# アクターシステムのためのトレース意味論

**Towards trace semantics for actor systems**

ヴァスコンセロス ヴァスコ*、所 真理雄†

**Summary.** The theory of traces is used to describe the behavior of actor systems. The semantics is built from two simple concepts: a set of events representing the reception of messages by objects, and a binary symmetric and irreflexive relation on events—independence—representing permissible concurrency. Causality, the dual notion of concurrency, is expressed by the dependence relation—the complement of independence. A particular execution of a system is described by a trace: a labeled acyclic graph where nodes are labeled with events and the only edges are between nodes labeled with dependent events. The behavior of a system is viewed as the set of traces representing all possible executions.

## 1 はじめに

従来の多くの並行オブジェクト指向計算のための意味論はインタリーヴを基本としたものであった。これらのモデルでは、並行性をインタリーヴとして表現する複数のイベントによってシミュレートすることにより、「並行性」の概念を「逐次性」と「非決定性」に置き換え表現する。インタリーヴモデルは、小数のプリミティブな概念から構成されているため扱い易い。さらに、インタリーヴモデルは逐次性のモデルから多くの結果を受け継ぐことが可能なので、並行性の意味論の研究はインタリーヴモデルに基づくものが多かった。一方、並行性を直接扱う意味モデルは並行性の概念、又はそれの対応する概念である因果性を、本質的なものとして表現することができる。

インタリーヴ意味論は諸般なアクター [Agha 86] を含む一般的なオブジェクトのモデルのために構築されてきた。特に、この意味論は Plotkin の構造的操作性意味論 [Plotkin 81] の提案、そして Milner の CCS [Milner 89] 等のプロセス代数のために概念の構築に用いられる。一方、非インタリーヴの並行計算モデルにはペトリネット [Petri 77] や Winskel の event structures [Nielsen *et al.* 81, Winskel 88] や Mazurkiewicz のトレース [Aalbersg and Rozenberg

* Vasco T. Vasconcelos. 慶應義塾大学
† Mario Tokoro. 慶應義塾大学、ソニーコンピュータサイエンス研究所

88] がある。これらのモデルはペトリネットのために元来考え出されたものである。Meseguer の concurrent term rewriting logic [Meseguer 90, Meseguer 91] も真の並行計算のモデルであり、様々な並行的システムを表現できる。プロセス代数やペトリネットのためのいくつかの研究 [Boudol and Castellani 90, Aalbersg and Rozenberg 88, Diekert 90, Mazurkiewicz 88] があるが、並行オブジェクト指向計算のための並行的意味論に関する研究事例は少ない。

本論文では細粒度オブジェクトのシステムの挙動を表現するために並行性と逐次性を記述できるモデルを考察する。このモデルでは並行オブジェクト指向のプログラムのための意味論を2段階で表現する。まず、標準的な表示的意味論でプログラムから、対応するオブジェクトシステムを構築し、そのオブジェクトシステムからトレース理論を用いて並行挙動を導き出す。本論文では特に第二段階に焦点を当てる。

特に、アクターシステムのための並行計算意味論はトレース理論 [Aalbersg and Rozenberg 88, Diekert 90, Mazurkiewicz 88] に基づいて提案する。トレース意味論はペトリネットの挙動を表現するために Mazurkiewicz によって計算機科学に導入されたもので、次の二つの概念の上に構築される。

- 原子アクションの集合 $X$
- 任意の二つのアクションの並行的に計算可能な表現の下での非依存関係 $I \subseteq X \times X$。

この関係は対称性と非反射性であることを仮定し、これらの仮定は並行性公理として表現される。対称性はどのアクションもそれ自身が同時に起こらないことを表現し、非反射性はアクションの並行性が相互に生じるという公理を表現する。

