数理解析研究所講究録
902巻 1995年 156-178

プロセス計算の統合支援環境の構築*

東北大通研　吉田 仙　(Sen Yoshida)
東北大通研　富樫 敦　(Atsushi Togashi)
東北大通研　白鳥 則郎　(Norio Shiratori) †

1　はじめに

近年，情報処理システムの分散化・並列化に伴い，並行システムの仕様記述を形式的に行ない計算機による仕様の検証や自動合成を可能にするための手段として，形式記述技法に対する重要性が高まっている．その中で，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 [13] が注目され，これまでに多数開発されてきた．CCS [7]，CSP [5]，ACP [1] 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 [12]．し

*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．

†{yoshida,togashi,norio}@shiratori.riec.tohoku.ac.jp

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システムを構築することが望まれる．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表示したり，プロセス間の等価性を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まとめて扱う形式的手法を与えることができる．

以上の理由から本論文では，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抽象プロセス計算を定義する．また第3章では抽象プロセス計算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** の実装例を第4章で紹介する．最後に第5章でまとめと今後の課題について述べる．

2 抽象プロセス計算

前章で触れたように，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．本論文では，プロセス計算を抽象的なレベルでとらえそれらの一般的な性質を調べるための土台として，抽象プロセス計算を定義する．抽象プロセス計算は，プロセス計算の操作的意味を SOS [8] 的アプローチによって与える手法であり，個々のプロセス計算特有の表現からプロセス計算の持つ意味を独立させることができる．

はじめに，抽象プロセス計算の構文を定め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 [9] 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，遷移式言語ではプロセスや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)
シグニチャはソートの集合 S と演算記号領域 Σ との対 $\langle S, \Sigma \rangle$ である．ここで Σ は集合族 $\langle \Sigma_{w,s} \rangle_{w \in S^*, s \in S}$ 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． □

定義 2.2 (遷移式言語)
遷移式言語は 4 項組 $\langle S, \Sigma, X, \rightsquigarrow \rangle$ 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．$\rightsquigarrow$ は遷移を表す論理記号である． □

定義 2.3 (項)

$\langle S, \Sigma, X, \rightsquigarrow \rangle$ を遷移式言語，Y を $Y_s \subseteq X_s$ であるような任意の変数集合族 $\langle Y_s \rangle_{s \in S}$ とする．このとき各ソート s に対して，$T[\Sigma(Y)]_s$ を次の条件を満たす最小の集合とする．

1. $\Sigma_{\varepsilon,s} \cup Y_s \subseteq T[\Sigma(Y)]_s$
2. $f \in \Sigma_{s_1 \cdots s_n, s}$ かつ $\xi_i \in T[\Sigma(Y)]_{s_i}$ $(i = 1, \cdots, n)$ ならば $f(\xi_1, \cdots, \xi_n) \in T[\Sigma(Y)]_s$

$T[\Sigma(Y)]_s$ の元をソート s の $\Sigma(Y)$ 項という．特に $T[\Sigma(X)]_s$ の元をソート s の項ということがある．また，シグニチャ $\langle S, \Sigma \rangle$ について，$T[\Sigma]$ は各 $T[\Sigma]_s$ が変数を含まない項の集合 $T[\Sigma(\emptyset)]_s$ であるような集合族である．$T[\Sigma]_s$ 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は，遷移の前後の状態を表す項と，遷移時に行な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中で起こるものとする．

定義 2.4 (遷移式)

遷移式言語 $\langle S, \Sigma, X, \rightsquigarrow \rangle$ における遷移式は記号列 $\xi \overset{\alpha}{\rightsquigarrow} \eta$ である．ここで ξ と η は同じソートの $\Sigma(X)$ 項である． □

既存のプロセス計算において各プロセスの動作は遷移規則によって規定される場合が多い．抽象プロセス計算では遷移規則が条件付き遷移式として与えられる．

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

$$\frac{\{\varphi_i \mid i \in I\}}{\varphi}$$

ここで I はインデックスの集合であり，各 φ_i $(i \in I)$ と φ は $\mathcal{L}$ の遷移式である． φ_i $(i \in I)$ を遷移規則の仮定， φ を結論という． □

定義 2.6 (抽象プロセス計算)
$\mathcal{L} = \langle S, \Sigma, X, \rightsquigarrow \rangle$ を遷移式言語とする．抽象プロセス計算は遷移式言語 $\mathcal{L}$ と $\mathcal{L}$ における遷移規則の集合 Γ との対 $\langle \mathcal{L}, \Gamma \rangle$ である． $\langle \mathcal{L}, \Gamma \rangle$ を $\langle S, \Sigma, X, \Gamma \rangle$ と略記することがある． □

