プロセス代数によるリアルタイムオブジェクト指向プログラミング言語の意味論

佐藤一郎† 所 真理雄†

本論文ではプロセス代数によるリアルタイムオブジェクト指向プログラミング言語の意味定義の方法を示す．この方法は時間的特性の表現能力を拡張したプロセス代数体系をもとにするもので，プログラミング言語の構文要素ごとにそのプロセス代数の式に変換する規則を与え，その規則を通じて言語の意味を定義していくものである．この意味定義の特徴は，このプロセス代数の表現性によりタイムアウト処理や遅延実行などの時間的特性，さらに並行実行，オブジェクト間通信，オブジェクトの動的生成などの並行オブジェクト指向計算の特徴が表現できることにある．さらに，この意味定義ではプログラムの実行時間と同時に実行される並列動作数を任意に与えることができることから，例えば特定個の並列プロセッサにより実行されるリアルタイムプログラムの動作内容とそれの実行時間が解析できるようになる．論文ではこの意味定義によるプログラムの解析技法についても例示する．

Process Algebra Semantics for a Real Time Object Oriented Programming Language

Ichiro Satoh† and Mario Tokoro†

This paper presents a framework to define a semantics for a real-time object-oriented programming language and to verify programs written in the language. The semantics is defined as the collection of translation rules that map the syntactic constructions of the language into expressions in a process calculus extended with the notion of time. By using the expressive capabilities of the calculus, the semantics can appropriately capture temporal features of the language such as timeout exception and execution time, and concurrent object-oriented features such as concurrency, class, and object creation. Since it can restrict the number of concurrent activities which take place simultaneously, it allows to analyze and predict temporal and behavioral properties of programs executed even with a number of processors smaller than the number of active objects. Along with an example, we illustrate a method of analyzing real-time programs.

1. はじめに

近年，リアルタイムシステムのためのプログラミング言語が並行オブジェクト指向計算[18]にもとづいて提案されている[5),15)]．これは複数の能動的実行実体の相互作用にもとづいてシステムをモデル化していく並行オブジェクト指向計算の考え方が，センサやアクチュエータなどの多くの能動的機器からなるリアルタイムシステムの構成形態と親和性が高く，リアルタイムシステムの設計や実装に優れた方法を提供すると考えられているからである．しかし，その一方で並行オブジェクト指向計算がもつ特性の中にはリアルタイムシステムの正当性を保証するうえで障害となるものがある．リアルタイムシステムの正当性の判定では，その時間的特性を事前に解析・予測することが重要となるが，並行オブジェクト指向計算がその特徴としてもつ能動的オブジェクトの動的な生成や，通信相手が動的に変化するような通信などは，プログラムの実行時間やプログラム中の並行動作数に影響を与え，その結果，プログラム全体の時間的特性の予測を困難にしてしまう．また，こうした特性はプログラム中の変数の内容や if 文や while 文などの制御フローなどのプログラムの動作内容と密接に関係している．このため，並行オブジェクト指向計算にもとづくリアルタイムプログラムの時間的正当性を考える場合には，時間性だけでなくその動作内容についても解析する必要が生じる．

リアルタイムプログラムの解析方法として，リアルタイムプログラミング言語の意味論を定式化し，これ

† 慶應義塾大学理工学研究科計算機科学専攻
Department of Computer Scinece, Faculty of Science and Technology, Keio Unviversity

を通じて解析する方法がいくつか提案されている．これらの方法ではプログラムの実行に沿ってその動作内容と時間性を詳細に解析することができるという利点があるが，対象となる言語がオブジェクト指向言語ではなく，プロセスの動的生成やプロセス間の動的通信が扱えないなどの問題がある．

そこで本論文では，上記のような動的な特性をもつリアルタイムプログラミング言語のための意味定義の方法を提案する．この方法では，数量的時間の記述能力をもつプロセス代数体系を意味定義のメタ言語として利用する．ここでプロセス代数体系を用いるのは，プロセス代数体系の並行や通信に関する優れた表現性を利用するためと，それの検証技法を利用するためである．また，並列計算機による実行ではプログラムの実行時間は利用可能な並列プロセッサ数に依存することから，われわれの意味定義では同時に実行される並列動作数を任意に与えられるようにする．

ここで本論文の構成を述べる．つづく第2章ではプログラミング言語$\mathcal{R}_{\mathcal{G}}$を提案する．これはリアルタイムプログラミング言語がもつ時間記述性と，並行オブジェクト指向言語の特徴をもつ言語で，これを通じてわれわれの意味定義方法を説明していく．第3章では意味定義のメタ言語であるプロセス代数体系を示す．ただし，通常のプロセス代数体系[7]は数量的時間の表現性をもっていないことから，時間性が取り扱えるプロセス代数体系RtCCS[11]を用いる．第4章では，$\mathcal{R}_{\mathcal{G}}$の各構文要素ごとにその動作内容的・時間的意味をRtCCSの式に変換する規則を与え，その規則全体として$\mathcal{R}_{\mathcal{G}}$の意味を定義する．第5章では関連研究を，第6章では結論と将来の研究課題を述べる．付録に$\mathcal{R}_{\mathcal{G}}$のプログラム例を通じて本論文で示した意味定義にもとづいた解析方法を簡単に紹介する．

2. $\mathcal{R}_{\mathcal{G}}$：リアルタイムオブジェクト指向言語

本章では$\mathcal{R}_{\mathcal{G}}$の構文と非形式的意味を与える．

$\mathcal{R}_{\mathcal{G}}$の構文

$\mathcal{R}_{\mathcal{G}}$の構文は**図1**により与えられる．ただしここでは次のようなメタ記号を用いる．*Prog*はプログラム，*Dec*は変数宣言，*Class*はクラス定義，*Method*はメソッド定義，*Stat*はプログラム文，*Exp*は式*，*Bool*はブール式を表し，Cはクラス名，Mはメソッド名，Xはプログラム変数名とする．

