}$ と初期条件 qについて、プロセス R_mが、qのもとで動作 m の忠実なシミュレーションであるとは、R_mが以下のプロセスと qについて限定失敗集合等価 であることをいう. 通信動作の集合 Cを $C \cap \alpha(\{m \rightarrow P\}) = \emptyset$ とし、また特別な出力動作 $s!\omega(m) \notin \alpha(\{m \rightarrow P\})$ について、

$$\{\|_{c \in C} c \rightarrow R' \| m \rightarrow \{s!\omega(m) \rightarrow nil\}\}$$

ここで R' はやはり m の qのもとで忠実なシミュレーションで、C及び $s!\omega$ を R_mと共有するもとする[1].

ここで Cは R_mの部分計算出力動作集合と呼び $PEO(R_m)$ と表記する. また $s!\omega(m)$ は R_mの停止信号と呼び $halt(R_m)$ と表記する.

定義 3.1 で qが恒真な述語 $(true)$ であるとき、Rは単に m の忠実なシミュレーションであるという.

定義 3. 2: [忠実な部分計算]

プロセス Pについて、通信動作の集合 Cが $(C \cap \alpha(\{m \rightarrow P\}) = \emptyset)$、また特別な入力動作 $s?u$ について $s?u \notin \alpha(\{m \rightarrow P\})$ とする. プロセス R が、プロセス Pの条件 qによる忠実な部分計算であるとは、Rが以下のプロセスと等価であることをいう.

$$\{\|_{c \in C} c \rightarrow R' \| s?u \rightarrow P'\}$$

ここで R'はやはりプロセス Pの条件 qによる忠実な部分計算で、Cおよび $s?\omega$を Rと共有する[2]. また P'は qについて P と限定失敗集合等価であるようなプロセスであるものとする.

ここで Cは Rの部分計算入力動作集合と呼び $PEI(R)$ と表記する. また $s?\omega$は Rの起動信号と呼び $wake(R)$ と表記する.

---

[1]後に導入する記法を用いれば、$PEO(R_m) = PEO(R')$ かつ $halt(R_m) = halt(R')$ と書ける.

[2]定義 3.1 の場合と同様に、後に導入する記法を用いれば、$PEI(R) = PEI(R')$ かつ $wake(R) = wake(R')$ と書ける.

命題 1

プロセス $\{m \rightarrow P\}$ と条件 $\mathsf{X}q$について、プロセス Rを m の忠実なシミュレーション、Q をプロセス Pの条件 q による忠実な部分計算で、かつ $PEO(R) = PEI(Q) = C$であり、$halt(R)$ と $wake(Q)$ によって Rから Q への通信 s が行なわれるものとする．このとき $\{m \rightarrow P\}$ は以下のプロセスと $\mathsf{X}q$について限定失敗集合等価である．

$$(R \| Q) \backslash (C \cup \{s\}).$$

上のプロセスの動作は直観的には、以下のようなものである．Rが動作 m の実行を完了させるまでは、Q は Rの実行によって得られた情報を $c \in C$ によって受けとり、それを用いて Pの部分計算を行なっている．s は m の実行が実際に完了したので、Q に Pの部分計算を打ち切らせて得られたプロセス (定義 3.2 の P') の実行を開始させるはたらきをする．

定義 3. 3: [動作の部分計算]

プロセス $\{m \rightarrow P\}$ と初期条件 qについて、通信動作の集合 Cが $C \cap \alpha(\{m \rightarrow P\}) = \emptyset$、また特別な入力動作 $s?\omega(u)$ について $s?\omega(u) \notin \alpha(\{m \rightarrow P\})$ とする．プロセス Rが、動作 m の qによる忠実な部分計算であるとは、Rが以下のプロセスと qについて限定失敗集合等価 であることをいう．

$$\{\|_{c \in C} c \rightarrow R' \| s?\omega(u) \rightarrow Q\}$$

ここで R'はやはり、プロセス Pの条件 qによる忠実な部分計算で、Cおよび $s?\omega(u)$ を Rと共有する[3]．また Q は m の忠実なシミュレーションであるプロセスと qについて限定失敗集合等価であるとする．

