Vol. 37 No. 4　　情　報　処　理　　Apr. 1996

解　説

並行計算の理論の最近の動向

2. π-計算とその周辺†*

吉田展子††　久保　誠††　本田耕平†††

1. 名前通信プロセス計算の背景

π-計算を含む名前通信プロセス計算（以下π-計算族と呼ぶ）は，過去20年間ほど研究されてきたプロセス計算の最近の発展である．1990年にMilner, Parrow, Walkerによって提案されたπ-計算[8]は，CCS[5]の純粋な同期のみを基礎とする通信機構に，名前を送信する機能を加えることによって，従来困難であったプロセス間のリンクや構造の動的変化を簡潔に表現することを可能とした形式系である．

この名前を直接通信するという概念をはじめてCCSに形式的に導入したのはEngbergとNielsenであったが，彼らの提案したECCSは値として名前・変数・コンスタントなどを含み複雑であったため，結局公に発表されることはなかった．一方，すでに70年代後半からCCSの創始者Milnerは，この名前通信の重要性について強く意識していた．79年にMilnerは先のNielsenとArhausで議論し，理論的困難さから一度は断念したものの，プログラミング言語に実際数多く見られるポインタやチャネルなどを用いた名前付けによる参照・識別などのアイデアを重視し，ECCSをさらに形式的に整備かつ理論的に強化，形式系の構成を行った．π-計算が簡潔に形式化された欠くことのできない背景として，Milnerが影響を受けていたActorモデルにおける「計算実体はすべてActorである」という概念的な統一性[1]があげられる．π-計算では受け渡されるものはすべて名前として単一化され，名前でアクセスされるものはプロセスとして統一的に表現される．識別子の通信を含むが最終的にはActor自体を送信することによって計算が行われるActorモデル[1]とは異なり，π-計算ではプロセスを送信することなく，（プロセスを参照する）名前のみを送信するという単純化が行われた．このため，Actorモデルでは困難であった代数的取扱いが可能になり，様々な角度からの発展を遂げて，今日に至った．最近では実際に並行プログラミング言語設計[11),13)]や検証系[14)]の基礎として用いられたり，関数型計算の意味論の問題を解決する手助けになるなど[4)]，幅広い研究がなされている．

以下，2.ではπ-計算族の標準的な構文を定義し，3.ではその意味論について簡単に議論する．続く4.ではπ-計算の名前通信機構を使った計算表現について述べ，最後に5.で全体の総括を行う．

2. π-計算の基本要素

2.1 名前通信の概念

π-計算[8]は，名前通信によってプロセス間における動的なネットワークの変化を実現できるように，CCS[5]を拡張した形式系である．通信で計算システムを表現する最も単純な形式系として提案されたCCSでは，名前渡しを取り扱えなかったため，プロセス間のアクセス関係の動的な変更を表すことができなかった．たとえば，以下のようなプロセスを考えてみよう．

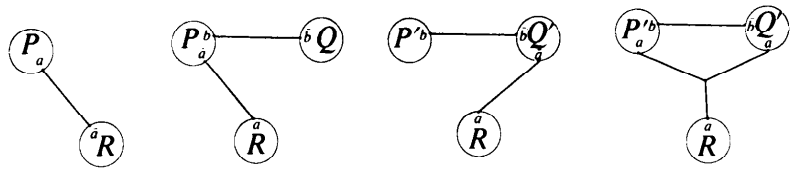
$$P \stackrel{\text{def}}{=} \bar{a}5.P' \quad R \stackrel{\text{def}}{=} a(x).R'$$

ここでPはリンクaに値5を送った後P'になるプロセスを，Rはリンクaからある値vを受け取り，R'内に自由に存在するxを値vで置き

† π-Calculi and their Semantics by Nobuko YOSHIDA, Makoto KUBO (Keio University, Department of Computer Science) and Kohei HONDA (Manchester University, Department of Computer Science).

†† 慶應義塾大学理工学研究科計算機科学専攻
††† マンチェスター大学計算機科学科

* この研究は日本学術振興会および英国EPSRCから補助を受けている．また，一部慶応義塾大学大型研究助成の援助を受けている．



(a)　(b)　(c)　(d)

図-1

換える($R'\{v/x\}$)プロセスを示している．この2つのプロセスを合成し，aがPとRのみに限定された局所的な名前であるとすると，プロセス全体は

$(a)(P|R)$

と書ける．これを図で示すと，**図-1**(a)のようになる．ここでPとRの間に同期通信が行われることにより，上記のプロセスは

$(a)(P'|R'\{5/x\})$

と変化するが，リンクが新しく生成されていない．つまり，図-1(a)のままか，あるいはリンクが消去されてしまっているかどちらかである．これがCCSにおける典型的な通信の例である．一方π-計算では，通信に用いられるポート（上記a）自体が動的に受け渡される．まず以下の3つのプロセスを考える．

