Vol. 37 No. 4　　情　報　処　理　　Apr. 1996

解　説

並行計算の理論の最近の動向

1. プロセス代数の意味論†

堀　田　英　一††

1. はじめに

本稿では広い意味でのプロセス代数と総称される並行プログラミング言語の意味論について解説する．プログラミング言語の意味論に対しては，いくつかのアプローチがあるが，ここでは表示的意味論と呼ばれるアプローチを中心にし，これとその他のアプローチの関係について述べる．

広い意味でプロセス代数と呼ばれる体系は多数あるが，その中でもCCS[12]，CSP[10]，ACP[1]，ATP[5]などがよく知られている．これらの言語の構文上の特徴は次のように代数的にプロセスを表現する点にある．まず基本プロセスを表す定数記号およびプロセスを合成するための演算記号の集合（シグニチャ）を定め，それから構成される代数項としてプロセスを定義する．また再帰的プロセスを表現するために不動点演算子が用いられることが多い．

直観的には，プロセス代数は互いに通信しながら並行動作をする種々の動作主体（agent）からなるシステムを表現していると考えられる．このような動作主体の典型的なものとしてはOSのプロセスがあげられる．プロセス代数の主要な記述対象は環境と相互作用をしながら持続的に動き続ける反応的（reactive）システムであり，この点は通常の逐次型言語が入力と出力（あるいは初期状態と終了状態）を対応付ける変換システムを主要な記述対象とするのと大きく異なる．このためプロセス代数の意味論においては逐次型言語の意味論とは異なる手法が要求される．

プロセス代数の意味論に対しても，逐次型言語に対するのと同様に，次の3つの代表的なアプローチがある．**表示的意味論**においては，プログラムが表す数学的対象の集合である意味領域を基礎に，シグニチャの各要素を意味領域上の関数として解釈することにより，複合的なプログラムの意味をその構成要素の意味から合成的に定義する．（このことの具体的な意味については**4.**，特に(7)を参照）．**操作的意味論**はプログラムの実行過程を抽象機械等を用いて表し，それを介して意味を定義するものであるが，プロセス代数の操作的意味論においてはラベル付き遷移システムを用いて実行過程を表すのが一般的である．またプロセス代数の**公理的意味論**は(不)等式論理に基づく公理系と推論規則として与えられる場合が多い[1),5),12)]．

意味論に対する異なるアプローチは，プログラムの意味に対する異なる観点に基づくものであり，各々の適用分野・有用性を持つ．ある言語に対して複数の意味論が存在するとき，ある意味論から得られる情報が，他の意味論から得られる情報と矛盾しないこと（整合性），および互いに補い合うこと（相補性）が重要となる．本稿では，表示的意味論を中心にして，これと他のアプローチの整合性・相補性について述べる．

以下，**2.**では簡単なプロセス代数とその操作的意味論を紹介し，**3.**では表示的意味論の基礎となる2種類の意味領域について述べる．**4.**では意味モデルが持つべき基本性質について論じる．**5.**では表示的意味論と操作的意味論の整合性について論じ，特に与えられた操作的意味論に最もよく整合する表示的モデル—完全抽象モデル—を構成する問題について述べ，代表的な結果を紹介する．**6.**では，表示的モデルを構成しその完全抽象性を証明するための代表的な枠組の1つを紹介し，その適用例として**2.**のプロセス代数に対し完全抽象表示的モデルを具体的に与える．

2. 簡単なプロセス代数の例

ここで紹介する言語$\mathcal{L}$は，文献1)で**空プロセスを含むプロセス代数**（PA_ε）と呼ばれるものを

† Semantics of Process Algebras by Eiichi HORITA (NTT Communication Science Laboratories).
†† NTTコミュニケーション科学研究所

若干簡略化したものである*．$\mathcal{L}$は次の文法により定義される．

$$(\mathcal{L}\ni)S::=\mathbf{O}|\mathbf{e}|(a\cdot S)|(S_1\|S_2)$$
$$|(S_1+S_2)|X|(\mu X.S).$$

ここで，aは原子動作の集合$\mathbf{A}$を動き，Xはプロセスを表す変数の集合を動くものとする．定数記号$\mathbf{O}$と$\mathbf{e}$は各々**無動作プロセス**と**空プロセス**を表し，演算記号$\|$，$+$，$a\cdot$は各々**並行合成**，**選択合成**，**動作前置**を表す．$(\mu X.S)$は再帰プログラムを表現するためのμ記法であり，直観的には"$X=S$となるプロセスX"を表す（一般にSはXを含む）．μは不動点演算子と呼ばれ，μXはそのスコープSの中のXを束縛する**．Σをこれらの定数記号・演算記号からなるシグニチャとすると，$\mathcal{L}$はΣとμ記法から作られる項の集合である．自由変数を持たない$\mathcal{L}$の要素をプログラムと呼び，プログラムの集合を$\mathcal{L}_\theta$で表す．またプログラムを表す変数としてsを用いる．$\mathcal{L}$は実際的なプロセス，たとえばOSのシステム・プロセスを記述するには単純化され過ぎているが，そのようなプロセスの骨格を記述するための言語と見ることができる．本章の最後にある例2では単純なシステム・プロセスの骨格を$\mathcal{L}$で記述する．

$\surd$を正常終了を表す記号とし，$\mathbf{A}_\surd=\mathbf{A}\cup\{\surd\}$と置く．$\mathcal{L}_\theta$上のラベル付き遷移関係$\overset{a}{\to}(a\in\mathbf{A}_\surd)$を次の規則によって与える．（ここで，$s\overset{a}{\to}s'$は$s$が$a$を実行した後$s'$になることができることを表し，$s\overset{\surd}{\to}$は$s$が正常終了できることを表す．下の規則(3)-(8)は推論規則と見なしてよく，水平線の上部が規則の前提を下部が結論を表す．）

(1) $(a\cdot s)\overset{a}{\to}s$. (2) $\mathbf{e}\overset{\surd}{\to}$.