ここで Cは Rの部分計算動作集合と呼び $PE(R)$ と表記する．また $s?\omega(u)$ は Rの起動信号と呼び $wake(R)$ と表記する．また Rが動作の部分計算であるとき、Rの停止信号 $halt(R)$ とは $halt(Q)$ のことを指す．

命題 2

[3]やはり定義 3.1 と同様に、後に導入する記法を用いれば、$PE(R) = PE(R')$ かつ $wake(R) = wake(R')$ と書ける．

プロセス $\{m \to P\}$ と初期条件 q について、以下のプロセスは $\{m \to P\}$ の q による忠実な部分計算 P' と等価である.

$$(R \| R') \backslash (PEI(R') \cup \{s'\}).$$

ここで、Rは m の qによる忠実な部分計算であり、$PEI(P') \cup PEI(R') = PE(R)$ かつ $wake(R) = wake(P')$. また $q \supset \mathsf{X}q'$なる q'とするとき、R'は P の q'による忠実な部分計算である. ここで $s' = halt(R) = wake(R')$ である.

命題 2 には直観的には、$\{m \to P\}$ の qによる部分計算は、動作 m の q による部分計算と、Pの q' (ただし $q \supset \mathsf{X}q'$) による部分計算に分割することができることを意味している. この分割をさらに Pの部分計算にも再帰的に適用することにより、プロセスの部分計算が動作の部分計算に分割され、並列に実行可能となる可能性を示している. このことは、逐次的なプロセスをパイプライン並列に実行することに対応する.

命題 1 および命題 2 では、逐次型プロセスのうちでも最初の分岐が無い $\{m \to P\}$ という形のプロセスを扱った. 一般的には逐次型プロセスは 2 章で述べたように 分岐を含む形で与えられることがありうる. このような一般的な場合のうち、$\{m_1 \to P_1 \| \cdots \| m_n \to P_n\}$ のように分岐の数が有限である場合は、本稿で述べた方法を拡張し、$1 \leq i \leq n$ について m_iのシミュレーション及び P_iの部分計算を並列に実行することにより、複数のプロセスに分割することができる. これは所謂 OR 並列性を引き出していることに対応する.

4 変換例の概要

この節では、先に述べた枠組みにもとづいて、与えられた逐次型プロセスを並列化するための変換方法の例について述べる.

変換手法の概要は以下のようなものとなる. $P \stackrel{\text{def}}{=} \{m_1 \to P_1 \| \cdots \| m_n \to P_n\}$ であるとき、この変換結果を

$$trans(P)$$

と表記することにする．$trans(P)$ は、初期化のためのプロセス $Init$ と、P に対応するプロセスネットワークを起動する $decomp(P)$ を起動する．

$decomp(P)$ は、各分岐 $m_i \rightarrow P_i$ に対応する n 個のプロセス $Node_i$、及び外部との通信の整合をとるためのゲートウェイとして働くプロセス GW_j 等の補助的なプロセスを起動する．GW_jは、外部との通信に使われるチャネル名の数だけ起動される． $Node_i$ はさらに、各 m_iの実行を処理するプロセス $Act(m_i)$ と各 P_iに対応する $decomp(P_i)$ に分かれる．

これらのプロセスは起動された後実行が進むにしたがって、木状のトポロジーをもつプロセス・ネットワークを構成する．この木構造は、Pをその定義にしたがって展開して得られる木構造に対応するものである．すなわち木の根から子孫に至る道は、Pの具体的な計算のトレースをあらわしている．Pが起動されると、実行中にどの通信動作を行なうかによって親から子へ至る枝のひとつが選ばれる．この動作を、変換後のプロセス $trans(P)$ では $token$ という特別な制御用の信号を根から子孫に向けて送ることによってシミュレートする．

$trans(P)$ は以下のような形のプロセスとして定義される．

$$trans(P) \stackrel{\text{def}}{=} Init \| decomp(P)$$

$$\begin{aligned} decomp(P) \stackrel{\text{def}}{=} \{ Getinput \rightarrow (\; & Gettoken \| Forwardinput \| \\ & GW_1 \| GW_2 \| \cdots \| GW_l \| \\ & Node_1 \| \cdots \| Node_n) \} \end{aligned}$$

$$Node_i \stackrel{\text{def}}{=} Act(m_i) \| decomp(P_i)$$

各プロセスのはたらきは、以下のようなものである。

- $Init$: $decomp(P)$ に対して $token$ を発行する．これによって $decomp(P)$ がアクティヴな状態となり、外部との通信動作を行なうことができるようになる．