抽象プロセス計算上での遷移の導出は形式論理体系における定理の証明と同様にして行なわれる．すなわち，ある遷移式を導出することは仮定がなく結論がその遷移式であるような遷移規則の証明を与えることに相当する．

定義 2.7 (代入)
$\langle S, \Sigma, X, \rightsquigarrow \rangle$ を遷移式言語とする．このとき，代入 σ は写像族 $\langle \sigma_s : X_s \to T[\Sigma(X)]_s \rangle_{s \in S}$ である．

σ を項から項への写像族

$$\langle \sigma_s : T[\Sigma(X)]_s \to T[\Sigma(X)]_s \rangle_{s \in S}$$

に拡張する．すなわち，任意の $\xi \in T[\Sigma(X)]_s$ に対して，

1. $\xi = x \in X_s$ ならば $\sigma_s(\xi) = \sigma_s(x)$

2. $\xi = f \in \Sigma_{\varepsilon,s}$ ならば $\sigma_s(\xi) = f$

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

さらに，σ を遷移式および遷移規則にも拡張する． □

定義 2.8 (証明)

Γ を遷移規則の集合，

$$\frac{\{\varphi_i \mid i \in I\}}{\varphi}$$

を遷移規則とする．この遷移規則の Γ における証明とは，上向きの枝を持ち，すべての上向きの道が有限であり，以下の性質を持つ木である．すなわち，各ノードが遷移式によってラベル付けされており，根がラベル φ を持ち，かつ各ノードについて，以下のいずれかが成り立つ．

- ノードがラベル φ_i $(i \in I)$ を持ち，そのノードの上にはノードが存在しない．

- ノードがラベル ψ を持ち，その真上のノードのラベルの集合が $\{\psi_j \mid j \in J\}$ であり，かつ遷移規則 $\gamma \in \Gamma$ と代入 σ が存在し $\sigma(\gamma)$ が

$$\frac{\{\psi_j \mid j \in J\}}{\psi}$$

となる．

□

定義 2.9 (証明可能性)

Γ を遷移規則の集合，γ を遷移規則とする．γ が Γ から証明可能であるとは，Γ における γ の証明が存在することである．また，仮定のない遷移規則

$$\frac{\emptyset}{\varphi}$$

が Γ から証明可能であるとき，遷移式 φ は Γ から導出されるといい，$\Gamma \vdash \varphi$ と書く． □

次に抽象プロセス計算の操作的意味を与える方法を示す．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．抽象プロセス計算では，状態やアクションがシグニチャ $\langle S, \Sigma \rangle$ の項であるようなラベル付き遷移システムを特に Σ ラベル付き遷移システムと呼び，その上でプロセスの意味が与えられる．この方法は LOTOS における構造的ラベル付き遷移システムを用いた操作的意味論 [6] を拡張したものといえる．

定義 2.10 (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 $\langle \mathcal{P}, \mathcal{A}, \mathcal{T}, p_0 \rangle$ である．ここで $\mathcal{P}$ は状態の非空集合，$\mathcal{A}$ はアクションの集合である．$\mathcal{T} = \{\xrightarrow{a} | a \in \mathcal{A}\}$ は遷移関係 $\xrightarrow{a} \subseteq \mathcal{P} \times \mathcal{P}$ の族である．$p_0 \in \mathcal{P}$ は初期状態である． □

定義 2.11 (Σ ラベル付き遷移システム)

$\langle S, \Sigma \rangle$ をシグニチャとする．Σ ラベル付き遷移システム $LTS[\Sigma]$ はラベル付き遷移システム $\langle \mathcal{P}, \mathcal{A}, \mathcal{T}, p_0 \rangle$ である．ここで，あるソート $s_{state} \in S$ に対し，$\mathcal{P} \subseteq T[\Sigma]_{s_{state}}$ である．また，$\mathcal{A} \subseteq \bigcup_{s \in S} T[\Sigma]_s$ である． □

定義 2.12 (導出)

$\mathcal{C} = \langle S, \Sigma, X, \Gamma \rangle$ を抽象プロセス計算とする．項 $\xi \in T[\Sigma(X)]s$ $(s \in S)$ の導出 $Der_{\mathcal{C}}(\xi)$ は以下の条件を満たす最小の集合である．

1. $\xi \in Der_{\mathcal{C}}(\xi)$
2. $\zeta \in Der_{\mathcal{C}}(\xi)$ かつある α について $\Gamma \vdash \zeta \overset{\alpha}{\rightsquigarrow} \eta$ ならば，$\eta \in Der_{\mathcal{C}}(\xi)$