(3) $\dfrac{s_1\overset{a}{\to}s_1'}{(s_1\|s_2)\overset{a}{\to}(s_1'\|s_2)}$. $\dfrac{s_1\overset{a}{\to}s_1'}{(s_2\|s_1)\overset{a}{\to}(s_2\|s_1')}$.

(4) $\dfrac{s_1\overset{\surd}{\to},s_2\overset{\surd}{\to}}{(s_1\|s_2)\overset{\surd}{\to}}$.

(5) $\dfrac{s_1\overset{a}{\to}s_1'}{(s_1+s_2)\overset{a}{\to}s_1'}$. $\dfrac{s_1\overset{a}{\to}s_1'}{(s_2+s_1)\overset{a}{\to}s_1'}$.

(6) $\dfrac{s_1\overset{\surd}{\to}}{(s_1+s_2)\overset{\surd}{\to}}$. $\dfrac{s_1\overset{\surd}{\to}}{(s_2+s_1)\overset{\surd}{\to}}$.

(7) $\dfrac{S[(\mu X.S)/X]\overset{a}{\to}s}{(\mu X.S)\overset{a}{\to}s}$.

(8) $\dfrac{S[(\mu X.S)/X]\overset{\surd}{\to}}{(\mu X.S)\overset{\surd}{\to}}$.

ここで，$s\in\mathcal{L}_\theta$に対して，$S[s/X]$はSの中のXの自由な出現をsで置き換えたものを表す．$a\in\mathbf{A}$に対して，$s\overset{a}{\to}$は$\exists s'[s\overset{a}{\to}s']$を表すものとする．$s\in\mathcal{L}_\theta$に対して，$\mathrm{act}(s)=\{a\in\mathbf{A}_\surd|s\overset{a}{\to}\}$と置く．$\mathbf{A}$の要素の有限列$w=\langle a_1,\cdots,a_n\rangle\in\mathbf{A}^*$に対して，$\overset{w}{\to}$は$\mathcal{L}_\theta$上の2項関係$\overset{a_1}{\to}$，…，$\overset{a_n}{\to}$を合成したものとする（$\mathbf{A}^*$は$\mathbf{A}$の要素の有限列の集合）．

以下，集合A，Bに対し，$A\triangleright B=(A^*\cdot B)\cup A^\omega$と置く（$\omega$は自然数全体の集合）．$A\triangleright B$は$A$の要素の有限列で$B$の要素で終るもの，および$A$の要素の無限列からなる集合である．

遷移関係$\overset{a}{\to}$を用いて$\mathcal{L}$に対する線形操作的意味モデル$\mathcal{O}$を定義する*．ここで，線形というのは$\mathcal{O}$における各プログラムの意味がある種の列—木等の分岐構造と対照して線形構造である—の集合として与えられることを表している．$\mathcal{O}$におけるプログラムsの意味はsが実行できる動作の列で正常終了するか，デッドロックに至るか，あるいは無限に続くものの集合として定義される．

定義1 δをデッドロックを表す記号とし，$\mathbf{E}=\mathbf{A}\triangleright\{\surd,\delta\}$と置く．意味関数$\mathcal{O}:\mathcal{L}_\theta\to\mathcal{P}(\mathbf{E})$を次のように定義する（ここで$\mathcal{P}(\mathbf{E})$は$\mathbf{E}$のべき集合）．各$s\in\mathcal{L}_\theta$と$v\in\mathbf{E}$に対して，$v\in\mathcal{O}[\![s]\!]$となるのは次の式(1)，(2)，あるいは(3)のどれかが成り立つときとする．

$$\exists w\in\mathbf{A}^*[v=w\cdot\surd\wedge\exists s'[s\overset{w}{\to}s'\wedge\surd\in\mathrm{act}(s')]]. \quad (1)$$

$$\exists w\in\mathbf{A}^*[v=w\cdot\delta\wedge\exists s'[s\overset{w}{\to}s'\wedge\mathrm{act}(s')=\emptyset]]. \quad (2)$$

$$v=\langle a_n\rangle_{n\in\omega}\in\mathbf{A}^\omega\wedge\exists\langle s_n\rangle_{n\in\omega}[s_0=s \quad (3)$$

* この言語を選んだのは，6.において技術的には簡単でかつトリビアルでない完全抽象モデルを紹介するためである．

** 変数の束縛について，μXはλ記法$(\lambda X.\cdots)$のλXと同様に働く．この点は関数型言語PCFにおけるμ記法と同様である[15]．$(\mu X.S)$が再帰プログラムを表現するとは，$(\mu X.S)$の動作/意味が，その本体Sの中のXを$(\mu X.S)$で置き換えたものの動作/意味により定義されるという意味（以下の遷移規則(7)と(8)を参照）．