本論文で提案される概念に基づいたアクターシステムの構成演算は [Vasconcelos and Tokoro 92] と [Vasconcelos 92] に示されている。関連研究 [Agha 86, Hewitt and Baker 77, Clinger 81, Mazurkiewicz 88, Meseguer 90] と比較すると、本論文の定理等の証明は [Vasconcelos and Tokoro 92] と [Vasconcelos 92] に示されている。そして、以下に本論文の構成を述べる。次節では、アクターシステムを概説する。2節では、アクターシステムの初期コンフィギュレーションから導出可能なすべての逐次遷移の集合によりアクターシステムをインターリーブ意味論として表現する。その後、次節では非依存的なイベントの概念を導入し、5節では逐次遷移からイベントのトレースを抽出する方法を与える。そして最後の節では結論を述べる。

## 2 アクターシステム

アクターの元となるプログラミング言語のプログラムはアクターの実行可能な挙動の表現と初期値のアクターとメッセージの宣言によって構成される。そして、これらのアクターとメッセージは、初期値のコンフィギュレーションを構成する。

アクターはメッセージを介して通信し、メッセージはアクターが交換するすべての情報を受け渡す。メッセージが運ぶ情報は通信集合 $K$に含まれる。各メッセージは一つの宛先アクターの識別子を持っている。アクターの識別子

は名前集合$N$要素である。すべての可能なメッセージの集合は$M = K \times N$である。これら、メッセージの中にある識別子を抜き出す関数$target : M \to N$を定義する。

各アクターには一つの識別子が割当てられてあり、受信したメッセージに対応するあらかじめ記述された挙動を実行する。アクターはメッセージを受信した時に、メッセージを送信し、アクターを生成し、自分の新しい挙動を決定する。そして、挙動は$B = M \to \mathcal{P}(M) \times \mathcal{P}(A) \times B$の集合によって示される。ただし$\mathcal{P}(M)$はメッセージのすべての有限な部分集合である。また、$\mathcal{P}(A)$はアクターのすべての有限な部分集合である。識別子と挙動の直積$A = N \times B$によってすべてのアクターが表現される。

アクターシステムの計算はコンフィギュレーションの中で起こる。コンフィギュレーションはメッセージの集合とアクターの集合で、$(\mu, \alpha)$の一対で表示する。この$\mu$はメッセージの有限な集合で、$\alpha$はアクターの有限な集合であり、同じ識別子を持つアクターは存在しない。$C = \mathcal{P}(M) \times \mathcal{P}(A)$はすべてのコンフィギュレーションの集合である。さらに、コンフィギュレーションからメッセージの集合とアクターの集合を抽出する関数$msgs : C \to \mathcal{P}(M)$と$actors : C \to \mathcal{P}(A)$を定義する。

**定義** 1 (アクターシステム) アクターシステムは次の7項組で表現する。

$$S = (N, K, M, B, A, C, c_i)$$

たたし、$N$は名前集合、$K$は通信集合、$M = N \times K$はメッセージ集合、$B = M \to \mathcal{P}(M) \times \mathcal{P}(A) \times B$は挙動集合、$A = N \times B$はアクター集合、$C = \mathcal{P}(M) \times \mathcal{P}(A)$はコンフィギュレーション、$c_i \in C$は$S$の初期値である。

**例** 1 (二つのメッセージのフォワーダ（転送者）) 二つのメッセージのフォワーダのアクターは受け取った最初の二つのメッセージを転送して、次のメッセージを無視する。より詳しく 述べると、「二つのメッセージのフォワーダ」の挙動*forward2* は、受け取ったメッセージをあるアクターに転送して、「一つのメッセージのフォワーダ」になることである。一方、「一つのメッセージのフォワーダ」の挙動*forward1* は、受け取ったメッセージは同じアクターに転送して、スィンク (*sink*) になることである。ここでスィンクの挙動は受け取った全てのメッセージも無視することである。ただし、式を読み易くするために、$(n, k)$ のメッセージは$k \triangleright n$ として、$(n, b)$ のアクターは$n{:}b$と略記する。挙動は次の関数で表現する。

$$\begin{aligned} forward2_n(k \triangleright f) &= \langle\, \{k \triangleright n\},\ \emptyset,\ forward1_n \,\rangle \\ forward1_n(k \triangleright f) &= \langle\, \{k \triangleright n\},\ \emptyset,\ sink \,\rangle \\ sink(k \triangleright f) &= \langle\, \emptyset,\ \emptyset,\ sink \,\rangle \end{aligned}$$