* ただし，**wait** *Exp* と …**timeout** *Exp* の *Exp* は自然数とする．

Prog	::=	**program** *Class* **is** *Dec* **in** *Stat* **endp**
Class	::=	**class** C **is** *Dec* **new** *Stat Method* **endc**
	|	*Class Class*
Method	::=	**method** $M(X)$ **is** *Dec* **in** *Stat* **endm**
	|	*Method Method*
Dec	::=	**var** X
	|	*Dec Dec*
Stat	::=	X := *Exp*
	|	**return** *Exp*
	|	**wait** *Exp*
	|	**if** *Bool* **then** *Stat* **else** *Stat* **endif**
	|	**while** *Bool* **do** *Stat* **od**
	|	*Stat* ; *Stat*
Exp	::=	0|1|2|…
	|	X
	|	**error**
	|	C::**new**
	|	X::M(*Exp*) **timeout** *Exp*
Bool	::=	*Exp* == *Exp*
	|	**true**
	|	**false**

図1 $\mathcal{R}_{\mathcal{G}}$の構文
Fig. 1 Syntax of $\mathcal{R}_{\mathcal{G}}$.

$\mathcal{R}_{\mathcal{G}}$の非形式的意味

ここでは$\mathcal{R}_{\mathcal{G}}$の非形式的意味を簡単に述べる．$\mathcal{R}_{\mathcal{G}}$の各オブジェクトはある一つのクラスから動的に生成されたインスタンスで，各インスタンス内部は逐次的に実行される．また，$\mathcal{R}_{\mathcal{G}}$の通信は手続き呼び出しとしてメソッドを呼び出すことにより実現され，インスタンス間の通信に遅れはないものとする．

プログラム： プログラムは，クラス定義，初期化プログラム，グローバル変数宣言からなる．ここで，初期化プログラムとはプログラムの開始時に実行されるプログラム文，グローバル変数とは初期化プログラムとすべてのインスタンスからアクセス可能な変数である．

program *Class* **is** *Dec* **in** *Stat* **endp**

クラス定義： クラスはオブジェクト（インスタンス）のテンプレートであり，同じクラスのインスタンスは，同一構造の内部状態とメソッドをもつ．

class C **is** *Dec* **new** $Stat_{new}$
  method $M_1(X_1)$ **is** Dec_1 **in** $Stat_1$ **endm**
  ⋮ ⋮
  method $M_n(X_n)$ **is** Dec_n **in** $Stat_n$ **endm**
endc

ここで，Cをクラス名，*Dec*をインスタンスの内部変数の宣言，$Stat_{new}$をインスタンス生成時に実行されるプログラム文，M_iをメソッド名とする．そして，X_i, Dec_i, $Stat_i$は，それぞれメソッドM_iの呼び出し引数，内部変数の宣言，プログラム文である．

オブジェクトの動的生成式： 以下の式により，クラ

スCのインスタンスを生成し，そのインスタンスの識別子を式の値として返す．

C:: **new**

タイムアウト機能つきメソッドコール式： オブジェクト間の相互作用は以下のメソッドの呼び出し式により行う．呼び出し側は呼び出されたメソッドからの返答値が戻るまでブロックされる．呼び出されたメソッドからの返答値が式の値となるが，所定の時間内に返答値が得られない場合は **error** がその値となる．

X:: M (Exp_1) **timeout** Exp_2

ここで X は呼び出されるインスタンスの識別子を格納している変数名，M はそのインスタンスの呼び出されるメソッド名である．また，Exp_1 の値がメソッド呼び出しの引数，Exp_2 の値がタイムアウトになるまでのデッドライン時間となる．

遅延実行文： 以下の文は式 Exp の値分の単位時間だけこれに続く文の実行開始を遅らせる．

wait Exp

非同期返答文： 呼び出されたメソッドは以下の文を実行することにより，呼び出し側に Exp の値をその返答値として返す．また，この返答文に続く文があるときは返答後も実行を続ける．このとき呼び出されたメソッドと呼び出し側が並行実行されることになる．

return Exp

3. RtCCS：意味定義のためのメタ言語

ここではわれわれの意味定義のメタ言語であるプロセス代数 RtCCS*を定義する．これは CCS[7]に**時間経過**と**時間経過に依存した動作**の表現能力を拡張した体系である**．

RtCCS では，時間経過はアクション：$\surd$ を新たに導入することにより表現される．これはすべてのプロセスが同時に受け取る一斉同期アクションで，1回の実行が1単位時間の時間経過を表す．時間経過に依存する動作は，タイムアウト処理の意味をもつ二項演算子 $\langle\ ,\ \rangle_t$ を導入することにより表現される．例えば $\langle P, Q\rangle_t$ では t 単位時間内に $\surd$ 以外のアクションが実行可能な場合は P として振る舞い，実行不可能な場合はタイムアウトとして Q として振る舞う．

3.1 RtCCS の構文

RtCCS の構文は，新たに導入した $\surd$ と $\langle\ ,\ \rangle_t$ 以外は CCS と同様である．まず，RtCCS の定義に用いるアクション名およびその集合の記法を示す．

アクションには $\surd$ アクション，入力・出力アクション，内部アクションがあり，入力アクション名の集合を $\mathcal{A}$ とし，その要素を $a, b, \cdots$ と記述する．出力アクション名の集合を $\bar{\mathcal{A}}$ とし，その要素を $\bar{a}, \bar{b}, \cdots$ と記述する．ただし $\bar{\bar{a}} \equiv a$ である．また，内部アクションを τ と記述する．また，通信アクション名の集合を $\mathcal{L} \equiv \mathcal{A} \cup \bar{\mathcal{A}}$ とし，$l, l', \cdots$ を $\mathcal{L}$ 上の要素とする．さらに通信アクション名と内部アクションの集合を $Act \equiv \mathcal{L} \cup \{\tau\}$ とし，$\alpha, \beta, \cdots$ を Act 上の要素とする．

以下のような構文規則を持つ式の集合を RtCCS の動作式集合 ε とし，その要素を $E, E_1, E_2, \cdots$ と記す．

$$E ::= \mathbf{0} \mid X \mid \alpha.E \mid E_1 + E_2 \mid E_1 | E_2 \mid E[f] \mid E\backslash L \mid \mathbf{rec}X : E \mid \langle E_1, E_2\rangle_t$$