* 本稿では'(意味) モデル'という語により構文領域（$\mathcal{L}$あるいは$\mathcal{L}_\theta$）から何らかの集合への写像を表す．モデルの定義は操作的にされることもあるし，表示的にされることもある．

$\wedge \forall n \in \omega[s_n \overset{a_n}{\rightarrow} s_{n+1}]]$. ■

プログラムから得られる情報のうち何に注目するかはその使い方・目的等に応じて変わるが，$\mathcal{O}$は実行可能な動作の順序と終り方のみに注目する場合にプログラムから観測されるもの（この場合の**プログラムの効果**）を表現したものとなっている．

次の例に示すように，$\mathcal{O}$においては，複合的なプログラムの意味をその部分の意味から合成することはできない（具体的には**4**.の式(7)で$\mathcal{D}$を$\mathcal{O}$で置き換えたものが成り立たない）．この意味で，$\mathcal{O}$は合成的でない（合成性の一般的な定義は**4**.で与える）．

例1　$s_1 \equiv a\cdot\mathbf{e} + a\cdot a\cdot\mathbf{e}$,　$s_2 \equiv a\cdot(\mathbf{e} + a\cdot\mathbf{e})$と置く．$\mathcal{O}$の定義から，$\mathcal{O}[\![s_i]\!] = \{\langle a,\surd\rangle, \langle a,a,\surd\rangle\}(i=1,2)$．しかし$\mathcal{O}[\![s_1 \| \mathbf{O}]\!] \neq \mathcal{O}[\![s_2 \| \mathbf{O}]\!]$である．なぜならば$\langle a,\delta\rangle \in \mathcal{O}[\![s_1 \| \mathbf{O}]\!] \setminus \mathcal{O}[\![s_2 \| \mathbf{O}]\!]$. ■

例2　k個の端末をサポートしている計算システムsを考える．i番目の端末$(i=1,\cdots,k)$に対応しているプロセスs_iは開始動作a_b^iを受けた後，要求動作a_1^i，…，a_n^iを受けて各々の応答動作$\tilde{a}_1^i$，…，$\tilde{a}_n^i$を実行することを繰り返すか，あるいは終了動作a_e^iを受けて最初に戻るものとすると，

$$s_i \equiv (\mu X\cdot a_b^i\cdot(\mu X'\cdot a_1^i\cdot\tilde{a}_1^i\cdot X' + \cdots + a_n^i\cdot\tilde{a}_n^i\cdot X' + a_e^i\cdot X))$$

と表される．システム全体は$s_i(i=1,\cdots,k)$の並行合成として，$s \equiv (s_1 \| \cdots \| s_k)$と表される． ■

3. 意味領域

1.で述べたように，表示的モデルの構成においては，個々のプログラムが表す数学的対象の集合である**意味領域**（semantic domain）を定義することが出発点となる．**2**.で述べたように，再帰プログラム$(\mu X.S)$は直観的には“$X=S$となるプロセスX”を表す（一般にSはXを含む）．これを数学的に表現して，表示的意味論においては，再帰プログラム$(\mu X.S)$の意味は，その本体Sの解釈として定まる意味領域上の関数Fの不動点として定義され，この不動点は意味領域上の適当な出発点pとFから作られる繰り返し列

$$p, F(p), F^2(p), F^3(p), \cdots \qquad (4)$$

の極限として与えられる[14),15)]．ここで$F^n(p)$はpにFをn回適用した結果である．**6**.の例3では**2**.の言語における再帰プログラムに対する不動点としての意味付けの例を与えている．）

このような極限を得るためには基礎となる意味領域において適当な収束の概念が定義されている必要がある．プロセス代数の表示的意味論においてよく用いられる意味領域としては，**完備半順序**（complete partial order, 以後cpoと略す）と**完備距離空間**がある．このうちcpoの方は逐次型言語の表示的意味論においても標準的に用いられる構造であるが[15)]，完備距離空間の方は並行言語の意味論において特によく用いられる構造である[4),11),13)]．

cpoおよび完備距離空間において，式(4)の形の繰り返し列が収束することは，各々，次の定理1および定理2によって保証される．

定理1（Tarskiの不動点定理）cpoからそれ自身への連続関数Fは最小不動点を持つ．しかも，cpoの最小元を$\perp$とすると，$\perp$，$F(\perp)$，$F^2(\perp)$，…という列は上昇列であり，この上限がFの最小不動点となる．Fの最小不動点を$\mathbf{Y}(F)$で表す． ■

定理2（Banachの不動点定理）完備距離空間からそれ自身への縮小写像Fは唯一の不動点を持つ．しかも，空間内の任意の点p_0に対して，p_0, $F(p_0)$, $F^2(p_0)$,…という列は収束し，この極限がFの唯一の不動点となる．Fの唯一の不動点を$\mathrm{fix}(F)$で表す． ■

対象とする言語や目的に応じて種々のcpoや完備距離空間が用いられている．簡単な関数型言語に対しては基礎となるデータ領域から最小元の付加や関数空間をとる操作により求める意味領域が得られる[15)]．一方，圏論的手法を用いて領域方程式を解き，その解を意味領域とする場合もある[7)]．