$P \overset{\text{def}}{=} \bar{b}a.\bar{b}5.P' \quad Q \overset{\text{def}}{=} b(y).b(z).\bar{y}z.0$

$R \overset{\text{def}}{=} a(x).R'$

Pはリンクbにまず名前aを送信し，次に5を渡してP'になる．Rはリンクaからある値vを受け取ったのちR'になる．Qはリンクbから最初にある名前uとある値vを受け取り，リンクuから値vを送信する．この3つのプロセスを結合し，aとbを局所的な名前であるとすると，項表現は

$(a)(b)(P|Q|R)$

のようになり，**図-1**(b)のように描かれる．ここでいくつか重要な点があげられる．第1に，PとRはリンクaを，PとQはリンクbを共有しているが，Q内にはaが現れていない．つまり，QはRとのリンクを最初は保持してない．第2に，Pはリンクとして使用したa自身を値として通信していることである．プロセス全体は2度の通信後に，

$(a)(b)(P'|\bar{a}5.0|a(x).R')$

のようになり，PとRの間で局所的であったリンクaがQ'とR間に新たに生成される．この項は，仮にP'にaが出現していないと仮定すれば，PからQへとリンクの解放が行われたことを示し（**図-1**(c)），一方，もしP'にまだaが出現しているとすれば，局所的であったリンクが拡大された状態を表現している（**図-1**(d)）．このようにπ-計算では，名前通信機構を兼ね備えることによって，動的にネットワーク構造が変化する並行事象のモデル化を簡潔に行うことができる．

2.2 π-計算族の構文

上で議論されたπ-計算の**構文**（syntax）を正式に定義しよう．$a, b, c, \cdots$は名前を表すメタ変数，$P, Q, \cdots$はプロセスを表すメタ変数，A, B, Cをプロセス変数を表すメタ変数，$\tilde{a}, \tilde{b}, \cdots$は名前の列を表すメタ変数とすると，$\pi$-計算[8]の構文は以下のように与えられる．

$P::=\bar{a}b.P$	出力
$\mid a(x).P$	入力
$\mid P\|Q$	並行合成
$\mid (a)P$	名前制限
$\mid [a=b]P$	マッチング
$\mid P+Q$	外部非決定和
$\mid 0$	ヌル
$\mid A(x_1, \cdots, x_n)$	プロセス定義

先に述べたように，$\bar{a}b.P$はaに値bを出力しPになるプロセスであり，$a(x).P$はaから値vを受け取ると，P内のxを値vで置き換えるプロセスである．$a(x)$や$\bar{a}v$は**プレフィックス**（prefix）と呼ばれる．$(a)P$はP内の自由な名前aが局所的であることを意味する．$\mathcal{FN}(P)$をPに出現する自由な名前の集合とすると，$\mathcal{FN}(a(x).P)=(\{a\}\cup\mathcal{FN}(P))\backslash\{x\}$かつ$\mathcal{FN}((a)P)=\mathcal{FN}(P)\backslash\{a\}$である．$P|Q$は$P$と$Q$の並行合成を示している．マッチング（matching）

表-1 π-計算族の構文

	形式系			
	ν	π_r	π	多引数π
実体(出力) 実体(入力)	$\leftarrow av$ $a(x).P$	$\bar{a}v.P$ $a(x).P$	$\bar{a}v.P$ $a(x).P$	$\bar{a}[\tilde{v}].P$ $a(\tilde{x}).P$
並行合成	$P\mid Q$	$P\mid Q$	$P\mid Q$	$P\mid Q$
名前制限	$(a)P$	$(a)P$	$(a)P$	$(a)P$
ヌル	0	0	0	0
複写	$!a(x).P$	$!P$	$!P$	$!P$
非決定和			$P+Q$	$P+Q$
マッチング			$[a=b]P$	$[a=b]P$

$[a=b]P$ は，もし a と b が構文的に等しい名前ならば P になり，そうでないならば 0 になることを意味している．非決定和 (summation) $P+Q$ は非決定的に P か Q か選択されるプロセスであり，0 は何もプロセスがないことを示す．