□

定義 2.13 (プロセス)

抽象プロセス計算 $\mathcal{C} = \langle S, \Sigma, X, \Gamma \rangle$ におけるプロセスとはある

ソート $s_p \in S$ の閉じた項の集合 $T[\Sigma]_{s_p}$ の要素である．プロセス $\xi \in T[\Sigma]_{s_p}$ の操作的意味は Σ ラベル付き遷移システム $\langle \mathcal{P}, \mathcal{A}, \mathcal{T}, \xi \rangle$ によって与えられる．ここで $\mathcal{P} = Der_{\mathcal{C}}(\xi)$, $\mathcal{A} = \cup_{s \in S} T[\Sigma]_s$ である．また $\mathcal{T}$ は各 $\alpha \in \mathcal{A}$ に対し $\xrightarrow{\alpha} = \{(\zeta, \eta) \mid \zeta, \eta \in \mathcal{P},\ \Gamma \vdash \zeta \overset{\alpha}{\rightsquigarrow} \eta\}$ となるような遷移関係の集合族 $\langle \xrightarrow{\alpha} \rangle_{\alpha \in \mathcal{A}}$ である． □

3 プロセス計算記述言語

この章では，抽象プロセス計算をモデルに持つ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．

3.1 LCC

LCC は，プロセス計算の構文規則と遷移規則を抽象的に記述することによって汎用処理系でその意味解釈を行なえるようにするためのプログラミング言語である．この言語は抽象プロセス計算をモデルとして持ち，実際の構文もモデルとしての遷移式言語に強く依存している．ただし実装上の問題として，演算記号や変数，遷移規則などの集合はすべて要素数が有限に制限される．また，記述の容易性を増すために LCC ではモデルにはない機能も付加されている．

3.2 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** ではシグニチャ，すなわちソートと演算記号などが宣言される．**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．

3.2.1 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⟩ はプロセス計算に付けられた名前である．

3.2.2 signature

signature は以下のように構成される．

⟨SortPart⟩
{{ ⟨SubSortPart⟩ }}

⟨OpPart⟩

{{ ⟨SetPart⟩ }}

⟨SortPart⟩ はソートの宣言部である． LCC ではあるソートの台の部分集合に名前をつけることができ，そのような名前をサブソートと呼ぶ．ソートとサブソートの間の包含関係は ⟨SubSortPart⟩ で定義される．各ソートの演算記号は ⟨OpPart⟩ において列挙される． LCC ではいくつかの項を集めて集合を作ることができ，その名前を ⟨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ソート ⟨SortKey⟩ が宣言される．

ここで宣言される通常のソートの他に， LCC は組み込みソートとして **internal** を持っている．このソートは意味解釈の段階においてそれぞれ特別な意味を持つ． **internal** の台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とみなされる．

⟨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⟩ .

記号 < は，その右側にある ⟨SortKey⟩ の台が左側のものを包含するということを意味する．組み込みソート **internal** の項は意味解釈の段階で内部アクションとみなされる．

演算記号の定義は下のようなものである．

ops ⟨OpForm⟩ . . . :
{{ ⟨SortKey⟩ . . . }} −> ⟨SortKey⟩ .

⟨OpForm⟩ . . . が定義される演算記号のリストであり，演算記号のソートと被演算記号のソートがすべて同じ演算記号が一度に定義される．: と −> にはさまれた部分は被演算記号のソートのリストである．−> に続く ⟨SortKey⟩ は演算記号により構成される項がどのソートの台となるかを指定する．

3.2.3 body

プロセス計算の遷移規則は **body** 部で以下のように記述される．

{{ ⟨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⟩ ... が宣言される変数のリストであり, ⟨VarSort⟩ はそれらがどのソートの変数であるのかを指定する. ⟨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, 解釈時において ⟨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⟩ .

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式のリストであり, 矢印の後に遷移規則の結論が書かれる. 遷移規則の仮定と結論は全て遷移式であり, その記法は下のようなものである.

⟨Term⟩ – ⟨Term⟩ –> ⟨Term⟩

1つの遷移式は3つの項とその間の矢印から構成される. 一番左の項は遷移前のプロセス, 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を意味している.

3.2.4 footer

プロセス計算の記述は下の 1 行によって終了する.

endcalc

3.3 LCC による記述例

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プロセス計算 BPAtau を LCC で記述すると, 以下のようになる.

calculus BPAtau *is*

sorts act proc .

subsorts internal < act .

ops a b c *:* −> act .

op tau *:* −> internal .

op 0 *:* −> proc .

op * *:* act proc −> proc .

op + *:* proc proc −> proc .

var A *:* act .

vars E El El1 El2 Er Er1 Er2 *:* proc .