ここでは，**6**.の準備として，**2**.で導入した列の集合$A \triangleright B$を基礎に完備距離空間とcpoを定義しておく．まず$A \triangleright B$上の距離dを次のように定義する．各$v_1, v_2 \in A \triangleright B$に対して

$$d(v_1, v_2) = \inf\{(1/2)^n \mid v_1 \restriction n = v_2 \restriction n\}. \qquad (5)$$

ここで$v \restriction n$は列$v \in A \triangleright B$を長さ$n \in \omega$で打ち切ったものを表す．$(A \triangleright B, d)$は完備距離空間となる．$\mathcal{P}_c(A \triangleright B)$を$(A \triangleright B, d)$の閉集合の全体とし，$\mathcal{P}_c(A \triangleright B)$上の距離$\tilde{d}$を$d$と同様に打ち切りを用いて定義すると，$(\mathcal{P}_c(A \triangleright B), \tilde{d})$も完備

距離空間となる*．また$(\mathcal{P}_c(A\triangleright B),\supseteq)$は$A\triangleright B$を最小元とするcpoとなる．

4. 意味モデルに求められる基本性質

一般にプログラミング言語のモデルに対しては**合成性**（compositionality），すなわち意味が等しい2つのプログラムs_1，s_2についてどんな**文脈**（context）においても全体の意味を変えることなくs_1とs_2が置き換え可能であるという性質が重要である（ここでいう文脈とは代入場所を含む記号列で，そこにプログラムを代入することにより全体が1つのプログラムとなるようなもののことである）．一般に代数的言語$\mathcal{L}$においては，たかだか1つの自由変数を持つ項として文脈を定義できる**．以下では変数Xを固定して，$\mathcal{L}_X$をX以外の自由変数を持たない$\mathcal{L}$の要素の集合とし，$\mathcal{L}_X$の要素を$\mathcal{L}$の**文脈**と呼ぶ．また$s\in\mathcal{L}_\theta$に対して$S[s]$は$S[s/X]$を表すものとする．これによりモデル$\mathcal{D}$が**合成的**であるとは，任意のプログラムs_1，s_2に対して

$$\mathcal{D}[\![s_1]\!]=\mathcal{D}[\![s_2]\!]\Rightarrow\forall S\in\mathcal{L}_X[\mathcal{D}[\![S[s_1]]\!]=\mathcal{D}[\![S[s_2]]\!]] \tag{6}$$

が成り立つことと定義できる．モデル$\mathcal{D}$が合成的であれば，複合的なプログラムの意味がその構成要素の意味から合成できることになる．すなわち，各演算記号F（r変数）に対して次の関係が成り立つ．

$$\wedge_{i=1}^{r}(\mathcal{D}[\![s_i]\!]=\mathcal{D}[\![s_i']\!])\Rightarrow\mathcal{D}[\![F(s_1,\cdots,s_r)]\!]=\mathcal{D}[\![F(s_1',\cdots,s_r')]\!]. \tag{7}$$

（$r=1$の場合は，式(6)のSとして$F(X)$をとればよい．一般の場合も同様．）$\mathcal{D}$の合成性（式(6)）は次の式(8)で定義される$\mathcal{L}_\theta$上の同値関係$\sim_{\mathcal{D}}$が合同関係であることと等価であり，モデル$\mathcal{D}$をプログラムの解析・検証の基礎として用いる際には最低限求められる性質である．

$$s_1\sim_{\mathcal{D}}s_2\Leftrightarrow\mathcal{D}[\![s_1]\!]=\mathcal{D}[\![s_2]\!]. \tag{8}$$

不動点演算子μを含むような言語に対しては，再帰プログラム$(\mu X.S)$の意味が本体Sの解釈を表す関数Fの不動点であること，すなわち

$$\mathcal{D}[\![(\mu X.S)]\!]=F(\mathcal{D}[\![(\mu X.S)]\!]) \tag{9}$$

となるのが自然である（**3.**の最初を参照）．これは$(\mu X.S)$を展開して$S[(\mu X.S)/X]$とするプログラム変換の正当性を保証するためにも必要となる．

例1で見たように，操作的モデルは必ずしも合成的であるとは限らない．一方**6.**で見るように表示的モデルはその定義の仕方から必ず合成的となり，また条件(9)を満たす．

プログラムの効果を表すモデル$\mathcal{O}$が何らかの方法で与えられているとする（**2.**参照）．モデル$\mathcal{D}$を用いて意味のあるプログラム解析・検証を行うためには，$\mathcal{D}$において等しい意味を持つ2つのプログラムがすべての文脈で同じ効果を持つこと，すなわち，任意のプログラムs_1，s_2に対して

$$\mathcal{D}[\![s_1]\!]=\mathcal{D}[\![s_2]\!]\Rightarrow\forall S\in\mathcal{L}_X[\mathcal{O}[\![S[s_1]]\!]=\mathcal{O}[\![S[s_2]]\!]] \tag{10}$$

となることが必要となる．この性質を$\mathcal{D}$の$\mathcal{O}$に対する**適切性**（adequacy）と呼ぶ．なお，この逆の性質，すなわち任意のプログラムs_1，s_2に対して

$$\mathcal{D}[\![s_1]\!]\neq\mathcal{D}[\![s_2]\!]\Rightarrow\exists S\in\mathcal{L}_X[\mathcal{O}[\![S[s_1]]\!]\neq\mathcal{O}[\![S[s_2]]\!]] \tag{11}$$