ここまでの構文では有限な動作を行うプロセスしか構成できないが，再帰を含むプロセス定義 $A(x_1, x_2, \cdots, x_n)\overset{\text{def}}{=}P$ を許すことによって，無限に動作を続けるプロセスへの拡張が行われる．ここで $x_1, x_2, \cdots, x_n$ はすべて異なる名前で，$\mathcal{FN}(P)\subseteq\{x_1, x_2, \cdots, x_n\}$ であるとする．こうしてプロセス定義があたえられたとき，$A(y_1, \cdots, y_n)$ は，$P\{\tilde{y}/\tilde{x}\}$ を意味する*．一方，線形論理にヒントを得た別の無限プロセスの表現 $!P$ が参考文献 6) の中で提案され，これは**複写** (replication) と呼ばれている．この複写は無限に並行合成された P を意味する．再帰表現と複写どちらの方法を選択しても，通常の弱双模倣性 (3.参照) の範囲で他方を表現することができる．

さて，以上の構文を基盤として今日まで様々な π-計算の変形が提案されてきた．中でも上記からマッチングと非決定和をのぞいた構文 (以下では**π_r-計算**と呼ぶ) が Milner によって参考文献 6) で提示され，それによって，λ-計算をシミュレートできることが証明された (4.2 参照)．さらにこの π-計算から出力を示す $\bar{a}v.P$ を取り去り，メッセージを示す $\bar{a}v$ ($\leftarrow av$ とも書く) をかわりに加えることによって，非同期名前通信を表現した**ν-計算** (「非同期 π-計算」とも呼ばれる)* が参考文献 2) で提案されている．のちの 4. で示唆されるように，ν-計算は π_r-計算と同等の表現能力を持つ．その他に ν-計算を基礎として，π_r-計算と同等な計算表現能力を持つ並行コンビネータの研究がなされている[3]．

以上は構文上の縮小をともなう変形だが，逆の方向への変形として，π-計算の単一の引数 (monadic) による通信を複数の引数に拡張した**多引数** (polyadic) **π-計算**が参考文献 7) で提案されている．これは「よりプログラミング言語へ近づけた π-計算」と捉えることができる．多引数の値送信の導入にともない，操作的意味論の変更や引数の数の一致や型の整合性の検査などの問題が出現し，これらの研究もなされている[7]．以下に多引数 π-計算の構文定義を与える．

$$P::=\bar{a}[b_1b_2\cdots b_n].P \quad \text{出力}$$
$$\mid\ a(x_1x_2\cdots x_n).P \quad \text{入力}$$
$$\mid\ \text{以下同じ}$$

この多引数 π-計算も，もともとの π-計算でシミュレートできることが知られている[7]．

最後にいままで定義した形式系を構成子別に**表-1** にまとめる．

3. π-計算の操作意味論

プロセスがどのように並行に相互作用を行い計算を進めるか，という操作の概念を形式的に表現した並行プロセス計算の操作的意味論は，関数型形式系には見られない並行プロセスの特性を探求するためにも重要となる．π-計算族の操作的意味論はいくつかの規則 (公理) から導出される，名前通信を考慮した遷移関係によって定義され，これらを基に動作意味論が導出される．以下では，構造合同性に基づく遷移関係を定義した後，π-計算族に一般的に定義される双模倣性を導入する．定義はもともとの π-計算に対して行い，その上で 2. の終りに述べた種々の変形に応じた変更について述べることとする．

3.1 構造合同性

2.1. で紹介した例からも類推されるように，いつも可簡約項が $(\lambda x.M)N$ のように隣り合って存在している λ-計算とは対照に，プロセスが独

* ここで $\{\tilde{y}/\tilde{x}\}$ は，$\tilde{x}$ と $\tilde{y}$ は同じ長さの名前列で，$\tilde{x}$ の中の名前は互いに異なるものとしたとき，(束縛を考慮した) 名前の同時複数代入を意味する．

* $\bar{a}v$ は π-計算における $\bar{a}v.0$ であるので，ν-計算は π_r-計算の厳密なサブセットとなっている．

立・並行に動作を行っている並行計算の構文的表現においては，可簡約項を形成するプロセス同士は以下のように遠く離れて存在するかもしれない．

$a(x).P|P_1|\cdots|P_n|\bar{a}v.Q$

しかし，これは各々のプロセスが独立に順番に関係なく並行合成されていることを表現しているので，

$a(x).P|\bar{a}v.Q|P_1|\cdots|P_n$

のように構文を読み変えてしまってもプロセスの集合をマルチセットと見なせば意味的に変化がない．このように λ-計算の α-変換と同様に，π-計算上に静的で自然な合同性をあらかじめ導入することで，遷移および簡約関係の操作に関する規則の定義が単純化された[6]．π-計算の項上の**構造合同性** (structual congruence) は，以下の条件を満たす最小の合同関係である．

(i)　P と Q が α-同値ならば，$P\equiv Q$

(ii)　$P|Q\equiv Q|P$　$P|(Q|R)\equiv(P|Q)|R$　$P|0\equiv P$

(iii)　$(a)(b)P\equiv(b)(a)P$　$(a)0=0$

(iv)　$(a)(P|Q)\equiv(a)P|Q$ ただし $a\notin\mathcal{FN}(Q)$

(v)　$P+Q\equiv Q+P$　$P+(Q+R)\equiv(P+Q)+R$　$P+0\equiv P$

(vi)　$[a=a]P\equiv P$

(vii)　もし $A(\tilde{x})\overset{\text{def}}{=}P$ ならば，$A(\tilde{y})\equiv P\{\tilde{y}/\tilde{x}\}$.

