プロセス計算記述言語とその支援環境

吉田 仙[†,☆] 富樫 敦[†] 白鳥則郎[†]

並行に動作するプロセスの振舞いを代数的な形式体系として扱うプロセス計算が注目され，これまでに多数開発されてきた．また，それらプロセス計算において，その構文規則にしたがって表現されたプロセスの操作的意味を調べることを目的とする処理系が，いくつかのプロセス計算に対し作成されている．しかしながら，個々のプロセス計算に対しそれぞれ処理系を作成するのは非効率的であり，プロセス計算の開発・利用を統合的に支援し，汎用の処理機能を提供できるようなシステムを構築することが望まれる．そこで本論文では，まずプロセス計算を形式的に扱うためのモデルとして多ソートSOSシステムを定義する．また多ソートSOSシステムをモデルに持つプロセス計算記述言語LCCを提唱し，さらにこの言語のインタプリタとしての機能を持つプロセス計算支援システム **ProCSuS** の実装例を紹介する．

A Process Calculus Description Language and Its Support Environment

SEN YOSHIDA,[†,☆] ATSUSHI TOGASHI[†] and NORIO SHIRATORI[†]

In the literature, many process calculi have been developed, and their interpreters have been made. In general, making such interpreters may take much effort and that has disturbed developing process calculi. For that reason, we have constructed an integrated support environment for developping or utilizing process calculi. In this paper, we propose a Language of Concurrent process Calculi, LCC which can describe syntax and transition rules of process calculi. We also describe the outline of **ProCSuS**, a Process Calculus Support System which interprets LCC and investigates operational semantics of the calculus.

1. 序 論

情報通信システムの仕様を厳密に記述し，計算機による検証やシステムの自動合成を可能にするための手段として，形式記述技法が用いられる．その中で，近年のシステムの分散化・並列化に伴い，並行に動作するプロセスの振舞いを代数的な形式体系として扱うプロセス計算[21]が注目され，これまでに多数開発されてきた．CCS[11),12)]，CSP[8)]，LOTOS[9)]，π-calculus[13)]などはプロセス計算の代表的な例である．

また，それらプロセス計算において，その構文規則にしたがって表現されたプロセスの操作的意味を調べることを目的とする処理系が，いくつかのプロセス計算に対し作成されている[6),10),20)]．しかしながら，そのような処理系を作成するには一般に多大な労力を必要とし，個々のプロセス計算に対しそれぞれ処理系を作成するのは非効率的である．そこで，プロセス計算の開発・利用を統合的に支援し，汎用の処理機能を提供できるような環境を構築することが望まれる．

以上の理由から本論文では，プロセス計算を抽象的に扱うための形式技法を示し，それに基づいてプロセス計算記述のためのプログラミング言語を提唱する．またそのインタプリタを実装し，既存のプロセス計算に適用することで本手法の有効性を実証する．

本論文の構成は，まず第2章でプロセス計算を形式的に扱うためのモデルとして多ソートSOS形式を定義する．また第3章では多ソートSOS形式に基づくプロセス計算記述言語LCCを提唱し，第4章でこの言語のインタプリタを基本構成要素とするプロセス計算支援システム **ProCSuS** を紹介する．さらに第5章でLCCや **ProCSuS** を既存の代表的なプロセス計算であるCCSに適用した例を示し，最後に第6章でまとめと今後の課題について述べる．

† 東北大学電気通信研究所/情報科学研究科
Reserch Institute of Electorical Communication/Graduate School of Information Science, Tohoku University
☆ 現在，日本電信電話株式会社コミュニケーション科学研究所
Presently with Communication Science Laboratories, Nippon Telegraph and Telephone Corporation

2. 多ソート SOS 形式

プロセス計算は並行システムを形式的に記述するのに適した計算体系であり，これまで多くの研究者が多数のプロセス計算を提唱しその性質を解明してきた．この章では，プロセス計算を抽象的なレベルでとらえ統一的に記述するための土台として，多ソート SOS 形式を定義する．

既存のプロセス計算のほとんどは，その操作的意味が SOS[14] 指向で定義されている．SOS 指向のプロセス計算を抽象的に扱うことができる形式記述技法として，GSOS[1),3)] や TSS[7)] が知られている．多ソート SOS 形式は，これらの形式を多ソートに拡張し，またアクションにも構造を持たせた形式である．多ソート SOS 形式ではプロセス計算の各プロセスおよびアクションが多ソート代数の項で表され，また操作的意味が SOS 指向で与えられる．

2.1 多ソート SOS 形式

はじめに，多ソート SOS 形式の構文を規定する遷移式言語を定義する．遷移式言語は，抽象データ型の構文である等式言語[17)] に似た構文を持つ．等式言語では等式の左辺と右辺が多ソートの項で表されるが，遷移式言語では遷移前後のプロセスや遷移に伴うアクションが多ソートの項として表される．