が成り立つという性質を$\mathcal{D}$の$\mathcal{O}$に対する**完全性**（completeness）と言う．

5. 意味モデルの望ましい性質—完全抽象性

言語$\mathcal{L}$とプログラムの効果を表すモデル$\mathcal{O}$が何らかの方法で与えられているものとする．モデル$\mathcal{D}$が$\mathcal{O}$に対して**完全抽象的**（fully abstract）であるとは，$\mathcal{D}$が$\mathcal{O}$に対して適切でかつ完全であること，すなわち，任意のプログラムs_1，s_2に対して

$$\mathcal{D}[\![s_1]\!]=\mathcal{D}[\![s_2]\!]\Leftrightarrow\forall S\in\mathcal{L}_X[\mathcal{O}[\![S[s_1]]\!]=\mathcal{O}[\![S[s_2]]\!]] \tag{12}$$

が成り立つことと定義される*．$\mathcal{D}$が$\mathcal{O}$に対して完全抽象的であれば，$\mathcal{D}$は合成的でかつ$\mathcal{O}$に対して適切である**．さらに$\mathcal{D}$はこのようなモデルの中で最も抽象度の高いモデルとなる．（すなわち，式(10)を満たす任意の合成的モデル$\mathcal{D}'$に対して

---

* 集合$X\in\mathcal{P}_c(A\triangleright B)$の$n$での打ち切りを$\{v\restriction n|v\in X\}$と定義する．

** 本章，**5.**，および**6.1**では$\mathcal{L}$は不特定の代数的言語を一般的に表すものとする．記号の簡略化を計って，**2.**と**6.2**では，具体的に定義した特定の言語を指すためにやはり$\mathcal{L}$を用いるが，これら2つの用法を区別されたい．モデルの名前$\mathcal{O}$などについても同様．

* $\mathcal{O}$が合成的であれば式(12)は$\mathcal{D}$の導く同値関係と$\mathcal{O}$の導くそれが等しいことを表すことに注意されたい．

** $\mathcal{D}$の合成性（式(6)）は，ある文脈にもう一つの文脈を代入した結果も1つの文脈であることを用いて，式(12)の⇐部分から導かれる．

$\mathcal{D}'[\![s_1]\!]=\mathcal{D}'[\![s_2]\!]$ならば$\mathcal{D}[\![s_1]\!]=\mathcal{D}[\![s_2]\!]$となる．このような$\mathcal{D}'$は一般に多数あることに注意されたい．）$\mathcal{O}$が合成的でないときには，$\mathcal{D}$は，合成的となるために必要な最小限の情報を$\mathcal{O}$に付加して得られた最適なモデルであると言うことができる（$\mathcal{O}$が合成的であれば，$\mathcal{D}$と$\mathcal{O}$はプログラム間に同じ同値関係を導くという意味で等価である）．

完全抽象的な表示的モデルを与えることは，適切性のみが保証されたモデルを与えることに比較して格段に困難であることが多い．プロセス代数の完全抽象表示的モデルを構成する問題については，文献8)が先駆的な結果であり，その後80年代後半以降になっていくつかの結果が得られている．メッセージ伝達に基づく非命令型並行言語に対しては次のような結果が知られている．まずCCSと類似の一群の言語に対しては，表示的失敗集合モデル(failure set model)がある種の線形操作的意味論に対して完全抽象的であることが示されている[2),13)]．失敗集合モデルは元来CSPのモデルとして提案されたものである[10),*]．また，これらの言語を含む一群の言語に対して，操作的に定義されるMUSTテスト同値は合同関係であることが示されている[5)]．したがってMUSTテスト同値に対する完全抽象表示的モデルを構成する問題はMUSTテスト同値と等価な同値関係を導く表示的モデルを構成する問題に帰着され，受理木（acceptance tree）モデルがこの性質を持つことが示されている[5),7)]．これらの言語に対しては受理木モデルと失敗集合モデルが同型であることも知られている[5)]．また共有変数に基づく命令型並行言語に対しては，ある種の線形操作的意味論に対する完全抽象表示的モデルは，基本的に大域状態の対の列の集合を表示的意味として構成できることが示されている[3),8),11)]．

6. 完全抽象性証明の枠組とその2.のプロセス代数への適用

6.1 完全抽象性証明の枠組

前章と同様，言語$\mathcal{L}$とプログラムの効果を表すモデル$\mathcal{O}$が何らかの方法で与えられているとする．適当な意味領域を基礎に表示的モデル$\mathcal{D}$を定義したとき，その$\mathcal{O}$に対する完全抽象性は次のような方針で示すことができる．

（1） $\mathcal{L}_0$上で$\mathcal{D}$と一致するような，すなわち

$$\forall s\in\mathcal{L}_0[\mathcal{D}[\![s]\!]=\mathcal{C}[\![s]\!]] \tag{13}$$

となるような操作的モデル$\mathcal{C}$を定義する．このようなモデルを**中間モデル**と呼ぶ．

（2） 任意のプログラムs_1，s_2に対して，

$$\mathcal{C}[\![s_1]\!]=\mathcal{C}[\![s_2]\!]\Rightarrow\mathcal{O}[\![s_1]\!]=\mathcal{O}[\![s_2]\!] \tag{14}$$

が成り立つことを示す．

（3） $\mathcal{C}$の$\mathcal{O}$に対する**完全性**（式(11)で$\mathcal{D}$を$\mathcal{C}$に置き換えたもの）を示す．

