プログラミング-言語・基礎・実践- 21－18
(1995. 3. 10)

名前渡しを含む並行プロセスの合成

奥畑 彰夫　　村上 昌己

岡山大学工学部情報工学科
岡山市津島中3丁目1番1号

e-mail:{okuhata,murakami}@momo.it.okayama-u.ac.jp

あらまし　本稿ではプロセス論理式の集合からπ-計算のプロセスを合成する手法を提案する．L を論理式の集合，A を論理式，また L に含まれる論理式をすべて満たすプロセスを P とする．このときプロセス合成は，P と A から論理式の集合 $L \cup \{A\}$ を満たすプロセスを構成する手続きの繰り返しとみなせる．この手続きを，プロセスを表わす木の再構成を行なうものとして形式化する．本手法では並行演算子を含むプロセスの合成が可能でなる．またπ-計算の名前渡しという特徴より，複数のプロセス間で動的なリンクの張り替えを行なうプロセスの合成が可能である．

キーワード　プログラム合成，プロセス代数，π-計算，プロセス論理

Synthesis Algorithm for Processes with Name-Passing

Akio Okuhata　　Masaki Murakami

Department of Information Technology, Okayama University
3-1-1 Tsushima-Naka, Okayama

e-mail:{okuhata,murakami}@momo.it.okayama-u.ac.jp

Abstract　We propose a method for synthesizing π-calculus processes from sets of process logic formulas. Let L be a set of formulas, A be a formula not in L and P be a process that satisfies all formulas in L. The basic idea of our method is that a process synthesis is the repetition of the procedure constructing a new process that satisfies $L \cup \{A\}$ from P and A. We formalize this procedure as a reconstruction of trees. We can synthesize a process with concurrent composition operator with this method. Our method also makes possible to synthesize mobile processes using the name passing feature.

key words　program synthesis, process algebra, π-calculus, process logic

1 はじめに

プロセス代数 CCS[2] は，複数のプロセスが互いに通信し合いながら並行に動作するシステムのプログラミングができる．よって CCS ではインタラクティブシステムや並行分散システム等の記述が可能である．このようなプロセス代数によるプログラムを合成する研究がいくつか報告されている．[1, 8] では，形式的仕様としてプロセス論理 [5] を用い，与えられたプロセス論理式から CCS のプロセスを合成する手法が報告されている．しかしこれらの手法では並行演算子により並行に動作するプロセスを合成できない．よってプロセス間で通信しながら並行に動作するシステムの記述ができない．

一方 CCS での通信は静的に記述されたポートを介して行なわれる．よって通信する相手をあらかじめ決めておく必要がある．従って実行が進むにつれて，通信する相手が動的に変わっていくようなシステムの記述には向かない．

π計算[5] は名前渡しという特徴を持つプロセス代数である． π計算の通信は名前を介して行なわれ，その通信の際に名前を送受信することが可能である．よって通信する相手が動的に変わっていくようなシステムの記述ができる．

本稿では，複数のプロセスが並行に動作し，かつこれらのプロセスの通信相手が動的に変化するプロセスを合成する手法を提案する．

本手法は $\langle\alpha\rangle$, $[\alpha]$, $\wedge$, **true**, **false** で構成されるプロセス論理を対象とし，仕様として与えらえたプロセス論理式の集合から，並行演算子を含む π計算のプロセスを合成する．任意のプロセス P は P を根としノードがプロセス，枝がプロセスの遷移を表わす（導出）木とみなせる．よって今まで入力された論理式を満たす導出木から，新たに入力された論理式も満たすように木を再構成する手続きを繰り返すことによりプロセスを合成することが可能である．本稿では， π計算のサブセットについてこの手続きを形式化する．

2 π計算とプロセス論理

2.1 π計算(π-calculus)

π計算[3, 4, 5] は CCS[2] を拡張したプロセス代数で名前渡し (name passing) という特徴を持つ．π計算では名前渡しを用いて動的なリンクの張り替えを行なうプロセスを記述する．本稿では π計算の名前渡しを含むサブセットを用いる [5].