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

次に **signature** が記述されている． BPAtau では act と proc という 2 種類のソートを扱う．ソート act の台には a, b, c およびサブソート internal の台である tau という定数項が属し， 0 はソート proc の定数項である． * は act と proc 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 proc となる．次の行は選択を表すオペレータの定義である． + は proc の台から 2 つの引数をとり proc の項を構成する．

続いて **body** が記述される，ソート act の変数は A であり，proc の変数は E, El, El1 などである．最初の rule 文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．次の 2 つは選択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と解釈される．

最後の行が **footer** で，以上で BPAtau 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．

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，プロセスのラベル付き遷移システムのグラフ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 の様に表される．**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，シグニチャや遷移規則に関する情報を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，ツールの要求に応じてデータベース内の情報を参照しながらプロセスの遷移関係を導出する推論エンジンである．ツールは，計算器に導出関係を導出させることによってプロセスの操作的意

味としてのラベル付き遷移システムを作成したり，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アプリケーションの出力を画面に表示したりする．

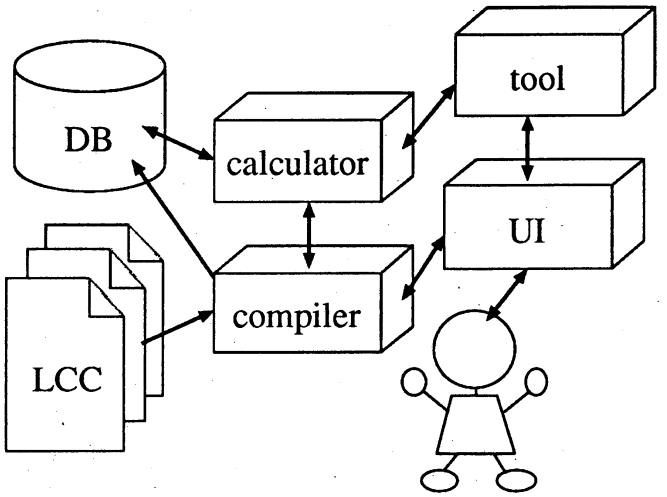

図 1: ProCSuS の構成

ProCSuS は SICStus Prolog [1] を用いて実装されており，X Window System 上で動作する．

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を初期状態とするラベル付き遷移システムを作成してグラフ表示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から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計算器を用いて枚挙する．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

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

り返すことによって，与えられたプロセスを初期状態とするラベル付き遷移システムを作り上げる．これをラベル付き有向グラフの形でウィンドウに表示する．

プロセスの間の等価性を判断する基準には様々なものがあるが，**ProCSuS** のツールは双模倣等価と観測等価という 2 つの等価性を有限のプロセスについて調べることが出来る．ここで，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 つのプロセスを入力すると，**ProCSuS** 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．

例として，次に示す 2 つのプロセスの間の等価性を判定してみる．

P_1 = *(a,+(*(b,0),*(tau,*(c,0))))

P_2 = +(*(a,*(c,0)),*(a,+(*(b,0),*(tau,*(c,0)))))

まずそれぞれのプロセスのラベル付き遷移システムを表示させると，図 2のような出力が得られる．さらに，2 つのプロセスの

間の等価性を自動判定させた結果を図 3に示す．結果が weak となっているのは 2 つのプロセスが双模倣等価ではないが観測等価であるという解析結果を示している．