上記において，f をアクション名の変更関数 $f: Act \to Act$，ただし，任意の f において $\overline{f(l)} = f(\bar{l})$，$f(\tau) = \tau$ とする，L は $\mathcal{L}$ の部分集合，**rec** $X: E$ の E 中に X がある場合はその X はガードされているとする*．今後，$X \triangleq E$ と略記することがある．さらに，自由変数を含まない動作式の集合を $\mathcal{P}$ とし（$\mathcal{P} \subseteq \mathcal{E}$），その要素を $P, Q, \cdots$ と記述する**．$\langle E_1, E_2\rangle_t$ の t は自然数とする．

ここで，RtCCS の演算子の意味を非形式的に述べておく．$\mathbf{0}$ は終了したプロセスを，$\alpha.E$ はアクション α を実行後，動作式 E として振る舞うことを，$E_1 + E_2$ は E_1 と E_2 のどちらかを実行することを，$E_1 | E_2$ は E_1 と E_2 が並列実行されることを表す．$E[f]$ は E 中のアクション名を関数 f で変更する，$E\backslash L$ は E に含まれるアクションのうち集合 $L \subset \bar{L}$ に含まれるアクションの観測を制限する，**rec** $X: E$ は E 中に含まれる変数 X に E が束縛されることを表す．

RtCCS の意味論

RtCCS はラベルつき遷移システム $\langle \mathcal{E}, Act \cup \{\surd\}, \{\xrightarrow{\mu} | \mu \in Act \cup \{\surd\}\}\rangle$ として与えられる．ここで $\xrightarrow{\mu}$ は2種類の規則によって定義される遷移関係（$\xrightarrow{\mu} \subseteq \mathcal{E} \times \mathcal{E}$）である．一つは Act 中のアクションによりラベル付けされた遷移関係 $\xrightarrow{\alpha}$ に関するもので，**図2**の推論規則を満足する最小の関係として与えられる．もう一つは $\surd$ アクションによってラベル付けされた遷移関係 $\xrightarrow{\surd}$ に関するもので，**図3**の推論規則を満足する最小の関係として与えられる．

---

* Real-time Calculus of Communicating Systems

** RtCCS の詳細は別稿[11]に譲る．

* $a.X$ の $a \in Act$ をガードという．ガードされてないとは **rec** $X: X$, **rec** $X: (X+E)$ などの式をいう．

** 以下，動作式を単にプロセスとして呼ぶことがある．

$$\frac{}{\alpha.E \xrightarrow{\alpha} E} \qquad \frac{E_1 \xrightarrow{\alpha} E_1'}{E_1 + E_2 \xrightarrow{\alpha} E_1'}$$

$$\frac{E_2 \xrightarrow{\alpha} E_2'}{E_1 + E_2 \xrightarrow{\alpha} E_2'} \qquad \frac{E_1 \xrightarrow{\alpha} E_1'}{E_1|E_2 \xrightarrow{\alpha} E_1'|E_2}$$

$$\frac{E_2 \xrightarrow{\alpha} E_2'}{E_1|E_2 \xrightarrow{\alpha} E_1|E_2'} \qquad \frac{E_1 \xrightarrow{a} E_1',\ E_2 \xrightarrow{\overline{a}} E_2'}{E_1|E_2 \xrightarrow{\tau} E_1'|E_2'}$$

$$\frac{E \xrightarrow{\alpha} E',\ \alpha \notin L \cup \overline{L}}{E \setminus L \xrightarrow{\alpha} E' \setminus L} \qquad \frac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]}$$

$$\frac{E\{\mathbf{rec}\,X : E/X\} \xrightarrow{\alpha} E'}{\mathbf{rec}\,X : E \xrightarrow{\alpha} E'} \qquad \frac{E_1 \xrightarrow{\alpha} E_1',\ t > 0}{\langle E_1, E_2\rangle_t \xrightarrow{\alpha} E_1'}$$

$$\frac{E_2 \xrightarrow{\alpha} E_2'}{\langle E_1, E_2\rangle_0 \xrightarrow{\alpha} E_2'}$$

図 2 $\xrightarrow{\alpha}$ に関する遷移規則
Fig. 2 Transition rules for $\xrightarrow{\alpha}$.

$$\frac{}{\ell.E \xrightarrow{\surd} \ell.E} \qquad \frac{}{\mathbf{0} \xrightarrow{\surd} \mathbf{0}}$$

$$\frac{E_1 \xrightarrow{\surd} E_1',\ E_2 \xrightarrow{\surd} E_2'}{E_1 + E_2 \xrightarrow{\surd} E_1' + E_2'} \qquad \frac{E_1 \xrightarrow{\surd} E_1',\ E_2 \xrightarrow{\surd} E_2',\ E_1|E_2 \not\xrightarrow{\tau}}{E_1|E_2 \xrightarrow{\surd} E_1'|E_2'}$$

$$\frac{E \xrightarrow{\surd} E'}{E \setminus L \xrightarrow{\surd} E' \setminus L} \qquad \frac{E \xrightarrow{\surd} E'}{E[f] \xrightarrow{\surd} E'[f]}$$

$$\frac{E\{\mathbf{rec}\,X : E/X\} \xrightarrow{\surd} E'}{\mathbf{rec}\,X : E \xrightarrow{\surd} E'} \qquad \frac{E_1 \xrightarrow{\surd} E_1',\ t > 0}{\langle E_1, E_2\rangle_t \xrightarrow{\surd} \langle E_1', E_2\rangle_{t-1}}$$

$$\frac{E_2 \xrightarrow{\surd} E_2'}{\langle E_1, E_2\rangle_0 \xrightarrow{\surd} E_2'}$$

図 3 $\xrightarrow{\surd}$ に関する遷移規則
Fig. 3 Transition rules for $\xrightarrow{\surd}$.