π計算では CCS におけるポートや値をすべて名前 (name) によって表現する．

定義 2.1 動作 (action)，パス (path)

$\mathcal{N}$ を名前 (name) の無限集合とし，

$$\mathcal{L}_I = \{x(y) \mid x, y \in \mathcal{N}\}$$
$$\mathcal{L}_O = \{\overline{x}y \mid x, y \in \mathcal{N}\} \cup \{\overline{x}(y) \mid x, y \in \mathcal{N} \text{ and } x \neq y\}$$
$$\mathcal{L} = \mathcal{L}_I \cup \mathcal{L}_O$$
$$Act = \{\tau\} \cup \mathcal{L}$$

とする． Act の要素を動作と呼ぶ． Act の要素を有限個並べたものをパスと呼ぶ．また ε で空のパス， Act^* でパス全体の集合を表わす．

定義 2.2 プロセス (process)

プロセスの集合 $\mathcal{P}$ を以下の 5 つの演算子，無動作 (no action)，プレフィックス (prefix)，和 (summation)，並行 (composition)，制限 (restriction) を用いて再帰的に定義する．

$$0 \in \mathcal{P}, \qquad \alpha.P \in \mathcal{P},$$
$$P + Q \in \mathcal{P}, \quad P \mid Q \in \mathcal{P}, \quad (\nu y)P \in \mathcal{P}$$

ここで $y \in \mathcal{N}$, $\alpha \in Act$, $P, Q \in \mathcal{P}$ とする．

プロセスやパス中の名前に対して自由 (free)・束縛 (bound) などの概念を定義する [5]．例えば $\overline{x}y.0$ において x は自由な名前， y は束縛された名前である．プロセス P について $\mathrm{fn}(P)$, $\mathrm{bn}(P)$ でそれぞれ出現が自由，束縛である名前の集合を表わす．また $\mathrm{n}(P) = \mathrm{fn}(P) \cup \mathrm{bn}(P)$ とする． $\mathrm{fn}(P) \cup \mathrm{fn}(Q) \cup \cdots \cup \{x, y, \cdots\}$ を $\mathrm{fn}(P, Q, \cdots, x, y, \cdots)$ と省略して書く． $\mathrm{bn}(\cdot)$, $\mathrm{n}(\cdot)$ も同様の省略記法を用いる．また $\equiv$ は両辺が構文的に等しいことを意味する．

π計算では自由な名前に対する代入操作で名前渡しを表現する．プロセス P と 代入 $\sigma \equiv$

$\{y/x\}$ とする．$P\sigma$ は P 中の自由な名前 x を同時に y に置き換えることを意味する [5].

プロセス（パス）中の束縛された名前に関して，通常と同様な方法で α 変換を定義する．α 変換による同値関係を $\equiv_\alpha$ とする．

次にプロセス $\mathcal{P}$ の部分集合 $\mathcal{P}_{rn}$ を定義する．

定義 2.3 $\mathcal{P}_{rn}(\subset \mathcal{P})$ は以下の条件を再帰的に用いることで得られるプロセスの集合である．

- $0 \in \mathcal{P}_{rn}$
- $\alpha.P \in \mathcal{P}_{rn}$ if $\alpha \in Act$, $P \in \mathcal{P}_{rn}$ and $\mathrm{n}(\alpha) \cap \mathrm{bn}(P) = \emptyset$
- $P \mid Q \in \mathcal{P}_{rn}$, $P + Q \in \mathcal{P}_{rn}$ if $P, Q \in \mathcal{P}_{rn}$ and $\mathrm{fn}(P,Q) \cap \mathrm{bn}(P) \cap \mathrm{bn}(Q) = \emptyset$
- $(\nu y)P \in \mathcal{P}_{rn}$ if $y \in \mathcal{N}$, $P \in \mathcal{P}_{rn}$ and $\{y\} \cap \mathrm{bn}(P) = \emptyset$

補題 2.4 $P \in \mathcal{P} \Longrightarrow \exists Q \in \mathcal{P}_{rn}.P \equiv_\alpha Q$

この性質を利用して $\mathcal{P}$ のプロセスの標準形を定義する．

定義 2.5 プロセスの rn 標準形

プロセスを P とする．プロセス Q が 2 つの条件 (1) $Q \in \mathcal{P}_{rn}$，(2) $P \equiv_\alpha Q$ を満たすとき，Q を P の rn 標準形という．

任意のプロセスに対してその rn 標準形は一意に定まらない．

π 計算のプロセス $\mathcal{P}$ の意味はラベル付き遷移システム (labeled transition system, lts) によって与えられる [5]．本稿では [5] の $\mathcal{P}$ 上の遷移規則 (transition rule) ではなく，次の $\mathcal{P}_{rn}$ 上の遷移規則を用いる．