定義 2.1 （シグニチャ）
シグニチャは対 $\langle S, \Sigma \rangle$ である．ここで S はソートの有限集合，Σ は演算記号の有限集合の族 $\langle \Sigma_{w,s} \rangle_{w \in S^*, s \in S}$ であり，S^* は S の元からなるすべての有限系列の集合である．特に，$\Sigma_{\varepsilon,s}$ の元をソート s の定数記号という． □

定義 2.2 （遷移式言語）
遷移式言語は 4 項組 $\langle S, \Sigma, X, \rightarrow \rangle$ である．ここで S と Σ は $\langle S, \Sigma \rangle$ がシグニチャとなる集合と集合族である．X は $X_s \cap X_{s'} = \emptyset$ $(s \neq s')$ であるような記号の加算無限集合 X_s の族 $\langle X_s \rangle_{s \in S}$ であり，X_s の元をソート s の変数という．$\rightarrow$ は遷移を表す述語記号である． □

定義 2.3 （項）
$\langle S, \Sigma, X, \rightarrow \rangle$ を遷移式言語とする．このとき各ソート s に対して，$T[\Sigma(X)]_s$ を次の条件を満たす最小の集合とする．

- $X_s \subseteq T[\Sigma(X)]_s$
- $f \in \Sigma_{s_1 \cdots s_n, s}$ $(n \geq 0)$ かつ $\xi_i \in T[\Sigma(X)]_{s_i}$ $(i = 1, \cdots, n)$ ならば $f(\xi_1, \cdots, \xi_n) \in T[\Sigma(X)]_s$

f が定数記号であるとき，$f()$ を f と略記する．$T[\Sigma(X)]_s$ の元をソート s の項という．集合族 $\langle T[\Sigma(X)]_s \rangle_{s \in S}$ を $T[\Sigma(X)]$ と書く．また，$T[\Sigma]_s$ は変数を含まない項の集合であり，その元をソート s の閉じた項という． □

遷移式言語における式には正の遷移式と負の遷移式の 2 種類の遷移式がある．正の遷移式はある状態から遷移を起こすことができることを表しており，遷移の前後の状態を表す項と，遷移時に行われるアクションを表す項からなる．遷移は同一のソートの台の間でのみ起こるものとする．また，負の遷移式はある状態からあるアクションを伴う遷移を起こすことができないことを表しており，状態を表す項とアクションを表す項からなる．

定義 2.4 （遷移式）
遷移式言語 $\langle S, \Sigma, X, \rightarrow \rangle$ における正の遷移式は記号列 $\xi \xrightarrow{\alpha} \eta$ である．ここで，あるソート $s_p, s_a \in S$ に対し，$\xi, \eta \in T[\Sigma(X)]_{s_p}$ $\alpha \in T[\Sigma(X)]_{s_a}$ である．ξ を遷移式 $\xi \xrightarrow{\alpha} \eta$ のソース，η をターゲットという．また，負の遷移式は記号列 $\xi \overset{\alpha}{\not\rightarrow}$ である． □

既存のプロセス計算の多くは各プロセスの動作を SOS 指向の遷移規則の集合によって規定している．ここでは，遷移規則を遷移式から構成される多ソート SOS 規則として与える．

定義 2.5 （多ソート SOS 規則）
遷移式言語 $\mathcal{L} = \langle S, \Sigma, X, \rightarrow \rangle$ における多ソート SOS 規則は次のような形の推論規則である．

$$\frac{\{\varphi_i \mid i \in I\} \cup \{\psi_j \mid j \in J\}}{\varphi}$$

ここで $I, J \geq 0$ はインデックスの集合であり，各 φ_i $(i \in I)$ と φ は $\mathcal{L}$ の正の遷移式，各 ψ_j $(j \in J)$ は負の遷移式である．$\{\varphi_i \mid i \in I\} \cup \{\psi_j \mid j \in J\}$ を多ソート SOS 規則の仮定，φ を結論という．$J = \emptyset$ である規則を正の多ソート SOS 規則という． □

定義 2.6 （多ソート SOS システム）
多ソート SOS システムは遷移式言語 $\mathcal{L} = \langle S, \Sigma, X, \rightarrow \rangle$ と $\mathcal{L}$ における多ソート SOS 規則の有限集合 Γ との対 $\langle \mathcal{L}, \Gamma \rangle$ である．特に，Γ が正の多ソート SOS 規則の集合であるとき，$\langle \mathcal{L}, \Gamma \rangle$ を正の多ソート SOS システムという．$\langle \mathcal{L}, \Gamma \rangle$ を $\langle S, \Sigma, X, \Gamma \rangle$ と略記する． □