条件(ii)(iii)(v)は，並行合成，名前制限，非決定和に対し，0 を単位元とする可換・結合則が成立するという規則である．条件(iv)は制限された名前 a がプロセス Q に現れないならば，その制限を外部に出したプロセスと同等であることを意味している．条件(vi)は，マッチングに関する規則で，$[a=b]P$ における P は，a と b が一致したときに限り，動作をすることを示す．プロセス定義に関する条件(vii)は，もしも複写 $!P$ を用いるなら，複写の規則

$!P\equiv P|!P$

で置き換えることが可能である．つまり $!P$ は $\equiv$ を適用していくと，P を無限回複写したプロセス $P|P|P\cdots$ と同等であることが分かる．

3.2 遷移関係

CCS などのプロセス代数における外部からの観測という概念は π-計算の動作意味論の定義にも応用され，観測された通信事象をラベルとして定義した**ラベル付き遷移関係** (labelled transition relation) が広く採用されている．ラベル付き遷移関係は

$$P \xrightarrow{l} Q$$

のような形で表現され，「P が動作 l を行った後，Q に変化した」と読まれる．π-計算のラベルは以下の構文で与えられる．

$l::=\tau \mid \bar{a}v \mid av \mid \bar{a}(v)$

τ は観測されないラベルを，$\bar{a}v$ と av は，プロセスの出力と入力を表すラベルであり，$\bar{a}(v)$ は束縛出力を示す．ここで (v) は束縛名である*．$\mathcal{N}(l)$，$\mathcal{BN}(l)$ をそれぞれ l に出現する名前，束縛された名前の集合とする．また，ラベル l とプロセス P の間に $\mathcal{BN}(l)\cap\mathcal{FN}(P)=\emptyset$ が成り立つとき，l は P に対し適切 (relevant) であるという．π-計算の遷移関係は以下の規則を満たす最小の関係として定義される．

$$\text{OUT}: \bar{a}v.P \xrightarrow{\bar{a}v} P$$

$$\text{IN}: a(x).P \xrightarrow{av} P\{v/x\}$$

$$\text{COM}: \bar{a}v.P|a(x).Q \xrightarrow{\tau} P|Q\{v/x\}$$

$$\text{PAR}: P \xrightarrow{l} P' \Rightarrow P|Q \xrightarrow{l} P'|Q$$

$$\text{SUN}: P \xrightarrow{l} P' \Rightarrow P+Q \xrightarrow{l} P'$$

$$\text{OPEN}: P \xrightarrow{\bar{a}b} P' \Rightarrow (b)P \xrightarrow{\bar{a}(b)} P'$$

$$\text{RES}: P \xrightarrow{l} P' \Rightarrow (a)P \xrightarrow{l} (a)P'$$

$$\text{STR}: P\equiv Q\wedge Q \xrightarrow{l} Q'\wedge Q' \equiv P'\Rightarrow P \xrightarrow{l} P'$$

ただし PAR，SUM では $\mathcal{BN}(l)\cap\mathcal{FN}(Q)=\emptyset$，RES では $a\notin\mathcal{N}(l)$，OPEN では $a\neq b$ とする．OUT，IN はプロセスの出力および入力が観測されたことに相当し，COM は観測されない τ-遷移が行われていること，すなわち基本的な計算遷移を表現する規則である．π-計算では出力・入力両プロセスが同期通信を行うことで名前が受け渡されることを示している．$\xrightarrow{\tau}$ は $\longrightarrow$ とも書かれる**．また，OPEN では通信の結果，名前制

* 対称性の点からは新しい名前を受けとる束縛入力 $a(v)$ もあってよいのだが，実際はこれを入れて構成した遷移関係は入れない遷移関係と同等の動作意味論を導出する．

** $\rightarrow$ は λ-計算の簡約関係に相当する．

限の範囲が外部に拡張されること（内部のみに知られた名前を外部のプロセスに渡すこと）を示している．PAR，SUM，RESは，各々並行，非決定的な選択，名前の制限に対応する規則で，CCSのそれぞれの規則と差異はない．