定義 2.6 lts を $\langle \mathcal{P}_{rn}, Act, \longrightarrow_{rn} \rangle$ とする．このとき $\longrightarrow_{rn} \in \mathcal{P}_{rn} \times Act \times \mathcal{P}_{rn}$ は以下の規則を満たす最小の関係である．

$$\text{ACT:}\ \frac{-}{\alpha.P \xrightarrow{\alpha}_{rn} P}$$

$$\text{SUM:}\ \frac{P \xrightarrow{\alpha}_{rn} P'}{P+Q \xrightarrow{\alpha}_{rn} P'+Q}$$

$$\text{COM:}\ \frac{P \xrightarrow{\overline{x}y}_{rn} P',\ Q \xrightarrow{x(z)}_{rn} Q'}{P \mid Q \xrightarrow{\tau}_{rn} P' \mid Q'\{y/z\}}$$

$$\text{PAR:}\ \frac{P \xrightarrow{\alpha}_{rn} P'}{P \mid Q \xrightarrow{\alpha}_{rn} P' \mid Q}$$

$$\text{RES:}\ \frac{P \xrightarrow{\alpha}_{rn} P'}{(\nu x)P \xrightarrow{\alpha}_{rn} (\nu x)P'}\ x \notin \mathrm{n}(\alpha)$$

$$\text{OPEN:}\ \frac{P \xrightarrow{\overline{x}y}_{rn} P'}{(\nu y)P \xrightarrow{\overline{x}(y)}_{rn} P'}\ y \neq x$$

$$\text{CLOSE:}\ \frac{P \xrightarrow{\overline{x}(y)}_{rn} P',\ Q \xrightarrow{x(z)}_{rn} Q'}{P \mid Q \xrightarrow{\tau}_{rn} (\nu y)(P' \mid Q'\{y/z\})}$$

さらに 2 項演算子 $+$ と $\mid$ に関して対称形の規則を加える．また $y \neq x$ のとき $(\nu y)\overline{x}y.P$ を略記して $\overline{x}(w).P$ と書くことがある．

[5] で定義された遷移規則と定義 2.6 の遷移規則との間には次の補題が成り立つ．

命題 2.7 任意の $P \in \mathcal{P}_{rn}$ について

(i) $P \xrightarrow{\overline{x}y} R \iff P \xrightarrow{\overline{x}y}_{rn} R$

(ii) $P \xrightarrow{\overline{x}(y)} R \iff \exists R.\left(P \xrightarrow{\overline{x}(z)}_{rn} S \text{ and } S \equiv R\{z/y\}\right)$

(iii) $P \xrightarrow{x(y)} R \iff \exists R.\left(P \xrightarrow{x(z)}_{rn} S \text{ and } S \equiv R\{z/y\}\right)$

(iv) $P \xrightarrow{\tau} R \iff P \xrightarrow{\tau}_{rn} R$

補題 2.7 の証明は [5] で定義された遷移規則と定義 2.6 の遷移規則について，推論の深さによる帰納法を用いて容易にできる．補題 2.7 より $\mathcal{P}_{rn}$ のプロセスについて，その意味として定義 2.6 の遷移規則を用いても [5] の遷移規則の意図を変えない．

次に $\mathcal{P}$ 上の同値関係 ESGB (Early Strong Ground Bisimilar) を定義する．

定義 2.8 プロセス上の 2 項関係 $\mathcal{R}$ が early strong simulation であるとは，$P\mathcal{R}Q$ のとき任意のアクションに対して

(1) $P \xrightarrow{\alpha} P'$ ($\alpha = \tau$, $\overline{x}z$ or $\overline{x}(y)$ with $y \notin \mathrm{fn}(P,Q)$) ならば，$\exists Q'.(Q \xrightarrow{\alpha} Q'$ かつ $P'\mathcal{R}Q')$

(2) $P \xrightarrow{x(y)} P' (y \notin \mathrm{fn}(P,Q))$ ならば，$\forall w.\exists Q'.\exists z.\left(Q \xrightarrow{x(z)} Q'$ かつ $P'\{w/y\}\mathcal{R}Q'\{w/z\}\right)$

が成立する場合である．

$\mathcal{R}$ と $\mathcal{R}^{-1}$ が early strong simulation であるとき，$\mathcal{R}$ を early strong bisimulation と呼ぶ．また $\sim_E = \bigcup\{\mathcal{R} \mid \mathcal{R}$ は early strong bisimulation $\}$ と定義する．

命題 2.9 $\mathcal{P}$ 上の関係 $\sim_E$ は同値関係である．

2.2 プロセス論理 (Process Logic)