2.2 操作的意味モデル

次に多ソート SOS システムに操作的意味を与える方法を示す．一般に，プロセス計算におけるプロセスの操作的意味は，そのプロセスを初期状態に持つラベル付き遷移システムで与えられる．

定義 2.7 （ラベル付き遷移システム）
ラベル付き遷移システムは 4 項組 $\langle \mathcal{P}, \mathcal{A}, \rightsquigarrow, p_0 \rangle$ である．ここで $\mathcal{P}$ は状態の空でない集合，$\mathcal{A}$ はアクショ

ンの集合である．また，$\rightsquigarrow = \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ は遷移関係であり，$(p,a,q) \in \rightsquigarrow$ を $p \overset{a}{\rightsquigarrow} q$ と書く．$p_0 \in \mathcal{P}$ は初期状態である． □

多ソート SOS システムにおけるプロセスの操作的意味は，プロセスグラフと呼ばれるラベル付き遷移システムによって与えられる．

定義 2.8 （代入）

$\langle S, \Sigma, X, \rightarrow \rangle$ を遷移式言語とする．このとき，閉じた代入 σ は変数から閉じた項への写像族 $\langle \sigma_s : X_s \rightarrow T[\Sigma]_s \rangle_{s \in S}$ である．

σ を項から閉じた項への写像族 $\langle \sigma_s : T[\Sigma(X)]_s \rightarrow T[\Sigma]_s \rangle_{s \in S}$ に拡張する．すなわち，任意の $\xi \in T[\Sigma(X)]_s$ に対して，

- $\xi = x \in X_s$ ならば $\sigma_s(\xi) = \sigma_s(x)$
- $\xi = f(\xi_1, \cdots, \xi_n)$, $f \in \Sigma_{s_1 \cdots s_n, s}$ ならば $\sigma_s(\xi) = f(\sigma_{s_1}(\xi_1), \cdots, \sigma_{s_n}(\xi_n))$

さらに，σ を遷移式にも拡張する． □

定義 2.9

$\langle S, \Sigma, X, \rightarrow \rangle$ を遷移式言語，$\rightsquigarrow$ を $T[\Sigma]$ 上の遷移関係☆，σ を閉じた代入とする．遷移式 φ に対し，述語 $\rightsquigarrow, \sigma \models \varphi$ を次のように定義する．

$$\rightsquigarrow, \sigma \models \xi \overset{\alpha}{\rightarrow} \eta \overset{\text{def}}{=} \sigma(\xi) \overset{\sigma(\alpha)}{\rightsquigarrow} \sigma(\eta)$$

$$\rightsquigarrow, \sigma \models \xi \overset{\alpha}{\not\rightarrow} \overset{\text{def}}{=} \nexists q : \sigma(\xi) \overset{\sigma(\alpha)}{\rightsquigarrow} q$$

遷移式の集合 Φ に対し次のように定義する．

$$\rightsquigarrow, \sigma \models \Phi \overset{\text{def}}{=} \forall \varphi \in \Phi : \rightsquigarrow, \sigma \models \varphi$$

さらに多ソート SOS 規則 Φ/φ に対し次のように定義する．

$$\rightsquigarrow, \sigma \models \frac{\Phi}{\varphi} \overset{\text{def}}{=} (\rightsquigarrow, \sigma \models \Phi \Rightarrow \rightsquigarrow, \sigma \models \varphi)$$

□

定義 2.10 （モデル）

$\mathcal{C} = \langle S, \Sigma, X, \Gamma \rangle$ を多ソート SOS システムとする．このとき，以下を満たす $T[\Sigma(X)]$ 上の遷移関係 $\rightsquigarrow$ を $\mathcal{C}$ のモデルという．

- Γ のすべての規則 γ と $T[\Sigma(X)]$ 上のすべての閉じた代入 σ に対し $\rightsquigarrow, \sigma \models \gamma$ となる．
- $\rightsquigarrow$ のすべての遷移 $p \overset{a}{\rightsquigarrow} q$ に対し，Γ の規則 Φ/φ と閉じた代入 σ が存在し $\rightsquigarrow, \sigma \models \Phi$ かつ $\sigma(\varphi) = p \overset{a}{\rightarrow} q$ となる．

$\mathcal{C}$ のモデルが唯一存在するとき，それを $\rightsquigarrow_{\mathcal{C}}$ と書く． □

☆ つまり，各ソート $s \in S$ に対する $T[\Sigma]$ 上の遷移関係 $\rightsquigarrow_s$ の族 $\langle \rightsquigarrow_s \rangle_{s \in S}$

