プログラミング-言語・基礎・実践- 16－9
(1994. 3. 9)

並行プロセス計算の開発・利用支援環境

吉田 仙　富樫 敦　白鳥 則郎

東北大学電気通信研究所

〒980 仙台市青葉区片平 2-1-1

あらまし

並列に動作するプロセスの振舞いを扱う形式記述技法にプロセス計算がある。これまでに多くのプロセス計算が開発され、またそれらの処理系が作成されてきた。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、このことがプロセス計算の開発及び利用に関して大きな障害となる。そこで我々はプロセス計算の開発・利用のための支援環境を構築した。本稿ではまずプロセス計算の構文規則と遷移規則を記述する能力を持つプロセス計算記述言語 LCC を紹介し、さらにこの言語のインタプリタとして多様なプロセス計算の動的意味を解析するシステム **ProCSuS** の概略を紹介する。

和文キーワード　プロセス計算、動的意味、遷移関係、支援環境

A support environment for developping or utilizing concurrent process calculi

Sen Yoshida, Atsushi Togashi and Norio Shiratori

Research Institute of Electrical Communication

Tohoku University

2-1-1, Katahira, Aoba-ku, Sendai, 980, Japan

Abstract

A process calculus is a Formal Description Technique dealing with the behavior of multi agents. 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.

英文 key words　process calculus, operational semantics, transition relation, support environment

1 はじめに

はじめに、並行プロセス計算の開発・利用支援環境を構築するに至った背景、および本論文の目的について述べる。

近年、情報処理システムの並列化に伴い、並列システムの仕様記述を形式的に行ない、計算機で処理できるようにするための手段として、形式記述技法 (FDT: Formal Description Technique) の重要性が高まっている。その中で、並列に動作するプロセスの振舞いを形式的に扱うプロセス計算 [5] が注目され、これまでに多数開発されてきた。CCS [2] 、LOTOS [1] 、π-calculus[3, 4] 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、動的意味としての遷移関係を調べることを目的とする処理系が、個々のプロセス計算に対し作成されている。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、このことがプロセス計算の開発及び利用に関して大きな障害となる。そこで、プロセス計算の開発・利用を支援するシステムに対する重要度が高まっている。

プロセス計算の統合支援環境の実現により期待される点を挙げると、以下のようになる。

- 既存のプロセス計算の処理系が容易に作成できる。
- 新たなプロセス計算を設計する場合に、設計されるプロセス計算上でのプロセスの遷移関係をグラフィカルに表示したり、プロセス間の等価性を自動判定したりすることによって設計を支援できる。
- 多様なプロセス計算を統一的に扱う形式的手法を与えることができる。

以上の理由から、本稿ではプロセス計算の構文規則と遷移規則を記述する能力を持つメタ言語 LCC を紹介し、さらにこの言語のインタプリタとして様々なプロセス計算の処理系を作成するシステム **ProCSuS** の実装例を紹介する。

2 プロセス計算記述言語

プロセス計算の統合支援環境では、様々なプロセス計算の記述を可能にするメタ言語が必要になる。この章では、プロセス計算を記述するための言語として設計された、プロセス計算記述言語 LCC (Language of Concurrent process Calcului) [6] を紹介する。

2.1 LCC

LCC は、プロセス計算の構文規則と遷移規則を記述するためのメタ言語である。既存のプロセス計算や新たに設計されるプロセス計算の構文規則と遷移規則をこの言語で記述し、**ProCSuS** で解釈実行することによって、そのプロセス計算上でのプロセスの性質を調べることができるようになる。

LCC は、代数仕様記述言語に似た構文を持つ。代数仕様記述言語は、書換え規則に基づき項書換えを行なうシステム (TRS: Term Rewriting System) としての操作的意味を持つ。これに対し、 LCC の操作的意味とは遷移規則に基づきプロセスの遷移関係を導出することである。

2.2 LCC の説明

LCC は、 **header**、 **signature**、 **body**、 **footer** の 4 つの部分に大別される。**header** はプロセス計算の記述の開始を表し、またプロセス計算の名前を定義する。 **signature** では構文規則が抽象データ型 (ADT: Abstract Data Type) で記述される。**body** は遷移規則を記述する部分である。最後の **footer** はプロセス計算の記述の終了を示す。