ただし、$k$は通信で、$f$と$n$はアクターの名前である。フォワーダのアクターと転送したメッセージを消費するアクター（スィンクの挙動を持つアクター)、さらにフォワーダに宛先を持つ三つのメッセージからなるコンフィギュレーションに関心がある。すなわち、

$$c_i = (\{x \triangleright f, y \triangleright f, z \triangleright f\}, \{f{:}forward2_u, u{:}sink\})$$

ただし、$x$, $y$, $z$は通信と$f$, $u$はアクターの名前である。 □

## 3 逐次的挙動

この節では、アクターシステムのための状態遷移に基づくインターリーブ的意味論が与えられる。この意味論は、次の節に述べられるより抽象的な並行意味論に用いる。インターリーブモデルでは、コンフィギュレーションにあるメッセージの手順に基づいてコンフィギュレーションの置き換えを行うことによりアクターシステムの計算を行う。ただし、コンフィギュレーション中に複数のメッセージがあれば複数の可能遷移が存在する。コンフィギュレーションの置き換えは遷移関数で表現する。

**定義 2** (遷移関数) $S=(N,K,M,B,A,C,c_i)$をアクターシステムとなる。$S$の遷移関数$\delta_S$ は部分関数で次のように定義される。

$$\delta_S : C \times M \to C$$

ただし、$m$が$c_0$中にある$n$に対応したメッセージであり、$n{:}b_0$が$c_0$含まれるアクターであり、さらに$b_0$が$m$によって定義可能である場合に限り、$\delta_S(c_0,m)=c_1$である。この場合、$b_0(m)=\langle \mu,\alpha,b_1\rangle$ より以下に成立する。[1]

$$\begin{aligned} msgs(c_0)-\{m\} &= msgs(c_1)-\mu \\ actors(c_0)-\{n{:}b_0\} &= actors(c_1)-(\{n{:}b_1\}\cup\alpha) \end{aligned}$$

ただし、生成したアクター$\alpha$の識別子は$c_0$の中に含まれないとする。$\mu$の中にあるメッセージは$msgs(c_0)$ に存在するならば、別のサブスクリプトで名前づけられているとする。

遷移関数はコンフィギュレーションの間で可能な1ステップ遷移を表現する。一つのメッセージが一つの遷移の原因となるメッセージ列は複数回の遷移列の原因としてみることができる。ただし、遷移列の作用は順番に実行される個々の全体の作用である。到達可能なコンフィギュレーションとは初期値のコンフィギュレーションから有限なメッセージ列による有限な遷移列からなるコンフィギュレーションのことである。

**定義 3** (到達可能関数) $S=(N,K,M,B,A,C,c_i)$をアクターシステムとする。$S$の到達可能関数 $R_S$ は部分関数で次のように定義される。

$$R_S : M^* \to C$$

ただし

$$\begin{aligned} R_S(\varepsilon) &= c_i \\ R_S(wm) &= \delta_S(R_S(w),m) \end{aligned}$$

任意の $m\in M$と $w\in M^*$となる。

到達可能関数の領域を$ST_S$で表し、$ST_S$の要素は $S$の逐次的遷移という。$ST_S$ は$S$システムのすべての逐次的遷移で、文字列からなる言語を構成している。同様に、到達可能関数の値域は$RC_S$と表し、$RC_S$の要素は$S$ の到達可能コンフィギュレーションという。$w$は逐次的遷移でそして$R_S(w)=c$より $w$ は$S$を $c$ に導くという。又、2項組 $(M,ST_S)$ は$S$の逐次的挙動と呼び、$SB_S$と表す。