定義 2.11 （プロセスグラフ）

$\mathcal{C} = \langle S, \Sigma, X, \Gamma \rangle$ を多ソート SOS システムとする．$\mathcal{C}$ における閉じた項 $p_0 \in T[\Sigma]_{s_p}$ のプロセスグラフはラベル付き遷移システム $\langle \mathcal{P}, \mathcal{A}, \rightsquigarrow, p_0 \rangle$ である．ここで，$\mathcal{P}$, $\mathcal{A}$, $\rightsquigarrow$ はそれぞれ $T[\Sigma]_{s_p}$, $\cup_{s \in S} T[\Sigma]_s$, $\rightsquigarrow_{\mathcal{C}}$ を p_0 から到達可能なものに制限したものである． □

3. プロセス計算記述言語

この章では，多ソート SOS 形式を実現するプログラミング言語として，プロセス計算記述言語 LCC (Language for Concurrent process Calculi) を提唱する．

LCC は，前章で定義した多ソート SOS 形式を実現し，様々なプロセス計算におけるプロセスの操作的意味を汎用処理系が導出することを可能にするプログラミング言語である．LCC の構文は基本的に多ソート SOS 形式と一致するが，記述の容易性を増すためにサブソートを扱えるようにするなどの便宜がはかられている．

簡単な例として，アクションプレフィックスと選択に関する演算記号だけを持つ CCS のサブセットを LCC で記述すると，以下のようになる．

```
calculus SUMMATION is
sorts action process .
subsorts internal < action .
ops a b c : -> action .
op tau : -> internal .
op 0 : -> process .
op * : action process -> process .
op + : process process -> process .
var A : action .
vars P P1 P2 Q Q1 Q2 : process .
rule => *(A,P) - A -> P .
rule P1 - A -> P2 => +(P1,Q) - A -> P2 .
rule Q1 - A -> Q2 => +(P,Q1) - A -> Q2 .
endcalc
```

最初の行は **header** であり，このプロセス計算が SUMMATION と名付けられていることを示している．

2 行目から 8 行目までが **signature** である．SUMMATION では action と process という 2 種類のソートを扱う．action は内部アクションのためのサブソート internal を持つ．

ソート action の台には，a, b, c という定数項と，サブソート internal の台で内部アクションを表す tau という定数項が属する．0 は無動作プロセスを表すソート process の定数項である．7 行目はアクショ

ンプレフィックスを表す演算子を定義している．*は action と process の二つの引数をとるオペレータであり，構成される項は process の台となる．次の行は選択を表すオペレータの定義である．+は process の台から二つの引数をとり process の項を構成する．

9行目から13行目までは **body** である．ソート action の変数は A であり，process の変数は P，P1，P2などである．第11行はアクションプレフィックスに関する遷移規則の記述であり，12行めと13行めには選択に関する規則が記述されている．

最後の行が **footer** で，以上で SUMMATION の記述が完了する．

4. プロセス計算支援システム

本章では，第3章で提唱したLCCのインタプリタを核にして設計されたプロセス計算支援システム **ProCSuS** について，その概要を説明する．

4.1 ProCSuS

ProCSuS（PROcess Calculus SUport System）は，プロセス計算記述言語LCCを解釈し，プロセス計算の構文規則に従って表現されたプロセスの操作的意味を解析するシステムである．**ProCSuS** は現在，プロセスグラフの導出とその描画，および二つのプロセスグラフ間の等価性判定の機能を持っている．これらの機能を用いることによって，ユーザはこのシステムをプロセス計算の開発および利用支援系として活用することができる．

ProCSuS はSICStus Prolog☆を用いて実装されており，X Window System上で動作する．**ProCSuS** のコンパイラは，LCCで記述された遷移規則を対応するPrologの節に変換する．以下にコンパイラによる変換の例を示す．

```
vars P P1 Q : process .
var A : action .
rule P - A -> P1 Q - A -/->
        => f(P,Q) - A -> g(P1) .
```

上の遷移規則は，次のようなPrologの節に変換される．

```
trans(f(P,Q), A, g(P1)) :-
        sort_element(P, process),
        sort_element(Q, process),
        trans(P, A, P1),
        \+ trans(Q, A, _),
        sort_element(A, action),
        sort_element(P1, process).
```

ここで，trans という述語は遷移関係があることを表す述語であり，trans(P, A, Q) で P はアクション A を起こし Q に遷移することが可能であるという意味にとらえることができる．また，sort_element(P, process) は項 P がソート process の台であるという意味である．\+ は否定を表す SICStus Prolog の組み込み述語である．このような節の集合からなる Prolog のプログラムを用いてゴール節 trans(p, A, Q) を満たす A，Q を探すことによって，p から可能な遷移を導出することができる．

4.2 プロセスグラフ導出

ProCSuS にはLCCで記述されたプロセス計算上でのあるプロセスのプロセスグラフ（定義2.11）を導出するツールが用意されている．このツールは，LCCでの記述によって与えられたプロセス計算に対し，一つのプロセスが入力されるとそのプロセスのプロセスグラフを次のようなアルゴリズムで導出する．

procedure processgraph(p_0);
function rel(p, P);
  begin
    $P := P \cup \{p\}$;
    ゴール節 trans(p, A, Q) を満たすすべての (A, Q) の組の集合をバックトラックを通じて求め，AQ とする;
    repeat
      $(a, q) \in AQ$ を選ぶ;
      $AQ := AQ - (a, q)$;
      $rel := rel \cup (p, a, q)$; $P := P \cup q$;
      if $q \notin P$ **then**
        $rel := rel \cup rel(q, P)$;
    until $AQ = \emptyset$;
  end;
begin
rel(p_0, $\emptyset$) を求め，ファイルに出力する
end.