以上定義した遷移関係は，π_τ-計算，ν-計算の項集合がそれぞれπ-計算の部分集合であることから，関係の制限をとることによってそのままπ_τ-計算，ν-計算上の遷移関係と見なすことができる．たとえば，ν-計算では，

$$\mathrm{COM}_\nu:\ a(x).P|\bar{a}v \xrightarrow{\tau} P\{v/x\}$$

となる．一方，多引数π-計算の計算規則は，たとえばCOM則は

$$\bar{a}[v_1\cdots v_n].P|a(x_1\cdots x_n).Q \xrightarrow{\tau} P|Q\{\bar{v}/\bar{x}\}$$

となり，その他の規則についてもラベルを多引数に変更することで，同様に構成することができる．

3.3 動作意味論

一般のプロセス代数と同様，2つのプロセスの等価性を外部からのラベルの観測という概念に基づいて定義することができる．以下では上で定義された名前通信遷移に，CCSの**双模倣性**(bisimilarity)の概念を拡張し，動的な振舞いを行うプロセスの等価性を定義する*．ここで動作意味論の背景に立ち戻って，プロセスの意味を，観測し得る動作列に基礎をおくと考えるならば，τ-遷移を除いた動作を基礎に意味論を構成することは自然である．実際にプロセスの動作を解析し等価関係を論じる際は，見えない内部遷移を除いて考えることが多い（プログラムの最適化の問題や次章の計算表現の例を考えよ）．よって一例として，ここでは特にτ-動作を捨象する弱双模倣性を導入する．

$\overset{l}{\Longrightarrow}$は$l=\tau$ならば$\to^*$を，他の場合は$\to^*\xrightarrow{l}\to^*$を示すものとする．$\pi$-計算の項上の対称な関係$\mathcal{R}$は以下の条件を満たすとき**弱双模倣関係**（weak bisimulation）と呼ばれる．すなわち，もしも$P\mathcal{R}Q$であり，Qに対して適切なlに関して$P\xrightarrow{l}P'$ならば，いつでもあるQ'があって，$Q\overset{l}{\Longrightarrow}Q'$かつ$P'\mathcal{R}Q'$である．この最大の関係は**弱双模倣性**（weak bisimilarity）と呼ばれ$\approx$と書かれる．例として$\Omega\overset{\mathrm{def}}{=}(c)(\bar{c}v.0|!c(x).\bar{c}v.0)$とすると，$\Omega\to\Omega\to\cdots$であり，これは文脈になにも影響を与えないプロセスと考えられ，実際に$\Omega\approx 0$となる．

$\approx$がすべての弱双模倣性の和であり，かつ同値関係であることは，それほど困難なく証明することができる[8]．π_τ-計算，ν-計算，多引数π-計算（多引数ラベルによる遷移関係を用いる）についても弱模倣性が同様に定義できる．特に前2者の弱双模倣性は，π-計算の$\approx$をそれぞれの項集合へ制限した関係と一致している．またCCS[5]と同様に，それぞれについてHennessy-Milner論理（様相論理の一種）による完全な性格付けが可能である[9]．有限プロセスにおけるπ-計算および多引数π-計算の等価理論による$\approx$の公理化は，従来の手法の延長で可能であると思われる（ただし参考文献10)参照）．一方，π_τ-計算およびν-計算では非決定和がないため，いわゆる展開則（並行合成を非決定和に変換する）を用いることができないことから，双模倣性の公理化は知られていない．また弱双模倣性の他に，強双模倣性や等価性の拡張に関する研究が多くなされている．最新の結果を含むこれらの研究については，本稿の続編に譲る．

4. 並行プロセスの表現例

本章ではπ-計算による並行計算のプログラミング例をいくつか示す．これによって前章までに抽象的に与えられたπ-計算の構成子の意味が具体的な計算表現を通じて明確になってくるだろう．またπ-計算で与えられる並行計算のための最小限の構成子を用いて重要な計算構造を表現することは，相互作用によるプログラミングの基礎的な形をπ-計算を通じて示すことでもある．その意味で，以下ではπ-計算族の中で最小のν-計算の構文から出発し，そこからどのように（多引数π-計算を含む）複雑な計算動作が構成されていくかを示すことにしよう．

4.1 直列化と分岐

関数的・並行的な動作を含む任意のプログラミングを並行事象の中で実現する際には，まず計算事象の順序を制御することが不可欠となる．2.で

* 以下に定義する関係は参考文献8)の**早期弱双模倣性**（early bisimulation）である．現在はこの早期双模倣性が一般的に用いられている．

述べたように，ν-計算には直列的な構成子は $a(x).P$ しか存在しないが，並行事象で一連の名前の列を順序を保ちつつ受け渡す表現，すなわち多引数 π-計算のプレフィックスの同期的動作を実現することができる．以下で z, c, y は新しい名前とし，

$$\bar{a}:[v_1\cdots v_n].P \overset{\text{def}}{=} (c)(\bar{a}c|c(z).(\bar{z}v_1|c(z).(\cdots|c(z).(\bar{z}v_n|P)\cdots)))$$

$$a:(x_1\cdots x_n).Q \overset{\text{def}}{=} a(y).(c)(\bar{y}c|c(x_1).(c)(\bar{y}c|\cdots(c)(\bar{y}c|c(x_n).Q)\cdots))$$