[1] 差集合'−'は、'∩' より結合性が強い、'∩' は'∩' より結合性が強い。

例 2　次のコンフィギュレーションは例 1のアクターシステムにより到達可能なコンフィギュレーションである。

$$c = (\emptyset, \{f{:}sink, u{:}sink\})$$

ここで、$w = xyzx'y'$のイベント例は $S$の逐次遷移で $R_S(w) = c$ ある。ただし、$x \triangleright f$のイベントは $x$ に、$x \triangleright u$ は $x'$に省略し同様に $y$と $z$を省略する。逐次遷移 $w' = xyy'zx'$も $S$を $c$ に導く。これは、$w$と $w'$の遷移が同じ並行計算の二つの逐次的観測である。 □

## 4　非依存的イベント

前節ではアクターシステムの逐次的挙動を定義した。$SB_S$ のイベントの順序はイベントの因果関係だけでなく、並行イベントを逐次的な列として観測した順序を反映したものである。従って、イベントの同時到着による衝突を逐次化すること、並行実行の別の観測によるイベントの順序の相違の原因となるか決定するのに逐次的挙動の構造は十分ではない。$S$の逐次的挙動からイベントの因果の順序を抽出するために$SB_S$ がイベントの依存性の情報を持っている必要がある。

依存的なメッセージの概念は送り先のアクターによって同時的または依存的に処理される。非依存的なメッセージの一番簡単な例は異なるアクターに受信されるメッセージである。それは、アクターは並行的に実行されるので、別のアクターが別のメッセージを並行的に処理できるからである。同じアクターを宛先とするメッセージはどの順序で処理されても同じコンフィギュレーションとなる場合は非依存関係となる。

**定義** 4 (非依存関係)　$S = (N, K, M, B, A, C, c_i)$ はアクターシステムである。$S$の非依存関係は、$I_S \subseteq M \times M$、$(x, y) \in I_S$からなる最小の対称的非反射的関係である。ただし、

1. $target(x) \neq target(y)$ と、$x$ と $y$を含むコンフィギュレーションがある。
   又は
2. $target(x) = n = target(y)$ と、$x$ と $y$を含むどのコンフィギュレーションに対して $n{:}b$ は $c$ の中にあるアクターである。ただし、

$$b(x) = \langle \mu_0, \alpha_0, b_0 \rangle,\ b_0(y) = \langle \mu_{01}, \alpha_{01}, b_{01} \rangle$$
$$b(y) = \langle \mu_1, \alpha_1, b_1 \rangle,\ b_1(x) = \langle \mu_{10}, \alpha_{10}, b_{10} \rangle$$

次の対応がある。

$$\alpha_0 \cup \alpha_{01} = \alpha_1 \cup \alpha_{10}$$
$$\mu_0 \cup \mu_{01} = \mu_1 \cup \mu_{10}$$
$$b_{01} = b_{10}$$

上記の定義は、非依存的なイベントが計算の未来に影響を与えないということを示している。つまり、いかなる非依存的イベントの列は同じコンフィギュレーションに導く。コンフィギュレーションに非依存的イベントの因果性は図 1のような束として表せられ、次の補題として定式化される。

**補題** 1 $c$ はアクターシステム $S$のコンフィギュレーションで、$x$ と $y$が $c$ の中にある二つの非依存的なメッセージであるならば、$c'$は高々一つのコンフィギュレーションとある。ただし、$c' = \delta_S(\delta_S(c,x),y) = \delta_S(\delta_S(c,y),x)$ とする。

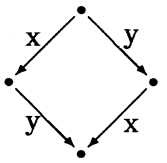


**Figure 1.** 非依存的なイベント $x$ と $y$による遷移は同じコンフィギュレーションに導出する （接点はコンフィギュレーションの表現で辺は遷移を表す)。

挙動が変わらないアクターを非順次と呼ぶ。以下の命題により、非順次なアクターは複数のメッセージを同時に処理することができる。

**命題** 2 到達可能なコンフィグレーションの中にあって、同じ非順次なアクターへの宛先を持つ二つのメッセージは非依存的である。

因果性は並行性の双対であり、依存関係によって記述される。ここで、依存関係というのは非依存関係の補完的関係である。ただし、$S$の依存関係は $D_S = M \times M - I_S$であり、$S$に関する並行的アルファベットは$\Sigma_S = (M, D_S)$ と定義される。