導出されたプロセスグラフは，ラベル付き遷移システム表示アプリケーション girl[18]を用いてラベル付き有向グラフの形でX Window System上のウィンドウに**図1**のように表示することができる．また，次節で述べるように，二つのプロセスグラフの間で等価性を判定することもできる．

4.3 等価性判定

プロセス計算において各プロセス間の等価性を判断する基準として，プロセスの操作的意味モデルとして

☆ SICStus Prolog は Swedish Institute of Computer Science の登録商標である．

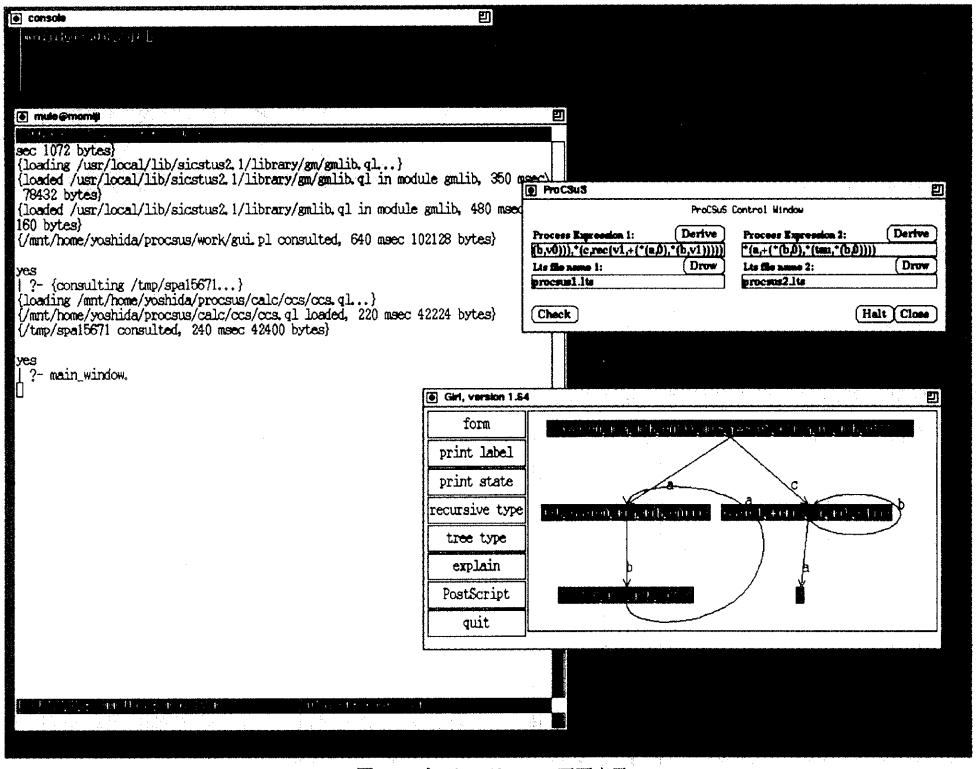


図1 プロセスグラフの画面表示
Fig. 1 A screen image of a process graph.

のプロセスグラフの間の観測等価性[4]を用いるのが一般的である．**ProCSuS** のツールは，二つのラベル付き遷移システムの間の観測等価性を自動的に判定することができる．

観測等価性は，内部アクションを考慮するか否かの違いにより，弱観測等価と強観測等価という二つの等価性に区別される．弱観測等価は，外部からは観測不能な内部アクションを考慮した等価性である．LCC では組み込みソート **internal** の項であるアクションをそのような内部アクションであると解釈する．