以下で LCC の 4 つの部分をそれぞれ説明する。

2.2.1 header

header は次のように記述される。

calculus ⟨CalcName⟩ *is*

イタリック体で書かれた *calculus* と *is* は LCC の予約語であり、 *calculus* が記述の開始を示す。⟨CalcName⟩ の部分にはプロセス計算に付けられた名前が入る。

2.2.2 signature

signature は以下のように構成される。

⟨SortPart⟩
{{ ⟨SubSortPart⟩ }}
⟨OpPart⟩
{{ ⟨SetPart⟩ }}

⟨SortPart⟩ はプロセス計算で扱うデータの種類の名前（ソート）の宣言部である。ここで宣言されるソートの間に包含関係がある場合は、その関係を ⟨SubSortPart⟩ で定義する。プロセス動作式やアクションなどを構成するオペレータは、⟨OpPart⟩ において列挙される。プロセス計算が集合を扱う場合は、その名前と要素を ⟨SetPart⟩ で定義する。

⟨SortPart⟩ は、次のような形をしている。

sorts ⟨SortKey⟩ ... .

予約語 *sorts* の後に、そのプロセス計算で扱われるデータの種類 ⟨SortKey⟩ が宣言される。

ここで宣言されるソートの他に、LCC は組み込みソートとして **internal**、**set**、**int** などを持っている。これらのソートは LCC の中でそれぞれ特別な意味を持つ。**internal** はプロセスが行なうアクションのうち外部からは観測不可能な内部アクションを表しており、特に観測等価の判定において重要な意味を持つ (3.3 節参照)。**set** は集合を表しており、⟨SetPart⟩ で具体的に定義される。**int** は自然数の集合であり、算術式の中で用いることができる (2.2.3 参照)。

⟨SortPart⟩ で宣言されたソートの間に包含関係がある場合は、その関係が ⟨SubSortPart⟩ の中で次のように定義される。

subsorts (⟨SortKey⟩ <)... ⟨SortKey⟩ .

不等号 < は、その右側にある ⟨SortKey⟩ が表すソートが左側のものを包含するということを意味する。このとき、あるソートを包含するソートに対して定義されたオペレータは、包含されるオペレータに対しても定義されたとみなされる。また、1 つのソートが 2 つ以上のソートの subsort となることも許され、そのときにはそれらのソートの全てのオペレータが継承される。組み込みソート **internal** の subsort に属するオペレータは内部アクションを構成する。

オペレータの定義は下のようなものである。

ops ⟨OpForm⟩ ... *:*
{{ ⟨SortKey⟩ ... }} –> ⟨SortKey⟩ .

⟨OpForm⟩ ... が定義されるオペレータのリストであり、ソートおよび引数のソートが同じオペレータが一度に定義される。*:* と –> にはさまれた部分はそのオペレータがとる引数 (arity) のソートのリストである。–> に続く ⟨SortKey⟩ はオペレータにより構成される項 (co-arity) のソートを指定する。arity が記述されていない場合、そのオペレータはソート ⟨SortKey⟩ に属する定数であるとみなされる。ここで定義されたオペレータを用いて、プロセスやアクションを表す項が prefix-with-parentheses-and-commas 記法で構成される。

プロセス計算ではしばしば集合を扱う。その集合を定義する ⟨SetPart⟩ は、以下のように記述される。

set ⟨SetName⟩ *:* ⟨Term⟩ ... .

⟨SetName⟩ が集合の名前であり、“*:*” の後の ⟨Term⟩ ... が集合の要素である。各要素は上のオペレータの定義から得られる項でなくてはならない。ここで定義された集合について、ある項が集合の要素となっているかどうかという判定を遷移規則の付帯条件として記述することができる (2.2.3 節参照)。

2.2.3 body

プロセス計算の遷移規則は一般的に下のような証明図の集まりとして与えられる。

$$\frac{E \xrightarrow{l} E' \qquad F \xrightarrow{\bar{l}} F'}{E \mid F \xrightarrow{\tau} E' \mid F'}$$

このような遷移規則を記述するのが **body** である。**body** は以下のように構成される。

{{ ⟨VarPart⟩ }}
⟨RulePart⟩

⟨RulePart⟩ は遷移規則の記述であり、その規則の中で用いられる変数項が ⟨VarPart⟩ において宣言される。

遷移規則の記述の中に現れる全ての変数は ⟨VarPart⟩ において宣言され、ソートが割り当てられる。その記法は以下のようなものである。

vars ⟨VarName⟩ ... : ⟨VarSort⟩ ... .