$$\begin{aligned}
\mathcal{V}(\mathbf{0}) &\rightharpoonup \mathbf{0}\\
\mathcal{V}(X) &\rightharpoonup X\\
\mathcal{V}(a(x).E) &\rightharpoonup \textstyle\sum_{d\in D} a_d.\,\mathcal{V}(E[d/x])\\
\mathcal{V}(\overline{a}(d).E) &\rightharpoonup \overline{a_d}.\mathcal{V}(E)\\
\mathcal{V}(\tau.E) &\rightharpoonup \tau.\mathcal{V}(E)\\
\mathcal{V}(E_1 + E_2) &\rightharpoonup \mathcal{V}(E_1) + \mathcal{V}(E_2)\\
\mathcal{V}(E_1|E_2) &\rightharpoonup \mathcal{V}(E_1)|\mathcal{V}(E_2)\\
\mathcal{V}(E[f]) &\rightharpoonup \mathcal{V}(E)[f'] \text{ where } f'(\ell_d) = f(\ell)_d\\
\mathcal{V}(E \setminus L) &\rightharpoonup \mathcal{V}(E) \setminus \{\ell_d \mid l \in L, d \in D\}\\
\mathcal{V}(\mathbf{rec}\,X : E) &\rightharpoonup \mathbf{rec}\,X : \mathcal{V}(E)\\
\mathcal{V}(\langle E_1, E_2\rangle_t) &\rightharpoonup \langle \mathcal{V}(E_1), \mathcal{V}(E_2)\rangle_t\\
\mathcal{V}(\text{if } \mathit{true} \text{ then } E_1 \text{ else } E_2) &\rightharpoonup \mathcal{V}(E_1)\\
\mathcal{V}(\text{if } \mathit{false} \text{ then } E_1 \text{ else } E_2) &\rightharpoonup \mathcal{V}(E_2)
\end{aligned}$$

図 4 値引き渡しつき動作式から $\mathcal{E}$ 式への変換
Fig. 4 Translation of expressions with value into $\mathcal{E}$ expressions.

次章で示す意味定義では，RtCCSのプロセス間で値の受け渡しが必要となることがある．そこで，RtCCSの動作式に値渡し機構を拡張する．ただし値渡しつき動作式は**図 4** に示される構文的な書き換え規則 $\mathcal{V}()$ により $\mathcal{E}$ 上の式に変換される．この規則の定義では受け渡される値の集合を D，その要素を d，値を受け取る変数を x とする．ここで $E[d/x]$ とは動作式 E の自由変数 x を d で置き換えることを示す．また，以下では動作式間で時間値の受け渡しが必要となることがある．よって D は時間値の集合，つまり自然数の集合を含むとし，演算子 $\langle\ ,\ \rangle_t$ の t を変数として扱うことがある．このとき $\mathcal{V}(\langle E_1, E_2\rangle_t[d/t]) \rightharpoonup \langle \mathcal{V}(E_1[d/t]), \mathcal{V}(E_2[d/t])\rangle_d$ とする．

4. プロセス代数に基づく意味定義

ここでは，$\mathcal{RT}$ の各構文要素を RtCCS の動作式に変換する規則を定式化する．$\mathcal{RT}$ の意味はこの変換規則の集まりとして定義される．われわれが用いる変換による意味定義の方法は，Milner による並列命令型プログラミング言語 $\mathcal{M}$ の意味定義の方法[7]をもとにしている．しかし，われわれの意味定義は次のような点で違いがある：(1)対象となる言語がリアルタイムオブジェクト指向プログラミング言語であること，(2)動作内容だけでなく実行時間や時間経過に依存した動作の意味が取り扱えること，(3)動作実体の動的生成や通信相手の動的変更などが扱えること，(4)並列実行数を任意に制限できること．

コンビネータ

リアルタイムプログラムの時間的正当性を調べるためには，プログラムの実行に沿ってその実行時間を詳細に調べることが必要となる．そこで，われわれの意味定義では各構文要素ごとの実行時間を導入し，プログラムの実行時間を解析できるようにする．ただし本論文では簡単化のため，文に相当する構文要素である *Dec* と *Stat* の実行時間のみを導入し，それを t_{stat} 単位時間とする*．また，プログラム全体の実行時間は実行環境であるコンピュータの並列プロセッサ数に依存することから，同時に実行される並列動作数の上限を任意に設定できるようにする．

これらの表現性は以下に示すコンビネータを通じて実現される．このコンビネータは $\mathcal{RT}$ の構文要素の変換により得られた RtCCS の動作式同士を結ぶもので，**図 5** のような "*Into*(x)" と "*Before*" の二つがある．

$$\begin{aligned}
P\ \mathit{Into}(x)\ Q &\stackrel{\text{def}}{=} (P|\mathsf{res}(x).Q) \setminus \{\mathsf{res}\}\\
P\ \mathit{Before}\ Q &\stackrel{\text{def}}{=} \mathsf{run}.(P[\mathsf{b}/\mathsf{done}]|\mathsf{b}.\langle \mathbf{0}, \overline{\mathsf{idle}}.Q\rangle_{t_{stat}}) \setminus \{\mathsf{b}\}
\end{aligned}$$

where
$$\begin{aligned}
\mathit{Processor} &\stackrel{\text{def}}{=} \overline{\mathsf{run}}.\mathsf{idle}.\mathit{Processor}\\
\mathit{Machine}_N &\stackrel{\text{def}}{=} \underbrace{\mathit{Processor}|\cdots|\mathit{Processor}}_{N}
\end{aligned}$$

図 5 コンビネータの定義
Fig. 5 Definition of combinators.