（4） （1）-（3）項から$\mathcal{D}$の$\mathcal{O}$に対する完全抽象性（式(12)）が導かれる．ここで，式(12)の$\Rightarrow$部分は（1），（2）項および$\mathcal{D}$の合成性から，$\Leftarrow$部分は（1），（3）項から導かれる．

6.2 上記枠組の2.のプロセス代数への適用

2.の言語$\mathcal{L}$に対して表示的モデル$\mathcal{D}$を定義し，上で述べた方針に沿って，$\mathcal{D}$が2.の操作的モデル$\mathcal{O}$に対して完全抽象的であることを示す*．技術的な理由により，2.の言語$\mathcal{L}$の要素Sを，guardedness条件と呼ばれる次の条件(15)を満たすものに制限する**．

"$(\mu X.S')$がSの部分式ならば，S'中のXのすべての自由な出現は$(a\cdot\ldots)$という形のS'の部分式の中にある．" (15)

モデル$\mathcal{D}$は拒否集合として$\mathcal{P}(\{\surd\})$の要素（すなわち$\{\surd\}$と$\emptyset$）のみを持つ，非常に単純化された失敗集合モデルとして定義される（失敗集合モデルについては左段下，脚注*を参照）．$\mathbf{F}=\mathbf{A}\triangleright\{\surd,\{\surd\},\emptyset\}$と置く．$\Gamma$を$\mathcal{P}(\{\surd\})$を動く変数とする．$\mathbf{F}$の要素のうち，$w\cdot\Gamma$という形のものは動作列$w$を実行した後$\mathbf{A}\cup\Gamma$のどの要素の実行も不可能である状態に達する可能性を表す（式(19)参照）*3．3.で述べたように$\mathcal{P}_c(\mathbf{F})$上に距離$\tilde{d}$を導入し$(\mathcal{P}_c(\mathbf{F}),\tilde{d})$を完全距離空間とすることができる．技術的な理由により，$\mathcal{P}_c(\mathbf{F})$の部分空間$\mathcal{P}_c^*(\mathbf{F})$を$\mathcal{P}_c(\mathbf{F})$の要素$p\neq\emptyset$で次の条件(16)を満

---

* プロセスsが動作列wを実行した後，動作集合Xのどの要素も実行不能となる可能性があるとき，対(w,X)をsの失敗（failure）と呼ぶ．プロセスの意味をその失敗の集合を基礎に（適当な付加情報を加えて）定義するような意味モデルを失敗集合モデルと総称する．(w,X)がsの失敗であるとき，Xを（wを実行後の）sの拒否集合（refusal set）と呼ぶ．

* このモデルは文献13)や11)に現れるモデルを2.の設定に合わせて単純化したものである．

** この条件と文脈自由文法のGreibach正規系を規定する条件の類似に注意されたい．

*3 列$w\cdot\Gamma$は文献10)等における対(w,Γ)に対応する．対ではなく列を用いるのは意味領域を距離空間として扱う上での便宜のため．

たすものの集合として定義し，$\mathcal{D}$の意味領域として用いる．以下，$p\in\mathcal{P}_c^*(\mathbf{F})$，$w\in\mathbf{A}^*$に対し，$p[w]=\{v\in\mathbf{F}|w\cdot v\in p\}$と置く．

$$\forall w\in\mathbf{A}^*[p[w]\neq\emptyset\wedge\forall a\in\mathbf{A}[p[w\cdot a]=\emptyset]\Rightarrow w\cdot\emptyset\in p]. \quad (16)$$

$\mathcal{P}_c^*(\mathbf{F})$が$\mathcal{P}_c(\mathbf{F})$の閉部分集合であることから$(\mathcal{P}_c^*(\mathbf{F}),\tilde{d})$も完備距離空間となることが分かる．

動作前置$a\cdot$の解釈は$\mathcal{P}_c^*(\mathbf{F})$の要素に$a$を前置する演算であり，これは$\mathcal{P}_c^*(\mathbf{F})$上の縮小写像である．並行合成の解釈$\tilde{\|}:(\mathcal{P}_c^*(\mathbf{F}))^2\to\mathcal{P}_c^*(\mathbf{F})$は次のように再帰的に定義される*．

$$\tilde{\|}(p_1,p_2)=(\{\surd\}\cap p_1\cap p_2)\cup\{\Gamma\in p_1\cup p_2|\Gamma\setminus\{\surd\}\in p_1\cap p_2\}\cup\tilde{\|}_{\ell}(p_1,p_2)\cup\tilde{\|}_{\ell}(p_2,p_1). \quad (17)$$

ここで

$$\tilde{\|}_{\ell}(p_1,p_2)=\cup\{a\cdot\tilde{\|}(p_1[a],p_2)|p_1[a]\neq\emptyset\}.$$

定数記号**O**，**e**の解釈$\tilde{\mathbf{O}}$，$\tilde{\mathbf{e}}\in\mathcal{P}_c^*(\mathbf{F})$，および選択合成の解釈$\tilde{+}:(\mathcal{P}_c^*(\mathbf{F}))^2\to\mathcal{P}_c^*(\mathbf{F})$も簡単に与えることができる．

上で定義した$F\in\Sigma$の$\mathcal{P}_c^*(\mathbf{F})$上での解釈$\tilde{F}$を用いて$\mathcal{D}$は次のように定義される．

定義2 変数の集合から意味領域$\mathcal{P}_c^*(\mathbf{F})$への関数を**付値**と言う．$S\in\mathcal{L}$と付値$\rho$に対して，$\mathcal{D}[\![S]\!](\rho)\in\mathcal{P}_c^*(\mathbf{F})$を次のように$S$の構造についての帰納法で定義する．

(1) Sが変数のとき，$\mathcal{D}[\![S]\!](\rho)=\rho(S)$.