⟨VarName⟩ が宣言される変数であり、⟨VarSort⟩ はその変数が束縛される項のソートを表す。⟨VarSort⟩ としては、⟨SortPart⟩ で定義される通常のソートのほかに、組み込みソートを指定することもできる。組み込みソート **set** が割り当てられた変数は、⟨SetPart⟩ で定義されたデータの名前 ⟨SetName⟩ に束縛され、⟨SetName⟩ が表す集合が代入されたとみなされる。

遷移規則は以下のように記述される。

rule {{ ⟨Trans⟩ ... }} => ⟨Trans⟩
{{ *where* ⟨Conditional⟩, ... }} .

予約語 *rule* と => に挟まれた部分は遷移規則の仮定となる遷移のリストである。矢印の後に遷移規則の結論が書かれ、最後に付帯条件が記述される。遷移規則の仮定と結論は全てアクションを伴うプロセスからプロセスへの遷移で表され、その記法は下のようなものである。

⟨Term⟩ − ⟨Term⟩ −> ⟨Term⟩

1つの遷移は3つの項の組で表される。一番左の項は遷移前のプロセスの動作式であり、真中の項がプロセスが遷移する時に行なうアクション、そして一番右の項が遷移後のプロセスの動作式である。

ところで、遷移規則の仮定として下のようにあるプロセスが遷移を行なうことが出来ないという条件を記述したい場合がある。

$$\frac{\cdots \quad \neg\exists E' : E \xrightarrow{a} E' \quad \cdots}{\cdots}$$

LCC ではこのような仮定の記述も可能であり、下のように書かれる。

⟨Term⟩ − ⟨Term⟩ −/−>

プロセス計算において、例えば CCS の再帰のように、ある項の部分項を別な項で置き換えるということがよくある。このことを記述するために LCC は組み込みオペレータ **repl** を提供している。**repl** は3つの項を引数に取り項を構成するオペレータであり、それぞれの引数の意味は、1番目が置き換えられる部分項を含む項全体、2番目が置き換えられる部分項、3番目が2番目の部分項のあった所に新たに置き換わる項である。この置き換えは、それを使った遷移規則を用いて遷移関係を導出する時点で実際に行なわれる。

遷移規則の付帯条件には以下のようなものがある。

⟨Term⟩ *contained* ⟨SetName⟩
⟨Term⟩ *not contained* ⟨SetName⟩
⟨Number⟩ *is* ⟨ArthmExp⟩

最初の記法は、⟨Term⟩ が表す項が ⟨SetName⟩ という名前の集合の要素であるという意味の条件文である。2番目の記法はその逆で、⟨Term⟩ が表す項は ⟨SetName⟩ という名前の集合の要素には含まれないという意味である。最後の記法は算術式を評価する場合に用いる。⟨ArthmExp⟩ という算術式を評価するとその値は ⟨Number⟩ になるという意味であり、⟨ArthmExp⟩ は組み込みソート **int**、**nat**、**no0nat** のいずれかに属する数と、+、* などの算術オペレータから構成される。組み込みソートの範囲はそれぞれ、**int** が整数、**nat** が 0 を含む自然数、**no0nat** が 0 を含まない自然数となる。

2.2.4 footer

プロセス計算の記述は下の1行によって終了する。

endcalc

2.3 LCC による記述例

簡単な例として、オペレータにアクションプレフィックスと選択だけを持つ単純なプロセス計算を LCC で記述すると、以下のようになる。

```
calculus BP is
sorts action expression .
ops a b c : -> action .
op 0 : -> expression .
op * : action expression -> expression .
op + : expression expression -> expression .
var A : action .
vars E El El1 El2 Er Er1 Er2 : expression .
rule => *(A,E) - A -> E .
rule El1 - A -> El2 =>
  +(El1,Er) - A -> El2 .
rule Er1 - A -> Er2 =>
  +(El,Er1) - A -> Er2 .
endcalc
```

最初の行は **body** であり、このプロセス計算が BP と名付けられていることを示している。

2 行目から 6 行目までが **signature** である。このプロセス計算では action と expression という 2 種類のソートを扱う。ソート action には a, b, c という定数項が属し、 0 はソート expression の定数である。5 行目はアクションプレフィックス * を定義している。* は action と expression の 2 つの引数をとるオペレータであり、それにより作られる項は expression となる。次の行は選択を表すオペレータ + の定義である。+ の arity は 2 つの expression であり、co-arity は expression である。