ProCSuS は二つのラベル付き遷移システムが与えられると，はじめにそれらが強観測等価であるかどうかを判定し，強観測等価でなかった場合のみ次に弱観測等価であるかどうかを判定する．以下は，3 章で示したプロセス計算 SUMMATION における，$a.(b.\boldsymbol{NIL} + \tau.c.\boldsymbol{NIL})$ と $a.c.\boldsymbol{NIL} + a.(b.\boldsymbol{NIL} + \tau.c.\boldsymbol{NIL})$ の間の等価性を判定した結果である．

```
| ?- write_equiv('*(a,*(b,0))',
  '+(*(a,+(*(b,0),*(b,0))),*(a,*(b,0)))').
These are STRONGLY observation equivalent.

yes
| ?- write_equiv(
  '*(a,+(*(b,0),*(tau,*(c,0))))',
  '+(*(a,*(c,0)),
     *(a,+(*(b,0),*(tau,*(c,0)))))').
These are NOT STRONGLY
  observation equivalent.
These are WEAKLY observation equivalent.

yes
| ?- write_equiv('*(a,+(*(b,0),*(c,0)))',
  '+(*(a,*(b,0)),*(a,*(c,0)))').
These are NOT STRONGLY
  observation equivalent.
These are NOT WEAKLY
  observation equivalent.

yes
| ?-
```

5. CCS への適用例

この章では，既存の代表的なプロセス計算である CCS について，LCC による記述例と **ProCSuS** を

用いて遷移関係を調べた例を紹介する．

5.1 合　　成

CCS の合成に関する遷移規則には，同期が起こる場合の規則

$$\frac{P_1 \xrightarrow{A} P_2 \qquad Q_1 \xrightarrow{\overline{A}} Q_2}{P_1 \mid Q_1 \xrightarrow{\tau} P_2 \mid Q_2}$$

と起こらない場合の規則

$$\frac{P_1 \xrightarrow{A} P_2}{P_1 \mid Q \xrightarrow{A} P_2 \mid Q} \qquad \frac{Q_1 \xrightarrow{A} Q_2}{P \mid Q_1 \xrightarrow{A} P \mid Q_2}$$

がある．同期が起こるかどうかの判定は，**complemental** という補助的なオペレータを用いて実現している．**complemental(Al,Ar)** は，**Al** と **Ar** が相補的な関係にあるとき **tt** に書き換えられる．ここで **tt** は LCC の組み込みソート **bool** の組み込み定数記号であり，真偽値 **true** を表す．

ところで，多ソート SOS システムは項書換えシステムの機能を包含すると考えることができる[7]．すなわち，多ソート SOS 規則の集合において，アクションを項の書換えを表す一つの項だけに限定し，また構造則を明示的に書くことによって，多ソート SOS システムは項書換えシステムのように動作する．そこで，遷移規則における補助的な規則として項書換えシステムを用いることが考えられる．この目的のために LCC は書換えを表す組み込み定数記号 **eq** を用意し，これを用いた規則

```
complemental(Al,Ar) - eq -> tt
```

を，

```
complemental(Al,Ar) --> tt
```

と書くことができるようになっている．これを用いて合成に関する規則は次のように記述できる．

```
rule P1 - Al -> P2 Q1 - Ar -> Q2
     complemental(Al,Ar) --> tt =>
        &(P1,Q1) - tau -> &(P2,Q2) .
rule P1 - A -> P2 =>
  &(P1,Q) - A -> &(P2,Q) .
rule Q1 - A -> Q2 =>
  &(P,Q1) - A -> &(P,Q2) .
```

例として $a.\boldsymbol{NIL} \mid \overline{a}.\boldsymbol{NIL}$ のプロセスグラフを **ProCSuS** を用いて導出し描画すると，**図 2** のようになる．このプロセスは，$a.\boldsymbol{NIL}$ が先に動作する場合，$\overline{a}.\boldsymbol{NIL}$ が先に動作する場合，両者が同期をとって動作する場合の 3 通りの非決定的選択になる．

5.2 制　　限

制限に関するオペレータは，アクションが制限されるかどうかを判定するための補助遷移規則を用いて次のように書かれる．

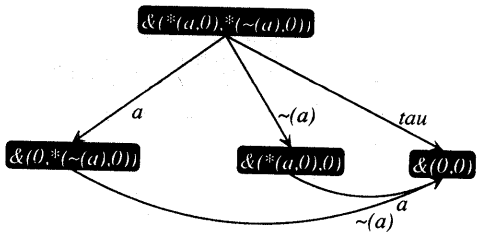


図 2　合成オペレータを持つプロセス
Fig. 2　A process with composition operator.