と定義する．このとき，任意の n について，

$$\bar{u}:[v_1\cdots v_n].P|v:(x_1\cdots x_n).Q \to\approx P|Q\{v_1\cdots v_n/x_1\cdots x_n\}$$

ここで，$\approx$ は **3.** で定義した弱双模倣性である．$n=1$ のときがオリジナルの π 計算にあたることに注意．また，上の写像では新しく生成された名前 c が Actor モデルにおける継続[1]の役割を果していることが分かる．

これをさらに応用することにより，**分枝表現** (branching)，すなわち $\bar{a}:\mathrm{in}_n^i[\tilde{v}].P$ は，$Q_1,\cdots,Q_n$ のうち i 番目を選択し，Q_i の自由に出現する $\tilde{x}_i$ に同じ長さの名前の列 $\tilde{v}$ を送信する，がコーディングできる．これはオブジェクト指向計算のメソッド分岐の基礎的な形と考えることができる．

$$\bar{a}:\mathrm{in}_n^i[\tilde{v}].P \overset{\text{def}}{=} (c)(\bar{a}c|c:(z_1z_2\cdots z_n).\bar{z}_i:[\tilde{v}].P)$$

$$a:\&_{i=1}^n(\tilde{x}_i).Q_i \overset{\text{def}}{=} a(z).(c_1c_2\cdots c_n)\bar{z}:[c_1c_2\cdots c_n].(c_1:(\tilde{x}_1).Q_1|\cdots|c_n:(\tilde{x}_n).Q_n)$$

ただし，$c, \tilde{c}, \tilde{z}$ は新しい名前とする．ここでも新しい名前の生成機能を用いることにより，任意の n, i において，

$$\bar{u}:\mathrm{in}_n^i[v_1\cdots v_k].P|u:\&_{i=1}^n(x_{i1}\cdots x_{ik}).Q_i \to\approx P|Q_i\{v_1\cdots v_k/x_{i1}\cdots x_{ik}\}$$

が満たされることが確認できる．多引数受渡しも分岐表現も最終的に ν-計算の構文内で実現されていることに注意．以下では特に

$$a:\&_{i=1}^2(\tilde{x}_i).Q_i \overset{\text{def}}{=} a:[(\tilde{x}_1).Q_1]\&[(\tilde{x}_2).Q_2]$$

と記述する．

4.2 高階関数と並行プロセスの表現

直列化と複写を組み合わせることで，π-計算は高階関数を表現することができる．このような例として λ-計算のコーディング[6]を示す．以下，$\underline{x}$ は λ-計算の変数を示すものとし，ある変数 $\underline{x}$ に名前 x が対応するものとする．

$$[\![\lambda\underline{x}.M]\!]u \overset{\text{def}}{=} u:(xz).[\![M]\!]z$$

$$[\![\underline{x}]\!]u \overset{\text{def}}{=} \bar{x}u$$

$$[\![MN]\!]u \overset{\text{def}}{=} (cc')([\![M]\!]c|!c'(z).[\![N]\!]z|\bar{c}:[c'u].0)$$

このとき，以下のことが分かる．

$$[\![(\lambda\underline{x}.M)N]\!]u \to^*(c')([\![M]\!]u\{c'/x\}|!c'(z).[\![N]\!]z) \approx [\![M\{N/\underline{x}\}]\!]u$$

再左項と中間項は $\approx$ の関係で結ばれるので，

$$[\![(\lambda\underline{x}.M)N]\!]u \approx [\![M\{N/\underline{x}\}]\!]u$$

が分かり，β-等価性がコーディング上で成立していることが分かる．なお，このことを使って（あるいは直接的なコーディングによって），ν-計算（したがって π-計算族一般）が $\approx$ の範囲で全計算可能関数を表現できることが分かる．