この節ではプロセス論理の構文と意味を定義し，ESGB との関係を述べる．

定義 2.10 論理式 (formula)（あるいはプロセス論理式）を次のように再帰的に定義する．

(1) **true** は論理式である．

(2) A, A_1, A_2 が論理式ならば，
$\neg A$, $A_1 \wedge A_2$ は論理式である．

(3) A が論理式ならば $\langle\alpha\rangle A$ は論理式である．ここで $\alpha \in Act$ である．

ここで様相演算子 $\langle\alpha\rangle$ は可能性様相演算子 (posibility modal operator) と呼ばれる．論理式について自由，束縛，代入および $\equiv_\alpha$, fn(·), bn(·), n(·) をプロセスと同様に定義する．

定義 2.11 プロセス論理 (process logic) $\mathcal{PL}$ は論理式全体からなる集合である．

次にプロセス論理に対して rn 標準形を定義する．まず $\mathcal{P}_{rn}$ と同様にして $\mathcal{PL}$ に対して $\mathcal{PL}_{rn}$ を定義する．

補題 2.12 $A \in \mathcal{PL} \Longrightarrow \exists B \in \mathcal{PL}_{rn}.A \equiv_\alpha B$

定義 2.13 論理式の rn 標準形

論理式を A とする．論理式 B が 2 つの条件 (1) $B \in \mathcal{PL}_{rn}$，(2) $A \equiv_\alpha B$ を満たすとき，B を A の rn 標準形という．

定義 2.14 rn 標準化

論理式 $A_1, A_2, \cdots, A_m$ およびプロセス $P_1, P_2, \cdots, P_n$ を rn 標準化するとは，論理式 $A_1, A_2, \cdots, A_m$ およびプロセス $P_1, P_2, \cdots, P_n$ を，それぞれ論理式 $A'_1, A'_2, \cdots, A'_m$ およびプロセス $P'_1, P'_2, \cdots, P'_n$ に置き換えることである．

ここで論理式 $A'_1, A'_2, \cdots, A'_m$ およびプロセス $P'_1, P'_2, \cdots, P'_n$ は，それぞれ論理式 $A_1, A_2, \cdots, A_m$ およびプロセス $P_1, P_2, \cdots, P_n$ の rn 標準形である．さらに $\bigcap_{1\le i\le m}\left(\mathrm{fn}(A'_i) \cap \mathrm{bn}(A'_i)\right) \cap \bigcap_{1\le j\le n}\left(\mathrm{fn}(P'_j) \cap \mathrm{bn}(P'_j)\right) = \emptyset$ である．

プロセス P が論理式 A を満たす (satisfy) ことを 以下のように定める．

定義 2.15 論理式の充足性

プロセス $\mathcal{P}$ とプロセス論理 $\mathcal{PL}$ の間の関係 $\models \subseteq \mathcal{P} \times \mathcal{PL}$ を次のように定義する．任意のプロセスと論理式に対して

$P \models \mathbf{true}$ for all P

$P \models A_1 \wedge A_2$ if $P \models A_1$ and $P \models A_2$

$P \models \neg A$ if not $P \models A$

$P \models \langle\tau\rangle A$
if $\exists P'.(P \xrightarrow{\tau} P'$ and $P' \models A)$

$P \models \langle\overline{x}y\rangle A$
if $\exists P'.(P \xrightarrow{\overline{x}y} P'$ and $P' \models A)$

$P \models \langle\overline{x}(y)\rangle A$
if $\exists P'.\exists w \notin \mathrm{fn}(A) - \{y\}.$
$(P \xrightarrow{\overline{x}(w)} P'$ and $P' \models A\{w/y\})$

$P \models \langle x(y)\rangle A$
if $\forall z.\exists P'.\exists w.\left(P \xrightarrow{x(w)} P'\right.$ and
$\left.P'\{z/w\} \models A\{z/y\}\right)$

プロセス論理と同値関係 ESGB に対して次の定理が成り立つ [6].

定理 2.16 L ($L \subseteq \mathcal{PL}$) と $P \in \mathcal{P}$ に対して

$$L(P) = \{A \in L \mid P \models A\}$$

とする．このとき任意の $P, Q \in \mathcal{P}$ に対して

$$P \sim_E Q \Longleftrightarrow \mathcal{PL}(P) = \mathcal{PL}(Q)$$

$\mathcal{PL}$ 上の α 変換では次の補題が成立する[6].