**例** 3 $S$は例 *1*に定義したアクターシステムとする。$S$の非依存関係 $I_S$は次の双対による最小の対称的関係から構築される。

$$(x',y'),(y',z'),(z',x')$$
$$(x,y'),(x,z'),(y,x'),(y,z')(z,x'),(z,y')$$

上段に述べされた双対は同じアクターへの宛先であり、命題 *2*により、双対なメッセージを含む到達可能なコンフィギュレーションが存在するならば、双対は非依存である。例えば、$R_S(xy)$ というコンフィギュレーションは $x'$と $y'$のメッセージを含んでいる。下段の双対は相違なアクターへの宛先のメッセージであり、双対のメッセージを含む到達可能なコンフィギュレーションの存在を容易に示すことができる。例えば、$R_S(y)$ のコンフィギュレーションは $(x,y')$ の双対を含んでいる。これは、$x$ と $y$は、初期コンフィギュレーション含めると図 *1*は導出である。しかし、$R_S(z)$ のコンフィギュレーションを含まないので非依存的なメッセージではない。$S$に関係づけられた並行アルファベットは$\Sigma_S = (M, D_S)$ となる。ただし、$D_S = M \times M - I_S$である。 □

## 5 並行的挙動

依存的なイベントの概念を用いると逐次的な遷移として表現されたイベントの全順序は部分順序に置き換えられる。この半順序には、依存的なイベント

だけが関係し、トレースとして表現される。このトレースは依存類または逐次遷移の同値類と考えられる。

ラベル付非循環グラフ（アルファベット $A$ 上の）は三項組 $(V, R, \varphi)$ である。ただし、$(V, R)$ は有限な有向な非循環グラフ, $V$ は接点の集合、$R \subseteq V \times V$ は辺の集合、$\varphi : V \to A$ はラベルを付ける関数である。ラベル付非循環グラフは異なる依存的なイベントでラベル付けられた接点間の辺だけで構成されると依存グラフとなる。

二つの依存グラフ $\gamma_0$ と $\gamma_1$ は辺とラベルを付ける関数を保存する接点の全単射があると $\gamma_0$ と $\gamma_1$ が同型写像で、$\gamma_0 \cong \gamma_1$ と記述する。$\Gamma_\Sigma$ はすべての依存グラフの同型写像の類で、$\lambda$ は空の依存グラフを表現する。そして、$\Gamma_\Sigma$ と掛け算 $\circ$ と単位元要素はモノイドを形成する。

ここで、逐次遷移から依存グラフへのマッピングを定義する。このマッピングは逐次遷移に関する依存グラフを抽出可能にする。

$$\langle\rangle_\Sigma : M^* \to \Gamma_\Sigma$$

ただし、各 $M^*$ の $u, v$ と $M$ の $e$ は次の関係が成り立つとする。

$$\begin{aligned} \langle\varepsilon\rangle_\Sigma &\cong \lambda \\ \langle e\rangle_\Sigma &\cong (\{e\}, \emptyset, \{(e,e)\}) \\ \langle uv\rangle_\Sigma &\cong \langle u\rangle_\Sigma \circ \langle v\rangle_\Sigma \end{aligned}$$