(2) $\mathcal{D}[\![F(S_1,\cdots,S_r)]\!](\rho)=\tilde{F}(\mathcal{D}[\![S_1]\!](\rho),\cdots,\mathcal{D}[\![S_r]\!](\rho))$.

(3) $\mathcal{D}[\![(\mu X.S)]\!](\rho)=\mathrm{fix}(\lambda p.\mathcal{D}[\![S]\!](\rho[p/X]))$. (18)

ここで，$(\lambda p.\mathcal{D}[\![S]\!](\rho[p/X]))$が縮小写像であることは前置演算が縮小写像で$\tilde{+}$，$\tilde{\|}$が非拡張写像であること，および$(\mu X.S)$が条件(15)を満たすことから保証される．■

この定義から$s\in\mathcal{L}_{\mathbf{0}}$に対しては，$\mathcal{D}[\![s]\!](\rho)$の値は$\rho$によらないことが分かる．この値を$\mathcal{D}[\![s]\!]$と書く．定義2から$\mathcal{D}$の合成性（式(6)）が導かれる．証明は文脈$S$の構造についての帰納法による．

例3 定義2から，再帰プログラム$(\mu X.a\cdot X)$の表示的意味$\mathcal{D}[\![(\mu X.a\cdot X)]\!]$は，$p\in\mathcal{P}_c^*(\mathbf{F})$に対し$a\cdot p$を対応させる関数の唯一の不動点となる．これは1つの無限列からなる集合$\{\langle a,a,\cdots\rangle\}$である．■

中間モデル$\mathcal{C}:\mathcal{L}_{\mathbf{0}}\to\mathcal{P}_c^*(\mathbf{F})$は**2.**の遷移系を用いて次のように定義される．

定義3 各$s\in\mathcal{L}_{\mathbf{0}}$と$v\in\mathbf{F}$に対して，$v\in\mathcal{C}[\![s]\!]$となるのは先の式(1)，式(3)，あるいは次の式(19)のどれかが成り立つときとする．

$$\exists w\in\mathbf{A}^*,\exists\Gamma\in\mathcal{P}(\{\surd\})[v=w\cdot\Gamma\wedge\exists s'[s\overset{w}{\to}s'\wedge\mathrm{act}(s')\cap(\mathbf{A}\cup\Gamma)=\emptyset]].\blacksquare \quad (19)$$

この定義から$\mathcal{C}$は次の意味で準同型写像であることが示される．すなわち，各$F\in\Sigma$(r変数)に対し，

$$\mathcal{C}[\![F(s_1,\cdots,s_r)]\!]=\tilde{F}(\mathcal{C}[\![s_1]\!],\cdots,\mathcal{C}[\![s_r]\!]).$$

この性質を用いて，$s\in\mathcal{L}_{\mathbf{0}}$の構造についての帰納法により$\mathcal{D}$と$\mathcal{C}$の$\mathcal{L}_{\mathbf{0}}$上での一致（式(13)）が示される．$s\equiv(\mu X.S)$に対しては，$\mathcal{D}[\![s]\!]$と$\mathcal{C}[\![s]\!]$が共に縮小写像$(\lambda p.\mathcal{D}[\![S]\!](\rho[p/X]))$の不動点であることから簡単にこの一致が示される．これが意味領域として完備距離空間を用いることの利点の1つである．

$\mathcal{C}$と$\mathcal{O}$を関係づける関数$\mathcal{A}:\mathcal{P}_c^*(\mathbf{F})\to\mathcal{P}(\mathbf{E})$を次のように定義する．各$p\in\mathcal{P}_c^*(\mathbf{E})$に対し，

$$\mathcal{A}(p)=\{w\cdot\delta|w\cdot\{\surd\}\in p\}\cup(p\cap\mathbf{E}).$$

モデル$\mathcal{O}$，$\mathcal{C}$の定義から，$\mathcal{O}=\mathcal{A}\circ\mathcal{C}$．したがって，式(14)が導かれる．

$\mathcal{C}$の$\mathcal{O}$に対する完全性（式(11)）を示すには，$\mathcal{C}[\![s_1]\!]\neq\mathcal{C}[\![s_2]\!]$と仮定し，ある文脈$S$について$\mathcal{O}[\![S[s_1]]\!]\neq\mathcal{O}[\![S[s_2]]\!]$となることを示せばよい．$\mathcal{O}[\![s_1]\!]=\mathcal{O}[\![s_2]\!]$の場合が本質的であり，$S\equiv X\|\mathbf{O}$と置けばよいことが分かる（例1参照）．