補題 2.17

(1) $P \models A$ かつ $A \equiv_\alpha B$ ならば, $P \models B$
(2) $P \models A$ かつ $u \notin \mathrm{fn}(P) \cup \mathrm{fn}(A)$ ならば, $P\{u/v\} \models A\{u/v\}$

補題 2.12 と補題 2.17 によりプロセス論理の構文領域として $\mathcal{PL}_{rn}$ を用いれば十分である. 定義 2.10 を用いて次の論理記号を導入する.

$$\mathbf{false} \stackrel{\mathrm{def}}{=} \neg\mathbf{true}$$
$$A_1 \vee A_2 \stackrel{\mathrm{def}}{=} \neg(\neg A_1 \wedge \neg A_2)$$
$$[\alpha]A \stackrel{\mathrm{def}}{=} \neg\langle\alpha\rangle\neg A$$

$[\alpha]$ は必然性様相演算子 (necessity modal operator) と呼ばれる様相演算子である. 次のように $[x(y)]A$ を定義することもできる.

$$P \models [x(y)]A$$
$$\text{if } \exists z.\forall P'.\forall w.\Big(P \xrightarrow{x(w)} P' \text{ ならば } P'\{z/w\} \models A\{z/y\}\Big)$$

このとき次の補題が成立する [8].

補題 2.18 任意の論理式は, 論理演算子 $\wedge$, $\vee$, $\langle\alpha\rangle$, $[\alpha]$, **true**, **false** のみを用いた論理式に等価変換できる.

本稿では形式的仕様としてプロセス論理のサブセット $\mathcal{PL}'$ を用いる.

定義 2.19 論理演算子 **true**, **false**, $\wedge$, $\langle\alpha\rangle$, $[\alpha]$ からなる $\mathcal{PL}$ の部分集合を $\mathcal{PL}'$ とする.

3 並行プロセスの表現法

3.1 本手法で扱うプロセス

プロセスを組織的に合成するアルゴリズムとして [1, 8, 9] がある. これらの合成アルゴリズムでは並行演算子 ($|$) や制限演算子 (ν) を含むプロセスは扱っていない. 本報告では次のようなプロセスのクラス $\mathcal{NP}$ を対象とする.

定義 3.1 $\mathcal{NP}$, $\mathcal{NP}_{rn}$
$\mathcal{NP}_{rn} \subseteq \mathcal{P}_{rn}$ は以下を用いて再帰的に定義されるプロセスからなる集合である.

- $P \in \mathcal{NP}_{rn}$ if $P \in \mathcal{BP}_{rn}$
- $(\nu y)P \in \mathcal{NP}_{rn}$ if $y \in \mathcal{N}$, $P \in \mathcal{NP}_{rn}$ and $\{y\} \cap \mathrm{bn}(P) = \emptyset$
- $P \mid Q \in \mathcal{NP}_{rn}$ if $P, Q \in \mathcal{NP}_{rn}$ and $\mathrm{fn}(P, Q) \cap \mathrm{bn}(P) \cap \mathrm{bn}(Q) = \emptyset$

ここで $\mathcal{BP}_{rn} \subseteq \mathcal{P}_{rn}$ は並行演算子の除くすべての演算子 (無動作, プレフィックス, 和, 制限) からなるプロセスの集合である. また $\mathcal{NP}$ は $\mathcal{NP} = \{P \mid P \equiv_\alpha Q,\ Q \in \mathcal{NP}_{rn}\}$ である.

$P \in \mathcal{NP}$ について P 中の並行演算子の個数を n とする. このとき $(n+1)$ を P の並行度という.

3.2 プロセスの内部表現

2.1 節においてプロセスの意味をラベル付き遷移システムによって与えた. ラベル付き遷移システムは導出木 [2] によって表現できる. 導出木は各プロセス P を始点とし, その枝が P の直接導出に対応するような有向木 $\langle V_P, E_P\rangle$ である. また本アルゴリズムによって合成されるプロセスは $\mathcal{NP}$ の元である. よって命題 2.7 よりアルゴリズム内部では $\mathcal{NP}$ のプロセスの rn 標準形を考えるだけでよい. 本手法では, アルゴリズムで必要となる情報を付加した導出木によってプロセスを表現する. まず $\mathcal{BP}_{rn}$ のプロセスを和導出木を用いて表現する.