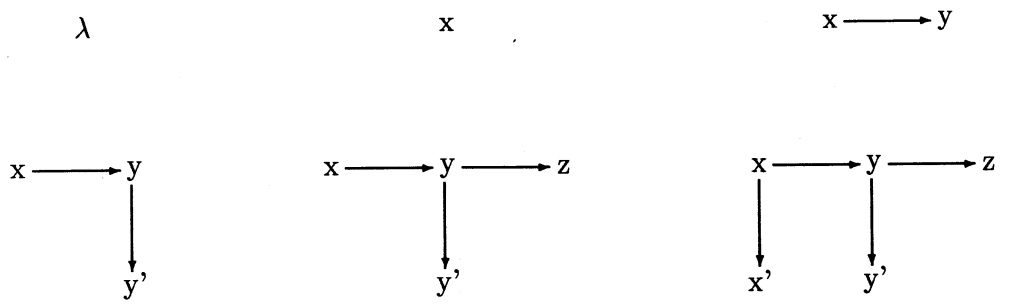


**Figure 2. 逐次遷移 $xyy'zx'$ に関する依存グラフの構築の順序（ただし、左から右、上から下方向）。**

例 4　$xyy'zx'$ のイベント例は二つのメッセージのフォワーダのアクターシステムの逐次遷移である。この逐次遷移と非依存的なイベントの情報（例 *3*）により、関係づけられた依存グラフを構築できる。空のグラフ $\lambda$ から始まり、イベント例の中の個々のイベントでラベル付けられた接点をグラフに付け加えて、各接点から新しい接点には依存的なイベントでラベル付けられた辺を書いていく。逐次遷移 $xyy'zx'$ に対して依存グラフの構築は図 *2* に表される。□

前に述べたように、トレースは同値類としても表現できる。逐次遷移の中に二つの非依存なイベントが続けて現れる場合、その順序は無関係である。逐次遷移とイベント列の同値類を置換することによって、一つ一つの非依存的なイベントの順序が抽出できる。$M^*$によってトレースの同値類関係 $\equiv_S$ は次のようなすべての一対から定義される。

$$(uxyv, uyxv) \text{ ただし、} u, v \in M^* \text{と} (x, y) \in I_S$$

イベント列 $w$を含む同値類 $\equiv_S$ は $w$のトレースと呼び、$[w]_S$で記述する。トレース理論の重要な定理は両項の表現の同値性を保証する。

**例 5** 例 1に定義したアクターシステムにより逐次遷移 $xyy'zx'$から生成した同値類、つまり、$xyy'zx'$のトレースは以下に示させる。

$$[xyy'zx']_S = \{xx'yy'z,\ xx'yzy',\ xyy'x'z,\ xyy'zx',\ xyx'y'z,\ xyx'zy',\ xyzx'y',\ xyzy'x'\}$$

これは図 2に示される初期グラフの別の表現でしかない。つまり、$[xyzx'y']_S$の要素はそのグラフの線形化である。 □

上記のトレース同値類 $\equiv_S$ の定義よりすべての逐次遷移はあるトレースに属する。逆に、以下の定理は各トレースの要素が逐次遷移で、すべての同値的な逐次遷移は同じコンフィギュレーションに導くことに保証する。

**定理 3** 到達可能関数$R_S$ はトレース同値に対して合同である。すなわち、$M^*$にあるそれぞれの二つの逐次遷移 $w_0$, $w_1$は

$$w_0 \equiv_S w_1 \Rightarrow R_S(w_0) = R_S(w_1)$$

アクターシステムの逐次遷移の集合$ST_S$によって、すべての依存グラフの集合は $\langle ST_S \rangle = \{\langle w \rangle_S \mid w \in ST_S\}$ というトレース言語となる。トレースシステム$CB_S = (\Sigma_S, \langle ST_S \rangle)$ は $S$の並行的挙動と定義され、$CB_S$の要素は $S$の並行的遷移と呼ぶ。一方、トレースは逐次遷移の同値類と見ると、$S$によるすべてのトレースの集合は $[ST_S] = \{[w]_S \mid w \in ST_S\}$ というトレース言語であり、$S$の並行的挙動は$CB_S = (\Sigma_S, [ST_S])$ となる。

$\sigma$と$\tau$は並行的アルファベットによる二つのトレースである。$\sigma$と$\tau$のトレース構成は、$\sigma\tau$と記述し、$\sigma\tau = [uv]_\Sigma$と定義する。ただし、それぞれに$u, v \in M^*$は$\sigma$と$\tau$の要素である。$\Sigma$によるトレース$\delta$が存在する場合$\tau = \sigma\delta$ において$\sigma$は$\tau$の接頭と定義され、トレース$\tau$すべての接頭の集合は $Pref(\tau)$ で記述される。$\Sigma$によってトレースの接頭順序は$\tau_1 \sqsubseteq \tau_2 \Leftrightarrow \tau_1 \in Pref(\tau_2)$ 関係で定義される。$\Sigma_S$によるすべてのトレースの集合は $\sqsubseteq$ で順序化されることは明かである。各トレース言語$T$によって、$Pref(T) = \bigcup_{\tau \in T} Pref(\tau)$ と定義し、$Pref(T)$の要素を$T$の接頭と呼ぶ。トレースシステムのトレース言語は閉接頭（prefix closed）があるとトレースシステムが閉接頭であるという。