7 行目から 11 行目までは **body** である。変数 A は action に属する項に束縛され、E、El、 El1 などは expression に属する項に束縛される。第 9 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、アクションプレフィックスによって構成される項からは、いつでもアクションを行なうことが出来るという意味の規則が書かれている。10 行目と 11 行目は選択オペレータに関する規則の記述であり、2 つの引数のプロセスのうちどちらかがアクションを行なうことが出来る場合、そのアクションが選択実行される。

最後の行が **footer** で、以上で BP の記述が完了する。

3 プロセス計算支援システム

本章では、第 2 章で紹介した LCC のインタプリタと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。また、 **ProCSuS** のツールを CCS に適用した例を紹介する。

3.1 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、動的意味としての遷移関係を調べるシステムである。**ProCSuS** は現在、プロセスの導出木のグラフィカル表示とプロセス間の等価性判定の 2 つ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及び利用支援系として活用することが出来る。

ProCSuS の構成は 図 1 の様な概念図で表される。

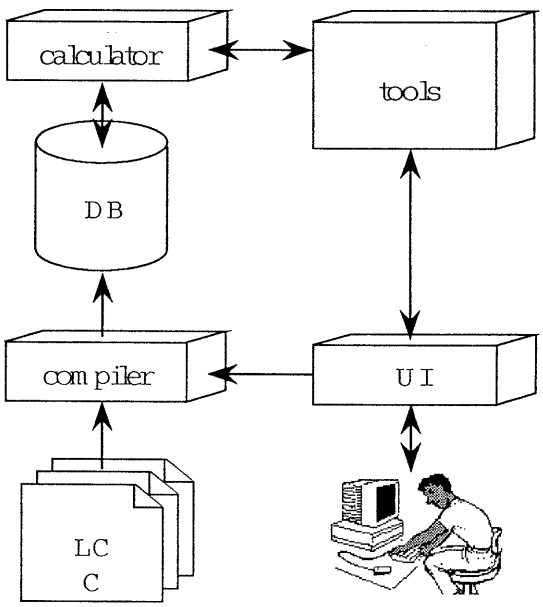

図 1: ProCSuS の構成概念図

ProCSuS は、コンパイラ、データベース、計算器、ツール、ユーザインタフェースの 5 つのモジュールから構成される。コンパイラは、

プロセス計算を記述した LCC ファイルを読み込み、計算器が参照できるような形に変換してデータベースに蓄積する。データベースはプロセス計算に関する様々なデータを蓄積し、必要に応じて計算器に情報を提供する。計算器は、データベースを参照しながら遷移規則の仮定や付帯条件を判断し、プロセスの遷移関係を調べる。ツールは計算器を駆使してプロセスの導出木を作成したり、等価性を判定したりする。ユーザインタフェースはコマンド入力を受け付けたりツールの出力を画面に表示したりする。

ProCSuS は、以前我々が作成した汎用並行プロセス計算システム GSC [6] を高機能化したものである。**ProCSuS** は Prolog を用いて実装されており、X Window System 上で動作する。

3.2 導出木表示

プロセスの遷移関係を表したラベル付き木を導出木という。**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導出木を作成して表示するツールが用意されている。

このツールは、指定されたプロセス計算に対し、プロセス動作式が入力されるとそのプロセスの遷移関係を計算器を使って調べ、任意のアクションによって到達可能な全てのプロセスを列挙する。さらにそれら遷移後のプロセスから到達可能なプロセスを調べるということを繰り返すことによって、与えられたプロセスをルートとする導出木を作りウィンドウに表示する。

3.3 等価性判定

2つのプロセスの間の等価性を判断する基準として、双模倣等価と観測等価という2つの概念がある。等価性判定ツールは、有限のプロセスについてこれらの等価性を調べることが出来る。

等価性判定ツールは、次のようにして2つのプロセス p, q が双模倣等価であるかどうかを判定する。双模倣等価の定義については文献 [2] を参照されたい。双模倣等価を $\sim$ で表す。