定義 3.2 和導出木
I をインデックス集合とする. $i \in I$ とする. $P \in \mathcal{P}_{rn}$ とする. また $P \in V_P$ が根であるような導出木を $\langle V_P, E_P\rangle$ とする. ここで

- 節点の多重集合 V_P
  $V_P = \{Q \mid P \xrightarrow{s} Q,\ s \in Act^*\}$
- 枝の多重集合 E_P
  $E_P = \{\alpha \mid Q \xrightarrow{\alpha},\ Q \in V_P\}$

今 f_i を $f_i(P) = c_{i0}$ であるような, V_P からラベルの集合 $\{c_{ij} \mid j \in J_i\}$ への一対一写像とする. ここで J_i $(0 \in J_i)$ は任意のインデックス集合である. このとき P の和導出木 S_i は和導出木ノードの集合 $\{\langle c_{ij}, N_{ij}, T_{ij}\rangle \mid j \in J_i\}$ である. ここで和導出木ノード $\langle c_{ij}, N_{ij}, T_{ij}\rangle$ は

- c_{ij} はノードラベル
- N_{ij} $(\subseteq \mathcal{N})$ は $\langle V_P, E_P\rangle$ の根から $f_i(Q) = c_{ij}$ となるような Q までのパスに現われた名前の集合

- $T_{ij} = \{(\alpha, c_{ij'}) \mid f_i^{-1}(c_{ij}) = R,$
$R \xrightarrow{\alpha} Q,\ f_i(Q) = c_{ij'}\}$

直観的には N_{ij} はそのノードがリンクを張るとき．使ってよい名前の集合である．

次に $\mathcal{NP}_{rn}$ を表現する方法を述べる．ここで $P \in \mathcal{NP}_{rn}$ について次の補題が成り立つ．証明には $\mathcal{P}_{rn}$ の定義と ESGB の性質 [5] を用いる．

補題 3.3 任意の $P \in \mathcal{NP}_{rn}$ について

$$P \sim_E (\nu x_1)\cdots(\nu x_l)(P_1 \mid \cdots \mid P_i \mid \cdots \mid P_n)$$

$$\begin{cases} \exists P_i \in \mathcal{BP}_{rn}, \\ \bigcap_{1\le i\le n}(\mathrm{fn}(P_i) \cap \mathrm{bn}(P_i)) = \emptyset, \\ l \ge 0,\ 1 \le i \le n \end{cases}$$

従って上式の右辺を表現すればよい．本手法ではこのように表現されたプロセス $P \in \mathcal{NP}_{rn}$ を次のような 3 つ組を用いて表わす．

定義 3.4 3 つ組 $\langle S, S_C, F\rangle$

$P \sim_E (\nu x_1)\cdots(\nu x_l)(P_1 \mid \cdots \mid P_i \mid \cdots \mid P_n)$

$$\begin{cases} \exists P_i \in \mathcal{BP}_{rn}, \\ \bigcap_{1\le i\le n}(\mathrm{fn}(P_i) \cap \mathrm{bn}(P_i)) = \emptyset, \\ l \ge 0,\ 1 \le i \le n \end{cases}$$

について，3 つ組 $\langle S, S_C, F\rangle$ を次のように定義する．ここで $\{1,2,\cdots,n\} = I$ はインデックス集合で，その要素数は P の並行度である．

- $S = \bigcup_{1\le i\le n} S_i$，$S_i$ は P_i の和導出木
- $S_c \subseteq \Big\{\langle C, A_C\rangle \;\Big|\; C \in \Big\{\{c_{1j_1},\cdots,c_{ij_i},\cdots,c_{nj_n}\} \;\Big|\; j_1 \in J_1,\cdots,j_i \in J_i,\cdots,j_n \in J_n\Big\}$，$A_C$ は $[\alpha]B$ の形をした論理式の集合 $\Big\}$
- $F = \mathrm{fn}(P)$

ラベルの集合 $C = \{c_{1j_1},\cdots,c_{ij_i}\cdots,c_{nj_n}\}$ は直観的には P を導出木で表現したときの各ノードを表わす．名前の集合 $N_{ij} \cup F$ の元のみが，P_i の和導出木ノード $\langle c_{ij}, N_{ij}, T_{ij}\rangle$ においてリンクとして使うことができる．

また S_c の要素を導出木ノードと呼ぶ．

$P \in \mathcal{NP}_{rn}$ から得られる 3 つ組 $\langle S, S_C, F\rangle$ と P の導出木の間には次の補題が成り立つ．

補題 3.5 $P \in \mathcal{NP}_{rn}$ から得られる 3 つ組を $\langle S, S_C, F\rangle$ とする．このとき $\langle S, S_C, F\rangle$ は P が根であるような導出木に変換できる．

4 合成アルゴリズム

4.1 合成アルゴリズムの概要

以下ではアルゴリズムで扱われるプロセスと論理式は常に rn 標準化されているものとする．よってアルゴリズムでは命題 2.7 の結果より遷移規則として定義 2.6 を用いる．またアルゴリズムに対する入出力を以下の通りとする．