関数による計算の表現を可能とする一方，プロセス計算，あるいは一般的には並行プログラミング言語では，プロセスの共有や非決定性の概念を積極的に採り入れることで，より現実世界に即した計算構造の記述法を提供してきた．以下，Church-Rosser 性を基礎にする関数的なプログラミング言語ではプリミティブから構成することが困難である 2 つの例を示す．まず，第 1 の例として並行 Or のコーディングを示す．真偽値と状態を変えないリンクを

$$\mathbf{T}(a) \overset{\text{def}}{=} \bar{a}:\mathbf{in}_2^1[\varepsilon].\mathbf{T}(a)$$

$$\mathbf{F}(a) \overset{\text{def}}{=} \bar{a}:\mathbf{in}_2^2[\varepsilon].\mathbf{F}(a)$$

$$\mathrm{Fw}(ab) \overset{\text{def}}{=} a(x).(\bar{b}x|\mathrm{Fw}(ab))$$

と定義すると（ただし，ε は空列を表す），And, Or そして並行 Or について以下のように簡素なコーディングが存在する．

$$\mathrm{AND}(bb_1b_2) \overset{\text{def}}{=} b:[(\varepsilon).\mathrm{Fw}(b_1b_2)]\&[(\varepsilon).\mathrm{Fw}(bb_2)]$$

$$\mathrm{OR}(bb_1b_2) \overset{\text{def}}{=} b:[(\varepsilon).\mathrm{Fw}(bb_2)]\&[(\varepsilon).\mathrm{Fw}(b_1b_2)]$$

$$\mathrm{POR}(b_1b_2b_3) \overset{\text{def}}{=} \mathrm{OR}(b_1b_2b_3)|\mathrm{OR}(b_2b_1b_3)$$

実際，たとえば Ω など停止しないプロセスに

対しても，$(cc')(\text{POR}(cc'a)|\mathbf{T}(c)|\Omega)\approx\mathbf{T}(a)$ となる．

もう1つの例として，状態を持つ計算構造であるセル（並行に読み書きが可能な変数とも考えられる）のコーディングを示す．セルは「読み」と「書き」の2つの選択肢（いわゆるメソッド分岐に対応する）を持つ．対応するエージェントにリードとライトがいて，リードは「読み」の選択肢を選んで値を受け取り，ライトは「書き」を選択して値を渡す．複数のリードやライトが競合することから非決定性がうまれる．

$$\text{CELL}(av)\overset{\text{def}}{=}a:[(c).\bar{c}v.\text{CELL}(av)]\&[(x).\text{CELL}(ax)]$$

$$\text{READ}(a(x).P)\overset{\text{def}}{=}(c)\bar{a}:\mathbf{in}_1^2[c].c(x).P$$

$$\text{WRITE}(aw)\overset{\text{def}}{=}\bar{a}:\mathbf{in}_2^2[w].0$$

このとき，たとえば

$$\text{CELL}(av)|\text{READ}(a(x).P)\rightarrow\approx\text{CELL}(av)|P\{v/x\}$$

$$\text{CELL}(av)|\text{WRITE}(aw)\rightarrow\approx\text{CELL}(aw)$$

選択肢を増やし，相互作用をより複雑にしていくことによって，いくらでも複雑な動作が構成できることが容易に分かるであろう．

π-計算における興味深い計算表現例は数多くあるが，誌面の関係上以上にとどめる．これらの例のみからでも，π-計算族が関数型計算を内部に埋め込むことができ，また，実践的な応用の基礎となりうるだけの並行事象を十分記述する能力を備えていることが分かるだろう．実際に今まで示したような構成を，プログラミングにより即した言語プリミティブとして拡張・活用し，さらに通信における型概念を導入することによって，π-計算族を基礎とした様々な並行プログラミング言語（たとえば $\mathcal{L}$[13]，PICT[11]など）が提案され，現在盛んに研究が行われている．

5. 結　論

最後に全体を振り返り，特に重要と思われる事項について総括する．

- π-計算族はCCSを背景として誕生した名前通信を計算機構として備えた並行プロセス計算であり，全体としては構文の小さい順に表-1のような構成になっている．
- π-計算族の独自の動作的意味論が，双模倣性を中心に様々な角度から研究され，計算事象の埋め込みの証明や並行プログラミング言語の意味論的基礎に使用されている．
- π-計算によって，λ-計算や様々な並行プロセスの動作の表現が可能である．特にこれらの表現は従来知られているπ-計算族の最小の構成であるν-計算に構文を制限しても可能である．
- π-計算はポインタやアドレスを受け渡すことで自分自身で状態を動的に変化させるアイデアを基盤として構築されていることから，計算プリミティブの拡張・改良を重ねながら，並行オブジェクト指向言語などの理論的基礎として将来用いられていく可能性が広く探求されている．

なお，本稿では，初めての紹介という性格上，π-計算の理論的な要ともいえる動作意味論に関する様々な重要な結果については触れることができなかった．これらに関しては続編として掲載予定である．その他にも本稿で取りあげていない重要な理論的項目として，Sangiorgiによるπ-計算内での項渡しプロセス計算の埋め込み[12]や参考文献7)で提案されている多引数π-計算の**ソーティング**（sorting）という型概念，λ-計算のSKコンビネータ表現と対応する並行コンビネータ[3]などがある．これらについてはまた他日を期したい．