* 簡単のために *Dec* と *Stat* のみを考慮したが，より詳細な実行時間モデルも容易に取り扱うことができる．

Into(x) **コンビネータ**は左側の構文要素（*Bool* と *Exp*）を表す動作式から結果をアクション **res** を介して受け取り，その結果を右側の動作式中の変数 x にバインドする．以下に簡単な展開式を示す．P は *Bool* または *Exp* を表す動作式とする．

$$P\ \mathit{Into}(x)Q = ((\cdots.\ \overline{\mathbf{res}}(d).\mathbf{0}) \mid \mathbf{res}(x).Q) \setminus \{\mathbf{res}\}$$
$$\xrightarrow{\tau(\mathbf{res})} (Q[d/x]) \setminus \{\mathbf{res}\}$$

where $P \stackrel{\text{def}}{=} \cdots.\ \overline{\mathbf{res}}(d).\mathbf{0}$

Before **コンビネータ**は次のような特性を表現するために用いられる：（1）逐次実行，（2）構文要素の実行時間，（3）並列動作数の制限．また，*Before* では括弧を次のように省略する："*P Before Q Before R*" とは "*P Before* (*Q Before R*)" である．

（1）変換されたプログラム文（*Stat* と *Dec*）は終了の際，終了通知としてアクション $\overline{\mathbf{done}}$ を送信する．このコンビネータでは左側の動作式からこのアクションを受け取ったあと，右側の動作式を実行可能にすることによりプログラム文の逐次実行を表す．

（2）このコンビネータでは RtCCS の $\langle\ ,\ \rangle_t$ により左側の動作式のアクション **done** の受信後，一つのプログラム文の実行時間に相当する時間，つまり t_{stat} 単位時間だけ右側の動作式の実行を遅らせる．この遅延時間が左側の動作式の実行コストに相当する．

（3）*Before* コンビネータは，左側の動作式を実行可能にする前にプロセッサを表すプロセス *Processor* からアクション **run** を受信する必要がある．また *Processor* は一度 $\overline{\mathbf{run}}$ を送信すると，アクション **idle** を受信するまでは $\overline{\mathbf{run}}$ は送信しない．これより左側の動作式が実行可能になる *Before* コンビネータの数は *Processor* の個数と等しくなり，$\mathit{Machine}_N$ はプロセッサ数が N 個の並列計算機に相当するものになる．

ここで，例として**図 6** のような二つのプロセッサによって実行される三つの並行プログラム文（P_i *Before* Q_i，$i=1,2,3$）を考える．P_1，P_2，P_3 はプログラム文を表す動作式でそれぞれ終了時に $\overline{\mathbf{done}}$ を実行するとする．このとき図 6 のように並行実行される三つのプログラム文のうち二つのだけが実行可能となり，残りの一つは少なくても二つのうちのどちらかが終了するまでは実行できないことになり，並列実行される文の数は *Processor* の個数と等しくなることがわかる．

$$\begin{aligned}
&\mathit{Machine}_2 \mid (P_1\ \mathit{Before}\ Q_1) \mid \\
&\quad (P_2\ \mathit{Before}\ Q_2) \mid (P_3\ \mathit{Before}\ Q_3) \\
&\xrightarrow{\tau(\mathbf{run})} \mathbf{idle}.\mathit{Processor} \mid \overline{\mathbf{run}}.\mathbf{idle}.\mathit{Processor} \mid \\
&\qquad ((P_1[\mathrm{b/done}] \mid \mathrm{b}.\langle \mathbf{0}, \overline{\mathbf{idle}}.Q_1\rangle_{t_{stat}}) \setminus \{\mathrm{b}\}) \mid \\
&\qquad (P_2\ \mathit{Before}\ Q_2) \mid (P_3\ \mathit{Before}\ Q_3) \\
&\xrightarrow{\tau(\mathbf{run})} \mathbf{idle}.\mathit{Processor} \mid \mathbf{idle}.\mathit{Processor} \mid \\
&\qquad ((P_1[\mathrm{b/done}] \mid \mathrm{b}.\langle \mathbf{0}, \overline{\mathbf{idle}}.Q_1\rangle_{t_{stat}}) \setminus \{\mathrm{b}\}) \mid \\
&\qquad\quad ((P_2[\mathrm{b/done}] \mid \mathrm{b}.\langle \mathbf{0}, \overline{\mathbf{idle}}.Q_2\rangle_{t_{stat}}) \setminus \{\mathrm{b}\}) \mid \\
&\qquad\qquad (P_3\ \mathit{Before}\ Q_3) \\
&\longrightarrow \quad \cdots
\end{aligned}$$

図 6 *Before* の展開例
Fig. 6 Derivation example of *Before* combinator.

以下ではコンビネータに関していくつかの注記点を述べる．

- 図 2 と図 3 より RtCCS では実行可能なプロセス間通信（*Act* 中のアクション）は√アクションより優先する*．よって $\overline{\mathbf{run}}$ を送信可能な *Processor* が存在している限り，**run** を待っている *Before* コンビネータは不要な時間経過をすることなく **run** による通信を行い，左側動作式を実行可能にする．また，*Before* コンビネータは，$\overline{\mathbf{run}}$ を送信可能などの *Processor* とも通信が可能で，特定の *Processor* とのみ通信する必要はない．このことはプロセッサの切り替えに相当する．
- *Before* コンビネータは入れ子状に使用されたとき，一番最後の動作式の実行時間が表現できないが**，われわれの意味定義では一番最後の動作式は実行コストが無視できる意味的要素に対応づけられることから，このことがプログラムの実行時間の解析において問題となることはない．
- 本論文では簡単化のために，プログラム文 *Dec* と *Stat* のみに実行時間を導入し，さらにその時間を t_{stat} 単位時間と仮定している．しかし，他の構文要素に対して実行時間を導入することや，t_{stat} 以外の実行時間を適宜に与えることは容易である．これはわれわれの意味定義の枠組が多様な実行環境に適用できることを示し，例えば t_{stat} を変えた場合に，プログラム全体の動作内容や実行時間がどのように変化するかを事前に解析・予想できるようになる．