入力　$A_1, A_2, \cdots, A_o$

出力　$P_1, P_2, \cdots, P_o$

ここで入力 $A_1, A_2, \cdots, A_o \in \mathcal{PL}'$ は満足すべき論理式の列である．また出力 $P_i (1 \le i \le o) \in \mathcal{NP}_{rn}$ は A_1 から A_i までの論理式を満たすプロセスである．各 P_i はアルゴリズム内部では 3 つ組 $\langle S, S_C, F\rangle$ で表現されている．

アルゴリズムの本体は，導出木ノード D_C と論理式 A_i から，導出木ノード D'_C を構成する手続きである．D_C を根とする導出木が満たす論理式の集合を $L = \{A_1,\cdots,A_{i-1}\}$ とする．この手続きにより構成された D'_C を根とする導出木は，論理式の集合 $L \cup \{A_i\}$ を満たす．

アルゴリズムに L の元が順次入力されたとする．このときアルゴリズム内にある合成中のプロセスを表わす導出木の根を D_C とする．また導出木ノード D_C を根とする導出木は L を満たすとする．アルゴリズムに新たに A_i が入力されたとき，D_C と A_i を先の手続きに適用して構成された導出木ノードを D'_C とする．この導出木ノード D'_C を根とする導出木は $L \cup \{A_i\}$ を満たす．このようにして本アルゴリズムは今まで入力された論理式を満たすプロセスから，新たに入力された論理式も満たすプロセスを合成する．

以下に手続きの概略を示す．手続きは渡された論理式の形で処理が異なる．また動作 α の種類によっても処理の詳細が異なる．$[\alpha]A$,

$\langle\alpha\rangle A$ の形の論理式についてのみ，その処理の概略を示す．親導出木ノードから論理式を渡された導出木ノードを $D_C = \langle C, A_C\rangle$ とする．

導出木ノード D_C に論理式 $[\alpha]A$ が渡されとき，導出木ノードの条件 A_C に $[\alpha]A$ を加える．そして D_C が α によって遷移した先を表わす子導出木ノードを1つ以上持てば，それらすべてに A を渡す．それがすべてが成功すれば D_C と $[\alpha]A$ に対する手続きは成功する．

導出木ノード D_C に論理式 $\langle\alpha\rangle B$ が渡されたとき，導出木ノードを構成する処理と，構成された導出木ノードを検査する処理を行なう．

導出木ノードの構成

(a) D_C が α による遷移を1つ以上持つ時
  (1) α で遷移した先を表わす子導出木ノードで，まだ (2) を実行していないものを D_{C_1} とする．
  (2) D_{C_1} に A を渡す．すなわち D_{C_1} と A に対して手続きを適用する．
  (3) (2) が成功すれば導出木ノードの検査の処理に入る．
  (2) が失敗すれば (1) に戻る．

(b) D_C が α による遷移を持たない時
  D_C に α により遷移する子導出木ノードを作り (a) に戻る．子ノードは並行・和の順に各演算子を用いて作る．

(c) (a), (b) がすべて失敗した時
  D_C を根とする導出木は $\langle\alpha\rangle A$ を満たさない．よって失敗を返す．

導出木ノードの検査　導出木を構成した結果，$D_C\langle C, A_C\rangle$ を根とする導出木が今まで満足していた条件 A_C を満たさなくなっていれば，D_C と A_C に対して手続きを適用する．

上記のアルゴリズムを入出力述語に対してもバックトラックする PROLOG 風の言語を用いて記述した [7].

4.2 本手法の合成例

動的なリンクの張り替えを含む例として，ホテルのフロントと客の状況を考える．最初は，フロントは客の名前を知らず，同様に客はどの部屋に泊まれるか知らない．よって客が部屋に泊まるには，フロントと客の間で次のような情報のやりとりが必要である．

(1) 客はフロントに自分の名前を告げ，
(2) 名前を告げられたフロントは，
  空いている部屋を探し，
(3) 客の名前を呼び部屋番号を伝え，
(4) 部屋番号を聞いた客は，
  その部屋に入る．

すなわちフロントは告げられた“客の名前”という情報を使って，“部屋番号”という情報を客に伝える．

この (1) ～ (4) を論理式で表現する．

$$A_1 = \langle\overline{front}(myname)\rangle \langle front(name)\rangle\langle ack(room)\rangle \langle\overline{name}\ room\rangle \langle myname(room_n)\rangle\langle\overline{room_n}(in)\rangle\mathbf{true}$$