最後にπ-計算族についての研究はまだはじまったばかりである．興味深い結果や応用例が続々と提出される一方で，非決定和やマッチングとそれを除いたπ-計算族との厳密な関係や，プログラミング言語として実際に実装したときの効率に関する問題点など，まだ明らかにされていない点が多く残されている．これからさらに理論面からも実践面からもこれらの形式系がたくさんの人々の手によって強化され，さらに発展していくことを希望して，本稿を締めくくりたい．

謝辞　本稿の執筆に関し，構成その他の面で有用なご示唆をいただいた村上昌己先生および閲読者の皆様に感謝します．また，所真理雄先生の日頃の励しとご指導に感謝します．

参考文献

1) Hewitt, C., Bishop, P. and Steiger, R.: A Universal Modular ACTOR Formalism for Artificial Intelligence, Proc. of the 3rd International Joint Conference on Artificial Intelli-

gence, pp. 235-245 (1973).
2) Honda, K. and Tokoro, M.: An Object Calculus for Asynchronous Communication, Proc. ECOOP'91, LNCS 512, pp. 133-147, Springer-Verlag (1991). 日本語版: 本田耕平, 所真理雄: 並行オブジェクト計算のための形式系, コンピュータソフトウェア, Vol. 9, No. 2 (Mar.1992).
3) Honda, K. and Yoshida, N.: Combinatory Representation of Mobile Rrocesses, Proc. 21th. Annual Symposium on Principles of Programming Languages, pp. 348-360, ACM press (1994).
4) Hyland, M. and Ong, L.: Pi-calculus, dialogue games and PCF, 12 pages, 7th Annual ACM Conference on Functional Programming Languages and Computer Architecture (1995).
5) Milner, R.: Communication and Concurrency, Prentice Hall (1989).
6) Milner, R.: Functions as Processes, Mathematical Structure in Computer Science, 2(2), pp. 119-146 (1992).
7) Milner, R.: Polyadic π-Calculus: a Tutorial, Proceedings of the International Summer School on Logic Algebra of Specification, Marktoberdorf (1992).
8) Milner, R., Parrow, J. G. and Walker, D. J.: A Calculus of Mobile Processes, Information and Computation, 100(1), pp. 1-77 (1992).
9) Milner, R., Parrow, J. and Walker, D.: Modal Logics for Mobile Processes, ECS-LFCS-91-136, University of Edinburgh (1991).
10) Parrow, J. and Sangiorgi, D.: Algebraic Theories for Name-Passing Calculi, Research Report ECS-LFCS-93-262, Department of Computer Science, University of Edinburgh (1993).
11) Pierce, B. and Turner, D.: Concurrent Objects in a Process Calculus, Proc. TPPP'94, LNCS 907, Springer-Verlag, pp. 187-216 (1995).
12) Sangiorgi, D.: Expressing Mobility in Process Algebra: First-Order and Higher-Order Paradigms, PhD thesis, CST-99-93, Department of Computer Science, University of Edinburgh (1993).
13) Takeuchi, K., Honda, K. and Kubo, M.: An Interaction-based Language and its Typing System, Proc. PARLE'94, LNCS 817, Springer-Verlag, pp. 398-413 (1994).
14) Victor, B. and Moller, F.: The Mobility Workbench: A Tool for the Pi-Calculus, Research Report ECS-LFCS-94-285, Department of Computer Science, University of Edinburgh (1994).

(平成7年5月9日受付)

吉田 展子

1969年生. 慶応義塾大学電気工学科卒業. 1994年同大学院理工学研究科計算機科学専攻修士課程修了. 現在, 同大学院同学部博士課程在学中. 1995年6月より, 慶応義塾大学とマンチェスター大学のSplit-PhD studentとして, 英国に滞在中. 学術振興会特別研究員. 並行プロセス計算と関数型計算 (λ 計算) の基礎理論, 特に, その意味論研究に興味を持つ.

久保 誠

1968年生. 1991年慶応義塾大学電気工学科卒業. 1993年同大学院理工学研計算機科学専攻修士課程修了. 現在, 同大学院同学部博士課程在学中. 並行プロセス計算を基にしたプログラミング言語とコストに関する理論に興味を持つ.

本田 耕平

1992年慶応義塾大学大学院理工学研究科計算機科学専攻修士課程修了. 1994年同大学院理工学研究科計算機科学専攻博士課程修了. 現在, マンチェスター大学計算機科学科勤務. 並行計算基礎論, π 計算, ゲーム意味論などに興味を持つ.