---

* つまり，√アクションは *Act* 中のアクションが実行不可能なときだけ実行される．

** 例えば R_1, R_2, R_3 を *Before* を含まない動作式とするとき，$\mathit{Machine}_2 \mid (\ldots \mathit{Before}\ R_1) \mid (\ldots \mathit{Before}\ R_2) \mid (\ldots \mathit{Before}\ R_3)$ では R_1, R_2, R_3 は同時に実行可能となることがある．

$$
\begin{array}{rcll}
[\![\textbf{var } X]\!] & = & Loc_X|\overline{\mathsf{done}}.\mathbf{0} & \text{(V1)}\\
Loc_X & \stackrel{\text{def}}{=} & \mathsf{write}_X(x).Loc'_X(x) & \\
Loc'_X(x) & \stackrel{\text{def}}{=} & \mathsf{write}_X(y).Loc'_X(y) + \overline{\mathsf{read}_X}(x).Loc'_X(x) & \\
[\![Dec_1\ Dec_2]\!] & = & [\![Dec_1]\!]\ Before\ [\![Dec_2]\!] & \text{(V2)}\\
[\![n]\!] & = & \overline{\mathsf{res}}(\hat{n}).\mathbf{0} \quad n \in Int & \text{(E1)}\\
[\![X]\!] & = & \mathsf{read}_X(x).\overline{\mathsf{res}}(x).\mathbf{0} & \text{(E2)}\\
[\![\textbf{error}]\!] & = & \overline{\mathsf{res}}(error).\mathbf{0} & \text{(E3)}\\
[\![C::\textbf{new}]\!] & = & \overline{\mathsf{new}_C}.\mathsf{id}_C(i).\overline{\mathsf{res}}(i).\mathbf{0} & \text{(E4)}\\
[\![X::M(Exp_1)\ \textbf{timeout}\ Exp_2]\!] & = & [\![Exp_2]\!]\ Into(t)([\![Exp_1]\!]\ Into(x)(\mathsf{read}_X(i). & \\
 & & (\langle\overline{\mathsf{call}_{i,M}}(x).\mathsf{ret}_{i,M}(y).\overline{\mathsf{reply}}(y).\mathbf{0}\,,\ \overline{\mathsf{abort}}.\mathbf{0}\rangle_t| & \\
 & & \langle\mathsf{reply}(z).\overline{\mathsf{res}}(z).\mathbf{0}\,,\ \overline{\mathsf{res}}(error).(\mathsf{abort}.\mathbf{0}+\mathsf{reply}(z).\mathbf{0})\rangle_t) & \\
 & & \backslash\{\mathsf{reply},\mathsf{abort}\})) & \text{(E5)}\\
[\![\textbf{true}]\!] & = & \overline{\mathsf{res}}(true).\mathbf{0} & \text{(B1)}\\
[\![\textbf{false}]\!] & = & \overline{\mathsf{res}}(false).\mathbf{0} & \text{(B2)}\\
[\![Exp_1 == Exp_2]\!] & = & [\![Exp_1]\!]\ Into(x_1)\ ([\![Exp_2]\!]\ Into(x_2)(\overline{\mathsf{res}}(equal(x_1,x_2)).\mathbf{0})) & \text{(B3)}\\
 & & \text{where} \quad equal(x_1,x_2) = \begin{cases} true & \text{if } x_1 = x_2 \\ false & \text{otherwise} \end{cases} & \\
[\![X := Exp]\!] & = & [\![Exp]\!]\ Into(x)\ \overline{\mathsf{write}_X}(x).\overline{\mathsf{done}}.\mathbf{0} & \text{(S1)}\\
[\![\textbf{return}\ Exp]\!] & = & [\![Exp]\!]\ Into(x)\ \overline{\mathsf{ans}}(x).\overline{\mathsf{done}}.\mathbf{0} & \text{(S2)}\\
[\![\textbf{wait}\ Exp]\!] & = & [\![Exp]\!]\ Into(t)\ \langle\mathbf{0},\overline{\mathsf{done}}.\mathbf{0}\rangle_t & \text{(S3)}\\
[\![\textbf{if}\ Bool\ \textbf{then}\ Stat_1\ \textbf{else}\ Stat_2\ \textbf{endif}]\!] & = & [\![Bool]\!]\ Into(x)\ \text{if } x \text{ then } [\![Stat_1]\!] \text{ else } [\![Stat_2]\!] & \text{(S4)}\\
[\![\textbf{while}\ Bool\ \textbf{do}\ Stat\ \textbf{od}]\!] & = & W & \text{(S5)}\\
\text{where} \quad W & \stackrel{\text{def}}{=} & [\![Bool]\!]\ Into(x)\ \text{if } x \text{ then } [\![Stat]\!]\ Before\ W \text{ else } \overline{\mathsf{done}}.\mathbf{0} & \\
[\![Stat_1\ ;\ Stat_2]\!] & = & [\![Stat_1]\!]\ Before\ [\![Stat_2]\!] & \text{(S6)}\\
[\![\textbf{class}\ C\ \textbf{is}\ Dec\ \textbf{new}\ Stat\ Method\ \textbf{endc}]\!] & = & ClassDec_C \qquad \text{where} & \text{(C1)}\\
ClassDec_C & \stackrel{\text{def}}{=} & \mathsf{new}_C.\mathsf{getid}(i).\overline{\mathsf{id}_C}(i). & \\
 & & (ClassDec_C|([Dec]\!]\ Before\ [\![Stat]\!]\ Before\ Body_{C,i}) \setminus L_{Dec}) & \\
Body_{C,i} & \stackrel{\text{def}}{=} & ([\![Method]\!](C,i)|\mathsf{endm}.Body_{C,i}) \setminus \{\mathsf{endm}\} & \\
[\![Class_1\ Class_2]\!] & = & [\![Class_1]\!]|[\![Class_2]\!] & \text{(C2)}\\
[\![\textbf{method}\ M(X)\ \textbf{is}\ Dec\ \textbf{in}\ Stat\ \textbf{endm}]\!](C,i) & = & \mathsf{call}_{i,M}(x).(\mathsf{ans}(y).\overline{\mathsf{ret}_{i,M}}(y).\mathbf{0}|Loc_X|\overline{\mathsf{write}_X}(x). & \text{(M1)}\\
 & & ([\![Dec]\!]\ Before\ [\![Stat]\!]\ Before\ \overline{\mathsf{endm}}.\mathbf{0}) \setminus L_{Dec}) \setminus L_X \setminus \{\mathsf{ans}\} & \text{(M1)}\\
[\![Method_1\ \cdots\ Method_n]\!](C,i) & = & [\![Method_1]\!](C,i) + \cdots + [\![Method_n]\!](C,i) & \text{(M2)}\\
[\![\textbf{program}\ Class\ \textbf{is}\ Dec\ \textbf{in}\ Stat\ \textbf{endp}]\!] & = & [\![Class]\!]|ObjId_0|Machine_N| & \\
 & & ([\![Dec]\!]\ Before\ [\![Stat]\!]\ Before\ \mathbf{0}) \setminus L_{Dec} & \text{(P1)}\\
\text{where} \quad ObjId_i & \stackrel{\text{def}}{=} & \overline{\mathsf{getid}}(i).ObjId_{i+1} &
\end{array}
$$

図 7 変換規則
Fig. 7 Translation rules from $\mathcal{RG}$ to RtCCS expressions.