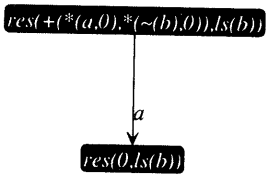


図 3　$(a.NIL + \overline{b}.NIL)\backslash\{b\}$
Fig. 3　$(a.NIL + \overline{b}.NIL)\backslash\{b\}$.

```
rule P1 - A -> P2 restricted(A,L) --> ff =>
        res(P1,L) - A -> res(P2,L) .
```

制限は，相補的なアクションに対しても有効になる．**図 3** の例では b を制限しているので，それと相補的な関係にある $\overline{b}$ も制限される．

5.3 再　　帰

再帰に関する遷移規則

$$\frac{P\{\mu x.P/x\} \xrightarrow{A} P_2}{\mu x.P \xrightarrow{A} P_2}$$

は次のように記述できる．

```
rule
  replacing(P,X,rec(X,P)) --> replaced(P1)
  P1 - A -> P2
    => rec(X,P) - A -> P2 .
```

再帰では，変数への項の代入の操作が重要な役割を果たすが，それは **replacing** というオペレータによって実現される．ところで，遷移式

```
replacing(P,X,rec(X,P)) --> replaced(P1)
```

のソース **replaced(P1)** を単に **P1** と記述することも可能であるが，その場合，遷移式のソースとターゲットは必ず同一のソートの台であることから，**replacing** はソート **process** の台を構成するオペレータであることになる．しかしながら，本来プロセスを構成する

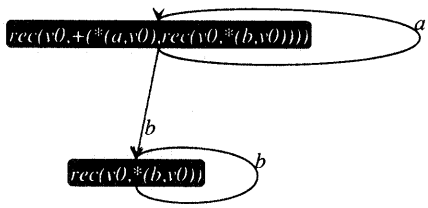


図4 $\mu v_0.(a.v_0 + \mu v_0.b.v_0)$
Fig. 4 $\mu v_0.(a.v_0 + \mu v_0.b.v_0)$.

オペレータは + や & や rec などに限られており，これらと replacing を同列のものとするのは好ましくない．そこで，process とは異なる別のソートを用意し，replacing や replaced がそのソートの台を構成するようにしている．

図4の例では，二つの v_0 がどう束縛されているかを正しく判断して遷移を導出している．

5.4 その他のオペレータ

この章では，合成，制限，再帰の三つのオペレータについて取り上げたが，CCS にはその他にアクションプレフィックス，選択およびリラベリングのオペレータがある．このうち，アクションプレフィックスと選択のオペレータに関しては補助的な遷移規則を用いることなく記述できる．リラベリングについては，リラベリング関数を実現するための補助遷移規則を用いて記述される．

6. 結　　論

第2章において，プロセス計算を形式的に記述する方法として多ソート SOS 形式を定義した．多ソート SOS 形式と同様にプロセス計算を SOS スタイルで記述する手法として TSS[7] と GSOS[1),3)] がある．これらと多ソート SOS 形式を比較すると，TSS や GSOS ではプロセスは単一ソートの項であり，またアクションは構造を持たないラベルであった．これに対し多ソート SOS 形式ではプロセスもアクションも多ソートの項で表される．

ところで，TSS や GSOS ではそのクラスにおける合同性などに関する一般的性質の解明が進んでいる[1)~3),5),7)]．多ソート SOS 形式のクラスにおいて同様な一般的性質の解明をすることは興味深い内容である．特に，定義 2.10 の条件を満たす遷移関係が唯一存在するときそれをプロセス計算のモデルと定義したが，モデルが唯一存在するとは限らない．GSOS ではモデルが唯一存在することが保証されており[3)]，多ソート SOS システムにおいてモデルが唯一存在するための条件を見極めることは重要な問題である．

第5章において CCS を記述したが，そのほかの代表的なプロセス計算に π-calculus[13)] がある．π-calculus を LCC で記述することは容易であるが，記述されたものを **ProCSuS** を用いて解析することはできない．π-calculus の入力アクションに関する遷移規則は次のようなものである．

$$\frac{}{x(z).P \xrightarrow{x(w)} P\{w/z\}} \; w \notin \mathrm{fn}((z)P)$$

この遷移規則に現れる変数 w は，ソース $x(z).P$ には現れない自由な[7)]変数である．したがって $x(z).P$ という形のプロセスのプロセスグラフを導出しようとすると，枝の数が無限にできてしまい有限の手続きでは導出できない．

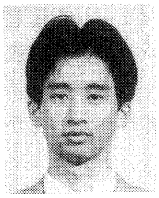
ProCSuS をさらに実用的なシステムにしていくためには，プロセス計算をモジュール化して記述できるような枠組を LCC に付加することが考えられる．その場合二つの多ソート SOS 形式を結合するようなモデルと結合したときの性質の変化の解明を行う必要がある．