**定理 4** $CB_S$は閉接頭のトレースシステムである。

トレース言語$T$の中にある各二つのトレースは、もし$T$の中のトレースの接頭であるならば$T$は有向的と呼ぶ。すなわち、$\tau_1, \tau_2 \in T \Rightarrow \exists \tau \in T : \tau_1 \sqsubseteq \tau$

and $\tau_2 \sqsubseteq \tau$の場合は$T$ は有向的である。トレース言語$T$は$T'$の部分集合あるとトレースシステム $X = (\Sigma, T)$ は $X' = (\Sigma, T')$ の部分システムである。$CB_S$の有向な部分システムはアクターシステムの計算の履歴を表現し、そして各部分システムは計算の可能な状態を表現し、全体システムは計算はそれを構成する全ての部分システムの状態を通して表現される。最大要素を持つ有向な$CB_S$の分システムは終了する計算の履歴を記述して、最大限の要素は計算の終りの状態を表現する、最大限のない有向な部分システムは停止しない計算を表現する。

例 6　例 *1*に定義されたアクターシステム $S$の並行挙動は次に示される依存グラフの集合と例 *3*に定義された並行アルファベット$\Sigma_S$ によって記述される。

$$\langle ST_S \rangle = Pref\left(\begin{array}{cc} x \to y \to z & \\ \downarrow, & \downarrow, \\ x', & y' \end{array}\right) \cup Pref\left(\begin{array}{cc} x \to z \to y & \\ \downarrow, & \downarrow, \\ x', & z' \end{array}\right) \cup Pref\left(\begin{array}{cc} y \to z \to x & \\ \downarrow, & \downarrow, \\ y', & z' \end{array}\right)$$
$$\cup Pref\left(\begin{array}{cc} y \to x \to z & \\ \downarrow, & \downarrow, \\ y', & x' \end{array}\right) \cup Pref\left(\begin{array}{cc} z \to x \to y & \\ \downarrow, & \downarrow, \\ z', & x' \end{array}\right) \cup Pref\left(\begin{array}{cc} z \to y \to x & \\ \downarrow, & \downarrow, \\ z', & y' \end{array}\right)$$

フォワーダが最初の二つのメッセージを受け取った順序に依存するシステムの計算の履歴は六つが存在する。つまり、この計算履歴は上記に示されている六つの最大要素を持つ有向部分システムである。

トレースは逐次遷移の同値類で表現されるならば、並行挙動$CB_S$は次式のすべての並行遷移と並行アルファベット$\Sigma_S$ によって記述される。

$$[ST_S] = Pref([xyzx'y']_S) \cup Pref([xzyz'x']_S) \cup Pref([yzxy'z']_S) \cup$$
$$Pref([yxzx'y']_S) \cup Pref([zxyz'x']_S) \cup Pref([zyxy'z']_S)$$

□

トレースは半順序多重集合（pomset）[Pratt 86] で考えると有用なことがある。半順序多重集合というのはラベル付部分順序の同型写像類である。ところが、並行アルファベット$\Sigma = (A, D)$ による依存グラフ$\gamma = (V, R, \varphi)$ はさらにラベル付部分順序 $(V, A, \leq, \varphi)$ である。すなわち、部分順序$\leq$は$\gamma$指令された辺の推移的閉包から与えられる。

## 6　結論

操作的モデルからトレース理論によって、アクターシステムのための単純な並行計算意味論を構築した。プログラムの意味はプログラムから導入したアクターシステムの並行挙動である。一方、システムの並行挙動はシステムの初期コンフィギュレーションからのすべての並行遷移を表す要素の集合である。各並行遷移はシステムの特有の計算によるトレースである。トレース理論は同型的に並行遷移を表す二つの異なる方法を与える。つまり依存グラフと逐次遷移の同値類である。