変換規則

図 7 に $\mathcal{RG}$ の各構文要素を RtCCS の動作式に変換する規則 [ ] を与える．この変換規則ではオブジェクト識別子を $0, 1, \cdots$ とし，その集合を Id とする．また，整数を $\hat{0}, \hat{1}, \cdots, \hat{n}, \cdots$ とし，その集合を Int とする．以下では主要な変換規則について説明する．

変数宣言 (Dec)： $\mathcal{RG}$ の変数宣言は規則 (V1)-(V2) により RtCCS 式に変換される．変数 X の宣言の意味は，整数またはオブジェクト識別子を格納するための領域（プロセス Loc_X）を生成することである．変数への値の書き込みは，値をアクション $\mathbf{write}_X$ の引数として Loc_X(または Loc'_X) の自由変数にバインドすることにより，また読み込みは格納値をアクション $\overline{\mathbf{read}_X}$ の引数として送信することにより表現される．ここで，$L_X \stackrel{\text{def}}{=} \{\mathbf{read}_X, \mathbf{write}_X\}$ を Loc_X のアクセスソートと呼び，$L_{Dec} \stackrel{\text{def}}{=} \cup L_X$ (ただし X は Dec 中で宣言されている変数) とする．

プログラム式 (Exp, $Bool$)： プログラム式 Exp と $Bool$ の変換規則を (E1)-(E5) と (B1)-(B3) に示す．変換されたプログラム式はその式の値をアクション $\overline{\mathbf{res}}$ の引数として $Into$ コンビネータに送る．

ここではタイムアウト機能付きメソッドコール式の変換 (E5) について考える．これはまずタイムアウトのデッドライン時間 Exp_2 とメソッド呼び出しの引数 Exp_1 を評価する．次に Loc_X に格納されている識別子 i を入力アクション $\mathbf{read}_X$ の引数として得る．そして，出力アクション $\overline{\mathbf{call}_{i,M}}$ により識別子が i のインスタンスのメソッド M を呼び出す．

このメソッド呼び出しがタイムアウトとなる場合として，(1)所定時間内にメソッドの呼び出しが実行で

きない場合，(2)所定時間内にメソッドの呼び出しはできたが返答値が所定時間内に戻ってこない場合の二つがある．変換では二つの RtCCS のタイムアウト演算子によりそれぞれのタイムアウトを判定する．

(1) 一つ目のタイムアウト演算子により，メソッドの呼び出し ($\overline{\mathbf{call}_{i,M}}$) がデッドライン時間内に行えるか判別する．成功した場合は返答値をアクション $\mathbf{ret}_{i,M}$ で受け取り，アクション $\overline{\mathbf{reply}}$ を通じて二つ目のタイムアウト演算子に送る．失敗した場合は出力アクション $\overline{\mathbf{abort}}$ を送信する．

(2) 二つ目のタイムアウト演算子では，デッドライン時間内にアクション **reply** を通じて返答値を受け取れた場合，その返答値をアクション $\overline{\mathbf{res}}$ を介して $Into(x)$ コンビネータに送る．時間内に受け取れなかった場合は返答値の代わりに *error* を送り，そのあとに送られてくるアクション **reply** (または **abort**) を待つ．

プログラム文 (*Stat*)： *Stat* の構文要素の変換規則は (S1)-(S6) のようになる．(S1) では Exp の評価値をアクション $\overline{\mathbf{write}_X}$ を通じて Loc_X に格納し，X への代入を表す．(S3) では RtCCS の演算子 $\langle\ ,\ \rangle_t$ により Exp の値分だけ終了通知 ($\overline{\mathbf{done}}$) の送信を遅らせて，次のプログラム文の実行開始を遅延させる．

クラス定義 (*Class*)： クラス定義の変換規則は (C1)-(C2) に与えられる．クラス C の変換ではアクション $\mathbf{new}_C$ を受け取ると，プロセス *ObjId* (後述) からアクション **getid** を介して新しいインスタンス識別子を受け取る．そして，アクション $\overline{\mathbf{id}}_C$ を介して生成依頼側にその識別子を送り，そのインスタンス本体に相当するプロセス "$([Dec]Before[Stat]Before\ Body_{C,i})\backslash L_{Dec}$" を生成する．またインスタンス変数のカプセル化は，"$\backslash L_{Dec}$" により L_{Dec} 中変数のアクションソート内のアクションとインスタンス外部との通信を制限することに表現される．

メソッド定義 (*Method*)： メソッド定義 (*Method*) の変換規則を (M1)-(M2) に定義する．インスタンス i のメソッド M はアクション $\mathbf{call}_{i,M}$ により呼び出される．呼び出されたメソッドでは，メソッド呼び出しの引数，つまり x を変数領域 Loc_X に格納し，メソッド文 [*Stat*] の実行を開始する．その後，非同期返答文を表す動作式からアクション **ans** の引数として返答値を受け取り，アクション $\overline{\mathbf{ret}_{i,M}}$ により呼び出し側にその返答値を送る．

プログラム (*Prog*)： プログラム *Prog* の変換規則を (P1) に示す．ここでプロセス *ObjId* はアクション **getid** の受信ごとに一意の番号を生成するプロセスで，この番号がインスタンスの識別子となる．