以上により，**6.1**に述べた方針に従って$\mathcal{D}$の$\mathcal{O}$に対する完全抽象性（式(12)）が導かれる．

定義2では完備距離空間を基礎に$\mathcal{D}$を定義したが，cpo $(\mathcal{P}_c^*(\mathbf{F}),\supseteq)$を基礎に$\mathcal{D}$を定義することもできる*．そのためには式(18)の右辺を$\mathbf{Y}(\lambda p.\mathcal{D}[\![S]\!](\rho[p/X]))$で置き換えればよい．ただしこの定義の下で式(13)を示すのは定義2を用いた場合ほど簡単ではない．

上ではプログラムの構文定義とは独立に意味領域を定義し，それを用いて完全抽象モデルを構成する方法を紹介したが，公理的意味論を介してあ

* 厳密には，$\tilde{\|}$は式(17)の表す高階縮小写像の不動点として定義される．

* 厳密には，$(\mathcal{P}_c^*(\mathbf{F}),\supseteq)$がcpoとなるためには，$\mathbf{A}$は有限と仮定するか，あるいは条件(16)を強める必要がある．

る種のターム・モデルとして完全抽象モデルを構成しようとするアプローチもある[5]．

7. むすび

表示的アプローチを中心にプロセス代数の意味論について論じ，特に完全抽象モデルを構成する問題について紹介した．

表示的意味論を中心としたプロセス代数意味論の今後の課題としては，より記述力が豊富な言語へのモデルの拡張が第一にあげられる．たとえば，(実)時間的特性や確率論的特性に対する記述力の付加，高次プロセスに対する記述力の付加，π 計算等への拡張が課題であり，活発に研究が進められている[6),9),16)]．

プログラム言語の意味モデルはプログラムの検証・変換等を厳密に議論する上で不可欠である．プログラミングをする上では意味モデルの理解，特に，良い表示的意味モデルの理解はプログラムについての種々の見通しを得る上で有用である(これは数値計算において，実数直線のイメージと数学的性質の把握が種々の見通しを得る上で有用であることに例えられる)．また新たに言語を設計する際には意味モデルの構成を同時に行うことにより，プログラムの意味を考える上で見通しのよい言語を得ることが期待できる．このような並行言語の例としてはABCL，POOLなどがあげられる．

謝辞 ATR通信システム研究所の田倉昭氏，およびNTTコミュニケーション科学研究所の真野健氏には草稿を読んでいただき貴重な助言をいただいた．

参考文献

1) Baeten, J. C. M. and Weijland, W. P.: Process Algebra, Cambridge University Press (1990).
2) Bergstra, J. A., Klop, J. W. and Olderog, E.-R.: Readies and Failures in the Algebra of Communicating Processes, SIAM J. of Computing, Vol. 17, No. 6, pp. 1134 1177 (1988).
3) Brookes, S. D.: Full Abstraction for a Shared Variable Parallel Language, in Proceedings of Eighth Annual IEEE Symposium on Logic in Computer Science, pp. 98-109 (1993).
4) de Bakker, J. W. and Zucker, J. I.: Processes and the Denotational Semantics of Concurrency, Information and Control, Vol. 54, pp. 70-120 (1982).
5) Hennessy, M.: Algebraic Theory of Processes, MIT Press (1988).
6) Hennessy, M.: Higher-Order Processes and Their Models, in Lecture Notes in Computer Science, Vol. 820, pp. 286-303 (1994).
7) Hennessy, M. and Ingólfsdóttir, A.: A Theory of Communicating Processes with Value Passing, Information and Computation, Vol. 107, pp. 202-236 (1993).
8) Hennessy, M. and Plotkin, G.: Full Abstraction for a Simple Parallel Programming Language, in Lecture Notes in Computer Science, Vol. 74, pp. 108-120, Springer (1979).
9) Hennessy, M. and Regan, T.: A Process Algebra for Timed Systems, Information and Computation, Vol. 117, pp. 221-239 (1995).
10) Hoare, C. A. R.: Communicating Sequential Processes, Prentice Hall International (1985).
11) Horita, E., de Bakker, J. W. and Rutten, J. J. M. M.: Fully Abstract Denotational Models for Nonuniform Concurrent Languages, Information and Computation, Vol. 115, pp. 125-178 (1994).
12) Milner, R.: Communication and Concurrency, Prentice Hall International (1989).
13) Rutten, J. J. M. M.: Correctness and Full Abstraction of Metric Semantics for Concurrency, in Lecture Notes in Computer Science, Vol. 354, pp. 628-658, Springer (1989).
14) 立木秀樹：プログラム理論と不動点，情報処理，Vol.33，No.4，pp.366-374 (1992)．
15) 横内寛文：プログラム意味論，共立出版 (1994)．
16) Yuen, S., Cleaveland, R., Dayar Z. and Smolka, S. A.: Fully Abstract Characterization of Testing Preorders for Probabilistic Processes, in Lecture Notes in Computer Science, Vol. 836, pp. 497-512.

(平成7年5月9日受付)

堀田 英一（正会員）

1958年生．1982年京都大学理学部（数学）卒業．1984年九州大学大学院工学研究科修士課程（応用数学）修了．同年日本電信電話公社（現NTT）入社．1989～90年，オランダ国立数学計算機科学研究所（CWI）客員研究員．Ph.D.（アムステルダム自由大学）．現在，NTTコミュニケーション科学研究所主任研究員．通信プロトコルの形式記述と検証，並行プログラミング言語の意味論と処理系の研究に従事．情報規格調査会FDT-SWG小委員会委員．電子情報通信学会，日本ソフトウェア科学会，EATCS各会員．