1. $p\ Id\ q$ ならば $p \sim q$ である。
2. 計算器を用いて、p から生起可能な全てのアクションの集合 $\mathcal{A}_p$ を作る。
3. $\mathcal{A}_p = \phi$ ならば 7 へ。
4. $a \in \mathcal{A}_p$ について、$p \xrightarrow{a} p'$ ならば、$q \xrightarrow{a} q'$ となるプロセス q' が存在するかどうかを計算器で調べる。存在しなければ $p \not\sim q$。
5. $p' \sim q'$ であるかどうかを、1 に戻って調べる。$p' \not\sim q'$ ならば $p \not\sim q$。
6. $\mathcal{A}_p = \mathcal{A}_p - \{a\}$ とし、3 へ。
7. 計算器を用いて、q から生起可能な全てのアクションの集合 $\mathcal{A}_q$ を作る。
8. $\mathcal{A}_q = \phi$ ならば $p \sim q$。
9. q について 4 から 6 と同様に調べる。

LCC における観測等価（弱等価）は次のように定義される。$\mathcal{P}$ をプロセスの集合、$\mathcal{A}$ をアクションの集合、$\mathcal{A}_{int} \subseteq \mathcal{A}$ をソートが組み込みソート **internal** の subsort である内部アクションの集合、$\mathcal{A}_{ext} = \mathcal{A} - \mathcal{A}_{int}$ とする。

定義 3.1 $t \in \mathcal{A}^*$ ならば、$\hat{t} \in \mathcal{A}^*_{ext}$ は t から $\mathcal{A}_{int}$ に属するアクションを全て取り除くことによって得られるアクション系列である。 □

定義 3.2 $t = a_1 \cdots a_n \in \mathcal{A}^*$, $t_{int1}, \ldots, t_{intn+1} \in \mathcal{A}^*_{int}$ ならば、$E \stackrel{t}{\Rightarrow} E'$ は

$$E(\stackrel{t_{int1}}{\rightarrow})^* \stackrel{a_1}{\rightarrow} (\stackrel{t_{int2}}{\rightarrow})^* \cdots (\stackrel{t_{intn}}{\rightarrow})^* \stackrel{a_n}{\rightarrow} (\stackrel{t_{intn+1}}{\rightarrow})^* E'$$

を表す。 □

定義 3.3 プロセス上の 2 項関係 $R \subseteq \mathcal{P} \times \mathcal{P}$ が弱双模倣であるとは、pRq ならば、任意の $a \in \mathcal{A}$ について、次の2つの条件が成り立つことである。

- $p \xrightarrow{a} p'$ ならば、ある q' が存在して $q \stackrel{\hat{a}}{\Rightarrow} q'$ かつ $p'Rq'$ を満たす。
- $q \xrightarrow{a} q'$ ならば、ある p' が存在して $p \stackrel{\hat{a}}{\Rightarrow} p'$ かつ $p'Rq'$ を満たす。

□

定義 3.4 プロセス p, q が観測等価であるとは、ある弱双模倣 R に対して、pRq が成り立つことである。 □

観測等価を $\approx$ で表す。観測等価を調べる方法は双模倣等価を調べる方法とほぼ同様である。

ところで、双模倣等価と観測等価の間には以下の関係がある。

命題 3.1 双模倣等価ならば観測等価である。 □

このことから、プロセス間の等価性のレベルを以下の 4 つに分けて定義する。

identical 恒等関係 $p\ Id\ q$

strong 双模倣等価 $p \sim q$

weak 観測等価であるが強等価でない
$p \approx q, p \not\sim q$

different 異なる $p \not\approx q$

2 つのプロセス動作式を入力すると、ツールはそれらの間の等価性が上の 4 つのレベルのうちどのレベルにあるかを調べ、答える。

3.4 実行例

以下において、既存の代表的な並行プロセス計算である CCS について **ProCSuS** を用いて遷移関係を調べた例を紹介する。

CCS など多くのプロセス計算ではオペレータの形として infix 記法を用いているが、LCC では prefix-with-parentheses-and-commas 記法しか許されていないので、ここでは実際の CCS の記法とは異なる形のオペレータを用いている。その対応関係は以下のようになっている。

オペレータ	CCS	LCC
silent action	τ	`tau`
prefix	$a.E$	`*(a,E)`
summation	$E+F$	`+(E,F)`

CCS における $a.0+\tau.b.0$ と $a.0+b.0+\tau.b.0$ という 2 つのプロセスの性質を調べてみる。それぞれのプロセス動作式を LCC の記法で書き直すとそれぞれ、`+(*(a,0),*(tau,*(b,0)))` と `+(*(a,0),+(*(b,0),*(tau,*(b,0))))` となる。これらの導出木を **ProCSuS** の導出木表示ツールを用いて描くと図 2 , 3 のようになる。