Hewitt と Baker のアクターシステムのための順序づけの方法 [Hewitt and Baker 77, Clinger 81] と比べて本論文はより抽象的な表現を導く。このため

本論文で示した意味論はこれは [Hewitt and Baker 77, Clinger 81] において異なるものとして表現されたものの同一性を導くことができる。しかし、トレース理論を安全ペトリネットの挙動の表現に適用するのに比べると複雑なものとなる。特に、アクターシステムの依存関係は動的概念により導くのは自明ではないが、それは可能な筈である。

**謝辞** 適切な助言と与えて下さったPeter Wegner教授と絶えまない御助力した頂いた本田耕平氏に深く感謝致します。また有意義な討議をして下さった米澤明憲教授と柴山悦哉教授、そして本論文の草稿に貴重なコメントを頂いた佐藤一郎氏に感謝致します。

## 参考文献

[Aalbersg and Rozenberg 88] J. Aalbersg and G. Rozenberg. Theory of traces. *Theoretical Computer Science*, Vol. 60, pp.1–82, 1988.

[Agha 86] Gul Agha. *Actors: A model of concurrent computation in distributed systems.* The MIT Press, Cambridge, MA, 1986.

[Boudol and Castellani 90] Gérard Boudol and Ilaria Castellani. Three equivalent semantics for CCS. In *Semantics of Systems of Concurrent Processes, Lecture Notes in Computer Science 469*, pp. 96–141. Springer-Verlag, April 1990.

[Clinger 81] William Clinger. Foundations of actor semantics. AI-TR 633, M.I.T. A.I. Laboratory, Cambridge, MA, May 1981.

[Diekert 90] Volker Diekert. *Combinatorics on traces*, volume 454 of *Lecture Notes in Computer Science.* Springer-Verlag, 1990.

[Hewitt and Baker 77] Carl Hewitt and Henry Baker. Laws for communicating parallel processes. In *IFIP*, pp. 987–992, August 1977.

[Mazurkiewicz 88] Antoni Mazurkiewicz. Basic notions of trace theory. In *Linear time, Branching Time and Partial Orders in Logics and Models for Concurrency, Lecture Notes in Computer Science 354*, pp. 285–363. Springer-Verlag, May 1988.

[Meseguer 90] José Meseguer. A logical theory of concurrent objects. In *OOPSLA/ECOOP*, pp. 101–115. ACM Press, October 1990.

[Meseguer 91] José Meseguer. Conditional rewriting logic as a unified model of concurrency. SRI-CSL 91-05, SRI-International, Menlo Park, CA, February 1991.

[Milner 89] Robin Milner. *Communication and concurrency.* C.A.R. Hoare Series Editor. Prentice-Hall Int., 1989.

[Nielsen *et al.* 81] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains. *Theoretical Computer Science*, Vol. 13, pp.85–108, 1981.

[Petri 77] C. A. Petri. Non-sequential processes. GMD-ISF 77-S, Gesselschaft Math., Datenverarb., St. Augustin, 1977.

[Plotkin 81] Gordon Plotkin. A structural approach to operational semantics. Daimi FN 91, Aarhus University, 1981.

[Pratt 86] Vaughan Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, Vol. 15, No. 1, pp.33–71, 1986.

[Vasconcelos 92] Vasco T. Vasconcelos. Trace semantics for concurrent objects. To appear as a technical report, Dep. of Computer Science, Keio University, February 1992.

[Vasconcelos and Tokoro 92] Vasco T. Vasconcelos and Mario Tokoro. Trace semantics for actor systems. In *Workshop on Object-based Computing, Lecture Notes in Computer Science 612*. Springer-Verlag, 1992.

[Winskel 88] G. Winskel. An introduction to event structures. In *Linear time, Branching Time and Partial Orders in Logics and Models for Concurrency, Lecture Notes in Computer Science 354*, pp. 364–397. Springer-Verlag, May 1988.