- $decomp(Q)$: Q が起動される以前の計算の結果を $Getinput$ で受けとり、プロセス Q に対応する木状のネットワークを起動する．このネットワークは、$token$ が届くまでに、$Getinput$ によって受け取った計算の途中結果を用いて Q の起動後に行なわれる計算をあらかじめ進めておく．これによって、Q を起動以前に部分計算している．
- $Gettoken$ ：起動された $decomp(P)$ は、このプロセスが $token$ を受けとることによって、実行がコンティニュエーション Pが呼び出されたところまで進行したことを認識する．受け取った $token$ は、次の世代の選ばれた枝に、$Act(m_i)$ によって転送される．
- GW_j : $trans(P)$ によって起動された各プロセスと Pの外部との同期及び通信の整合をとるためのプロセスである． 外部との通信に使われる j番目のチャネルが出力チャネルであるか入力チャネルであるかによって動作が場合分けされる．
- $Forwardinput$: 以前の計算結果を受け取るという通信動作 ($Getinput$ と同様) を行ない、子孫の $decomp$ プロセスに継続的に転送し続ける．転送された中間結果によって、Pのコンティニュエーションの部分計算が継続的に続けられる．
- $Act(m_i)$: m_iが入力動作の場合と出力動作の場合で、動きが異なるが、いずれの場合も Pの外部との通信動作によってどの枝が選ばれるか、すなわちどの子孫に $token$ が転送されるかを制御する．

  - m_iが出力動作 $c!f(x)$ であった場合:

    $token$ が届くまでは、前のレベルから送られてきた入力 x の値をもとに $f(x)$ の値をあらかじめ計算し、$decomp(P_i)$ に結果を先まわりして送っておく．

    $token$ が届いたら、出力 $f(x)$ を外部に公開して貰うため GW_iに送る． GW_i が出力を実際公開したことを知らせてきたら (すなわち、m_iの枝が選ばれて実行されたら)、$token$ を $decomp(P_i)$ に送る．

  - m_iが入力動作 $c?x$ であった場合:

*token*を受け取ったら、外部からの通信イベントを待つようにGW_iに知らせる．m_iに外部からの通信イベントが実際に届いたら、*token*と外部から届いた入力値を$decomp(P_i)$に送る．

以上のようにして、もとのPと等価な動作を行なうプロセスの定義を得ることができる．

5 考察

本稿では逐次的なプロセスが与えられたとき、それを部分計算の手法を用いて並列なプロセスに分割する方法の一般的な枠組みを与えた．実際は、具体的な部分計算手法が与えらた際、それが用いられるのは定義3.3のQを求める部分である．その後に命題2により、命題1のR, Qが構成できる． このように本稿で述べた方法は、並列化のための具体的な変換手法ではなく、与えられた部分計算手法を並列化に応用するための方法を示したものである．

したがって本稿の方法を用いた並列化手法は、具体的な部分計算手法の中身に依存することなく、その正しさが示されるという点に特徴がある．また既存の(並列化とは関係のない)部分計算手法を、新たな並列化のための変換手法のに応用できる可能性を示しているという点でも特徴があるといえる．

本稿で述べた変換では、逐次的に実行されていた場合には内部変数に格納されて暗黙のうちにコンティニュエーションに送られていた値を、変換後はプロセス間の通信動作を用いてコンティニュエーションに対応するプロセスに送っているため、通信のオーバーヘッドが生ずる．このように機械的な並列化が実行の効率化にどのような場合に有効であるかについての評価は、今後の課題である．

謝辞: 岡山大学山崎進教授以下知能情報処理講座の皆様には、有益な議論と研究上の御支援をいただきました．記して感謝します．

参考文献

[Ho85] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall (1985)

[Mu92] 村上, 応答型並行プロセスのための時相論理式を用いた部分計算法, ソフトウェア科学会, プログラム変換・合成研究会,(1992)