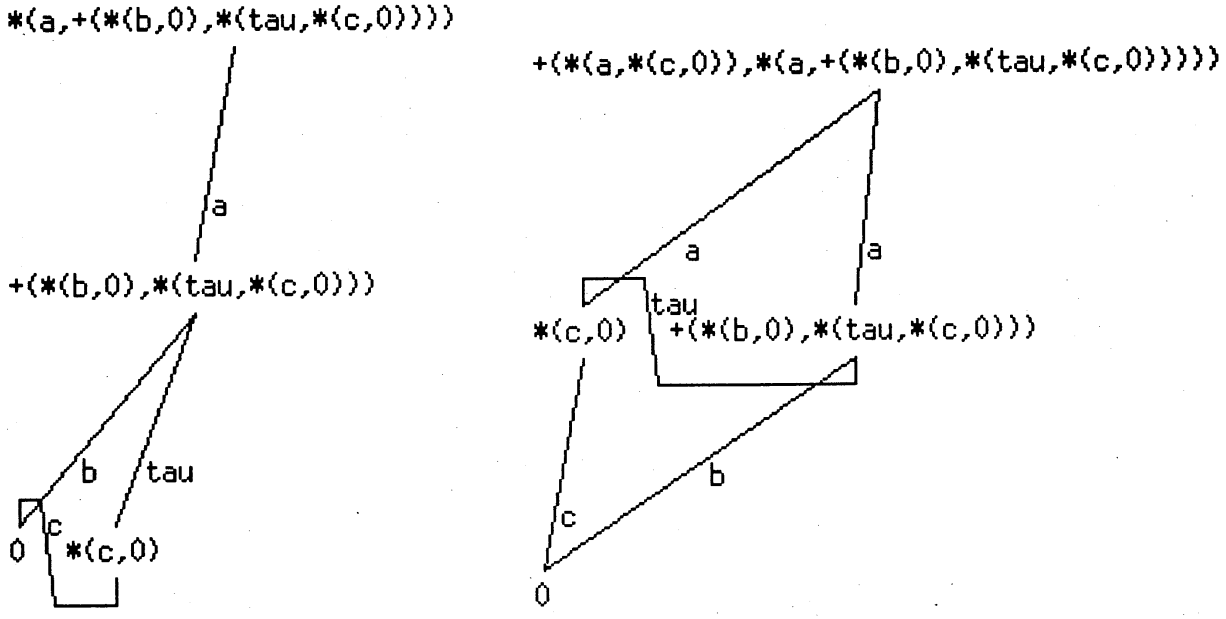

図 2-a: P_1　　図 2-b: P_2

図 2: 観測等価なプロセス

5　おわりに

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抽象プロセス計算を定義した．抽象プロセス計算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 [4] と GSOS [2] がある．これらと抽象プロセス計算を比較すると，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．これに対し抽象プロセス計算ではプロ

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

Equiv = weak ?

yes
| ?-
```

図 3: 等価性の判定結果

セスもアクションも多ソートの項で表される．このことは多くの点でプロセス計算の記述を容易で明確なものにしている．特にアクションに構造が伴う CHOCS や π-calculus などのプロセス計算を自然に記述できるのが抽象プロセス計算の特徴である．

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 [4, 3, 2]．抽象プロセス計算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．また，GSOS では遷移規則にあるアクションによって遷移することができないという負の仮定を記述することができるが，抽象プロセス計算にもこれを導入することが考えられる．

第 4章でプロセス計算の支援環境を提供するシステムを紹介し

た．このシステムの機能として，有限のプロセスの間の等価性判定を行なえるのであるが，同値類分割の手法を用いることによって再帰を含むプロセス間でも等価性を判定でき，そのアルゴリズムをインプリメントすることは早急に達成しなければならない課題である．

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2つの抽象プロセス計算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なう必要がある．

参考文献

[1] J. A. Bergstra and J. W. Klop. *Algebra of communicating process*, Vol. 1 of *de Bakker, J. W., Hazewinkel, M., Lenstra, J. K. eds. CWI Monographs.* North-Holland, 1986.

[2] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced: Preliminary report. In *15th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pp. 229–239, 1988.

[3] W. J. Fokkink. The Tyft/Tyxt fromat reduces to tree rules. In M. Hagiya and J. C. Mitchell, editors, *Proc. of the 2nd Inter-*

national Symposium on Theoretical Aspects of Computer Software, Lecture Notes in Computer Science, pp. 440–453. Tohoku University, Springer-Verlag, 1994.

[4] J. F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, Vol. 100, No. 2, pp. 202–260, 1992.

[5] C. A. R. Hoare. *Communicating Sequential Process*. Prentice Hall, 1985.

[6] 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.

[7] R. Milner. A calculus of communicating systems. *Lecture Notes in Conputer Science*, Vol. 92, , 1980.

[8] G. D. Plotkin. A structural approach to operational semantics. Technical report, Computer Science Department, Aarhus University, 1981. DAIMI FN-19.

[9] 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.

[10] 吉田仙, 富樫敦, 白鳥則郎. 並行プロセス計算の開発・利用支援環境. 電子情報通信学会技術報告, Vol. 93, No. 496, pp. 65–72, 1994. COMP93-85.

[11] 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. 電子情報通信学会技術報告, Vol. 93, No. 123, pp. 27–36, 1993. COMP93-23.

[12] 情報処理学会. 特集：通信システムの形式記述技法の標準化. 情報処理, Vol. 31, No. 1, pp. 2–81, 1990.

[13] 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.