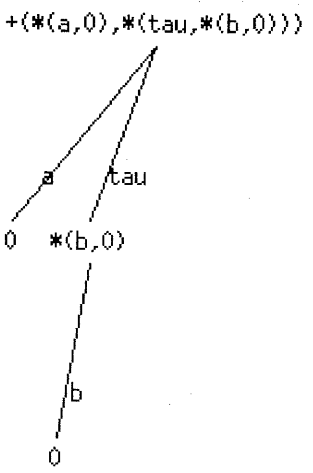

図 2: $a.0+\tau.b.0$

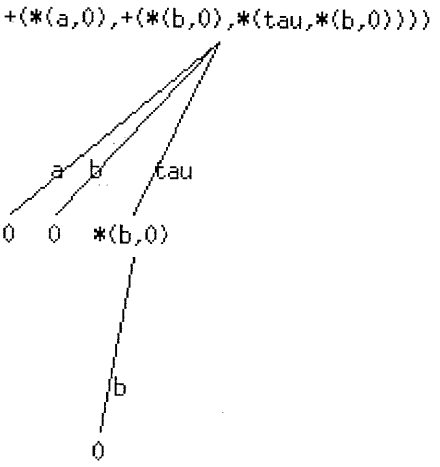

図 3: $a.0+b.0+\tau.b.0$

次に、等価性判定ツールを用いて上の 2 つのプロセスの間の等価性を判定してみる。ツールに 2 つのプロセス動作式を入力すると、以下のように自動判定された結果が出力される。

```
| ?- equivalence(
   '+(*(a,0),*(tau,*(b,0)))',
   '+(*(a,0),+(*(b,0),*(tau,*(b,0))))',
   Equivalence).
```

```
Equivalence = weak ?

yes
| ?-
```

等価性が weak であるとは、3.3 節で述べたように、観測等価であるが双模倣等価でないという意味である。

4 まとめ

この章では、LCC 及び **ProCSuS** に対する評価を行ない、それを元に本稿の結論を述べ、最後に今後の課題を検討する。

4.1 結論

3.4 節において、CCS の動的意味を **ProCSuS** を用いて解析した例を示した。CCS を LCC に翻訳する段階では、オペレータの記法に制限があることが問題となったが、遷移規則などの意味的な解釈は変更することなく記述することができた。導出木表示ツールや等価性判定ツールは、有限プロセスに限れば有効性を確認することができた。

また、本稿では触れていないが、basic LOTOS や π-calculus など CCS 以外のプロセス計算に対しても **ProCSuS** が処理系として機能することが確認されている。このことから、LCC はプロセス計算を記述するメタ言語として十分な能力を持ち、また **ProCSuS** が汎用処理系として有効に機能するということができる。

4.2 今後の課題

今後実現させていくべき課題を検討する。まず、無限プロセスのラベル付遷移システムを遷移規則から導出する手法を開発することが必要である。これを用いれば、ループを含む無限プロセスのグラフ表示や、等価性の自動判定といった機能の実現が期待できる。また、ことなる 2 つのプロセス計算上のプロセスの間で等価性を考えることも興味深い話題である。

さらに、論理式で表されたプロセスの仕様と実際のプロセスの間で、検証やプロセスの自動合成を行なうシステムも試作されている。

謝辞

本研究は一部、旭硝子財団、文部省科学研究補助金による援助を受けている。

参考文献

[1] Beinksma, E.: "A tutorial on LOTOS", pp73-84, *Diaz, M. Ed., Proc. IFIP Workshop on Protocol Specification, Testing and Verification V*, North-Holland (1986).

[2] Milner, R.: "Communication and Concurrency", *Hoare, C. A. R. Ed. Prentice Hall International Series in Computer Science*, Prentice Hall (1989).

[3] Milner, R., Parrow, J., Walker, D.: "A Calculus of Mobile Process, I", pp1-40, *Journal of Information and Computation 100*, Academic Press (1992).

[4] Milner, R., Parrow, J., Walker, D.: "A Calculus of Mobile Process, II", pp41-77, *Journal of Information and Computation 100*, Academic Press (1992).

[5] 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).

[6] 吉田仙, 木村成伴, 富樫敦, 白鳥則郎: "汎用並行プロセス計算システムの設計開発", pp27-36, 電子情報通信学会技術研究報告 *COMP93-23* (1993).