また (2) でフロントは空いている部屋を何らかの方法を使って探している．この方法を論理式で表現すると次のようになる．

$$A_2 = \langle\overline{ack}(room_1)\rangle\langle room_1(in)\rangle\mathbf{true}$$

本アルゴリズムに論理式 A_1, A_2 を順次入力として与えてみる．まず論理式 A_1 を入力したとき，出力されるプロセス P_1 は

$$P_1 \equiv (\nu\ myname)(\nu\ in)\Big(\overline{front}\ myname.\ myname(room_n).\overline{room_n}\ in.0 \mid front(name).\ ack(room).\overline{name}\ room.0\Big)$$

である．次の入力として論理式 A_2 をアルゴリズムに与えたとき，出力されるプロセス P_2 は

$$P_2 \equiv (\nu\ myname)(\nu\ in)(\nu\ room_n)\Big(\overline{front}\ myname.\ myname(room_n).\overline{room_n}\ in.0 \mid front(name).\ ack(room).\overline{name}\ room.0 \mid \overline{ack}\ room_1.room_1(in_1).0\Big)$$

である．P_2 は論理式 A_1, A_2 を同時に満たすプロセスである．また P_2 は3つのプロセスが

並行に動作するシステムを表現している．3つのプロセスはそれぞれ直観的に客，フロント，フロントが空いている部屋を探す方法 を表わすプロセスである．客を表わすプロセスはフロントを表わすプロセスに対して，名前 *front* を通じて名前 *myname* を渡す．そして渡した名前 *myname* を通じて名前 $room_1$ を受け取っている．このように3つのプロセスは動的なリンクを張り替えを行なっている．

5 むすび

本稿では，プログラム合成に対するアプローチのひとつとして，プロセス論理式から π 計算の並行プロセスを合成する手法を提案した．すなわちプロセス合成を，今まで入力された論理式を満たすプロセスを表わす木から，新たに入力された論理式を満たすように木を再構成する手続きであると捉え，この手続きの形式化の概要を示した．例としてプロセス論理でホテルと客の状況を記述した仕様を本手法に与えた．本手法はその仕様から，複数のプロセスが並行に動作し，そのプロセス間で動的なリンクの張り替えを行なうプロセスを合成した．

本手法では，入力としてプロセスの満たすべきプロセス論理式の集合を与えたとき，その集合の元を入力する順序に依存して異なるプロセスが合成される．しかし定理 2.16 より本手法では，十分に詳細な仕様が与えられれば，入力の順序に関わらず等価なプロセスが得られることが期待される．

また本手法では並行に動作しているプロセスを合成することができるため，通信を含むプロセスが記述できる．プロセス間の通信を表わす τ による遷移を外部から観測できないとする同値関係 [5] があるが，この同値関係を本手法に適用できるかどうかは興味深い．

π 計算は名前渡し以外に複製 (replication) という特徴を持つ．π 計算はこの特徴により再帰を含むプロセスを表現できる．よって本手法を複製も扱えるよう拡張することで再帰を含むプロセスの合成が期待される．

謝辞 本研究に関して御指導下さった山崎進教授に深く感謝致します．本研究に関する議論に参加して下さった西崎真也助手をはじめ，知能情報処理工学研究室諸氏に対して感謝します．

参考文献

[1] 木村，富樫，白鳥：Synthesis Algorithm for Recursive Processes by μ-calculus, COMP 93, 信学技報, 1994.

[2] Milner, R.: *Communication and Concurrency*, Prentice Hall, 1989.

[3] Milner, R.: The polyadic π-calculus: a tutorial, Technical Report ECS-LFCS-91-180, University of Edinburgh, 1991.

[4] Milner, R.: Functions as Processes, *Journal of Mathematical Structures in Computer Science*, Vol. 2 (1992), 119–141.

[5] Milner, R., Parrow, J. and Walker, D.: A Calculus of Mobile Processes, Reports ECS-LFCS-89-85 and -86, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1989.

[6] Milner, R., Parrow, J. and Walker, D.: Modal Logics for Mobile Processes, Reports ECS-LFCS-91-136, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1991.

[7] 奥畑：名前渡しを含む並行プロセスの合成に関する研究, 修士論文, 岡山大学, 1995.

[8] 富樫, 木村, 野口：並行プロセス自動生成の可能性について, 第29回東北大学電気通信研究所シンポジウム論文集, 1993.

[9] 臼井，吉田，木村，富樫，白鳥：プロセス合成のための支援環境に関する研究, COMP 93, 信学技報, 1994.