参 考 文 献

1) Aceto, L.: GSOS and Finite Labelled Transition Systems, Technical Report, School of Cognitive and Computing Sciences, University of Sussex (1993).
2) Aceto, L., Bloom, B. and Vaandrager, F.: Turning SOS Rules into Equations, *Information and Computation*, Vol.111, No.1, pp.1–52 (1994).
3) Bloom, B., Istrail, S. and Meyer, A. R.: Bisimulation Can't Be Traced: Preliminary Report, *15th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pp.229–239 (1988).
4) Bolognesi, T. and Smolka, S. A.: Fundamental Results for the Verification of Observational Equivalence: a Survey, *Proc. IFIP Workshop on Protocol Specification, Testing and Verification VII* (Rudin, H. and West, C.H.(eds.)), pp.165–179, North-Holland (1987).
5) Fokkink, W. J.: The Tyft/Tyxt Format Reduces to Tree Rules, *Proc. of the 2nd International Symposium on Theoretical Aspects of Computer Software, Lecture Notes in Computer Science* (Hagiya, M. and Mitchell, J. C.(eds.)), Tohoku University, pp.440–453, Springer-Verlag (1994).
6) Formal Systems (Europe) Ltd: *Failures Divergence Refinement User Manual and Tutorial Version 1.2β* (1992). obtained

from ftp://ftp.comlab.ox.ac.uk/pub/CSP/FDR/public.info/manual/manual.ps.
7) Groote, J. F. and Vaandrager, F.: Structured Operational Semantics and Bisimulation as a Congruence, *Information and Computation*, Vol.100, No.2, pp.202–260 (1992).
8) Hoare, C. A. R.: *Communicating Sequential Process*, Prentice Hall (1985).
9) ISO: *Information processing systems - Open Systems Interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour*, chapter 7, pp.25–70, ISO (1989). ISO 8807.
10) Laboratory for Foundations of Computer Science, University of Edinburgh: *The Edinburgh Concurrency Workbench (Version 7)* (1994).
11) Milner, R.: A Calculus of Communicating Systems, *Lecture Notes in Conputer Science*, Vol.92, Springer-Verlag (1980).
12) Milner, R.: *Communication and Concurrency*, Prentice Hall (1989).
13) Milner, R., Parrow, J. and Walker, D.: A Calculus of Mobile Processes, I, II, *Information and Computation*, Vol.100, No.1, pp.1–77 (1992).
14) Plotkin, G. D.: A Structural Approach to Operational Semantics, Technical Report, Computer Science Department, Aarhus University (1981). DAIMI FN-19.
15) Togashi, A., Yoshida, S., Kimura, S. and Shiratori, N.: **ProCSuS**: A Meta System for Concurrent Process Calculi based on SOS, *Proc. of the International Workshop on Theory and Practice of Parallel Programming, Lecture Notes in Computer Science*, Vol.907, pp.229–234, Springer-Verlag (1995).
16) Yoshida, S., Togashi, A. and Shiratori, N.: Integrated Support Environment for Concurrent Process Calculi, *Proc. of the 13th Annual ACM Symposium on Principles of Distributed Computing*, ACM SIGACT and SIGOPS, acm press, p. 395 (1994).
17) 稲垣康善，坂部俊樹：抽象データタイプの代数的仕様記述法の基礎 (1) 多ソート代数と等式論理，情報処理，Vol.25, No.1 (1984).
18) 臼井伸幸，富樫 敦，白鳥則郎：Graphical Interface for Representation of LTS，第50回情報処理学会全国大会論文集 (5)，pp.161–162 (1995).
19) 吉田 仙，富樫 敦，白鳥則郎：プロセス計算の統合支援環境の構築，京都大学数理解析研究所講究録，Vol.902, pp.156–178 (1995).
20) 川口研治，神長裕明，高橋 薫，白鳥則郎，野口正一：LOTOS 仕様の等価性検証システムの構築，電子情報通信学会技術報告，IN89-59, Vol.89, No.131, pp.13–18 (1989).
21) 富樫 敦：代数的プロセスの計算モデル，日本ソフトウェア科学会チュートリアルテキスト (1992).

(平成7年 3 月17日受付)
(平成7年10月 5 日採録)



吉田 仙（正会員）

1970年生．1993年東北大学工学部情報工学科卒業．1995年同大学院情報科学研究科修士課程修了．同年日本電信電話（株）入社．現在，NTT コミュニケーション科学研究所に勤務．並行計算，分散協調処理に興味を持ち，マルチエージェントシステムにおける協調プロトコルの記述に関する研究に従事．

富樫 敦（正会員）

1979年山形大学工学部卒業．1984年東北大学大学院工学研究科博士課程修了．現在，同大助教授（電気通信研究所）．並行計算，プロセス代数，帰納推論に基づくプログラム合成および新たな計算パラダイムに関する研究に従事．日本ソフトウェア科学会，ACM 各会員．

白鳥 則郎（正会員）

1946年生．1977年東北大学大学院博士課程修了．1984年同大助教授（電気通信研究所）．1990年同大教授（工学部情報工学科）．1993年同大教授（電気通信研究所）．情報通信システム，ソフトウェア開発環境，ヒューマンインタフェースの研究に従事．1993年本会マルチメディア通信と分散処理研究会主査．本会25周年記念論文賞受賞．IEEE，電子情報通信学会，人工知能学会各会員．