5. 関連研究

ここでは関連研究について述べる．リアルタイムプログラムでは動作内容だけでなく実行時間などの時間性に関してもその正当性を調べる必要がある．こうしたリアルタイムシステムの解析方法として，ペトリネット，時相論理，プロセス代数などの形式的体系に数量的時間の表現能力を拡張したものがあるが，これらの記述性は抽象レベルが高く，変数や制御フローなどのプログラムの詳細を解析できないことがある．このため，プログラムを詳細に解析する方法としてリアルタイムプログラミング言語の意味論がいくつか提案されている[3),4),6),14)]．これらは表示意味論をもとにし，言語構文要素ごとにその動作内容とその時間的特性の組を要素とする領域に写像することにより，言語の動作内容と時間性に関する両方の意味を定義するものである．しかし，これらの多くは逐次型コンピュータによる実行や無限個のプロセッサの存在を仮定している[4),6)]，また有限個の共有プロセッサの表現が可能であっても[3),14)]，各プロセスは常に特定のプロセッサが割り当てられると仮定し，複数プロセッサの切り替えによる実行が表現できない．

また，われわれと同様にプロセス代数にもとづいて意味が定義されるリアルタイムプログラミング言語として CSR[2)]がある．数量的時間性をもつプロセス代数体系の式に変換することにより定義を行う点で $\mathcal{RT}$ と類似しているが，CSR では並列実行されるプロセス数の制限や増減が扱えない．また，CSR や上記の表示意味論によるものはプロセスの動的生成や通信相手の動的変更などの機能をもつプログラミング言語を取り扱うことができない．

次にプロセス代数を利用したオブジェクト指向プログラミング言語の意味論に関する関連研究について述べる．Papathomas[9)]は継承機構 (inheritance) をもつ並行オブジェクト指向言語を CCS の動作式に変換することによって意味を定義する方法を示している．Walker[17)]は並行オブジェクト指向言語 POOL[1)]を CCS に通信ポート名の受け渡し機構を拡張した体系である π 計算[8)]に変換することにより，また，戸村[16)]は委譲機構 (Delegation) をもつ並行オブジェクト指

向言語を π 計算と同様に通信ポート名の受け渡し機構を拡張した CCS に変換することにより意味を定義する方法を提案している．特に Papathomas[9] と Walker[17] による方法は，われわれの方法と同様に Milner[7] による命令型並列プログラミング言語 $\mathcal{M}$ の CCS にもとづく意味定義の方法をもとにしている．しかし，上記の研究[9),16),17)]では時間的特性や並列動作数は考慮されていない．

6. 結　論

本論文では，プロセス代数にもとづくリアルタイムオブジェクト指向プログラミング言語のための意味定義の方法を示した．これはプログラミング言語の構文要素ごとに時間的表現性を拡張したプロセス代数体系である RtCCS の動作式に変換する規則を定式化し，その変換規則を通してプログラミング言語の意味を定義するものである．そして，任意のプログラムはこの変換規則により RtCCS の動作式に変換され，RtCCS の理論的な枠組みを通して厳密に解析できるようになる．特に同時に実行される構文要素数を制限できることから特定数の並列プロセッサによって実行されるプログラムの動作内容と，実行時間などの時間性が解析できるという点で従来のリアルタイムプログラミング言語の意味論にはない特徴をもっている．

また，本論文で示した意味定義方法は $\mathcal{RC}$ だけに依存したものではなく，並行性や，通信相手が動的変化する通信，動的プロセス生成などの動的特性をもつ他のリアルタイムプログラミング言語にも適用できる汎用的な枠組みである．

最後に，今後の研究課題について述べる．本論文では簡単化のため，実行スケジューリングや優先度に関して最小限の仮定しか設けていない．しかし，スケジューリング方法や優先度はプログラムの実行時間や並列動作数に影響を与えることから今後の課題としたい．また，本論文の意味定義では一つの全域的時間を仮定しているが，分散システムなどでは局所的時間性が本質的となる．そこで局所的時間性を導入したプロセス代数体系[12),13)]による分散システムのためのリアルタイムプログラミング言語の意味定義に関しても検討したい．

参 考 文 献

1) America, P., de Bakker, J., Kok, J. and Rutten, J.: Operational Semantics of a Parallel Object-Oriented Language, *Proceedings of POPL '87*, pp. 194-208 (1987).
2) Gerber, R. and Lee, I.: A Layered Approach to Automating the Verification of Real-Time Systems, *IEEE Trans. Softw. Eng.*, Vol. 18, No. 9, pp. 768-784 (1992).
3) Hooman, J.: A Denotational Semantics for Shared Processors, *Proceeding of PARLE '91, LNCS 506*, pp. 184-201, Springer-Verlag (1991).
4) Huizing, C., Gerth, R. and deRover, W. P.: Full Abstraction of a Real-Time Denotational Semantics for an OCCAM-like Language, *Proceeding of POPL '87*, pp. 223-237 (1987).
5) Ishikawa, Y., Tokuda, H. and Mercer, C. W.: Object-Oriented Real-Time Language Design: Construction for Timing Constraints, *Proceeding of ECOOP/OOPSLA '90*, pp. 289-298 (1990).
6) Koymans, R., Shyamasundar, R. K., deRover, W. P., Gerth, R. and Arun-Kumer, S.: Compositional Semantics for Real-Time Distributed Computing, *Information and Computation*, Vol. 79, No. 3, pp. 210-256 (1988).
7) Milner, R.: *Communication and Concurrency*, Prentice Hall (1989).
8) Milner, R., Parrow, J. and Walker, D.: A Calculus of Mobile Processes Part 1 & 2, Technical Report ECS-LFCS-89-85 & 86, University of Edinburgh (1989).
9) Papathomas, M.: A Unifying Framework for Process Calculus Semantics of Concurrent Object-Based Languages and Features, *Proceeding of Workshop on Concurrent Object-Based Concurrent Computing, LNCS 612*, pp. 53-79, Springer-Verlag (1992).
10) Satoh, I. and Tokoro, M.: A Formalism for Real-Time Concurrent Object-Oriented Computing, *Proceedings of OOPSLA '92*, pp. 315-326 (1992).
11) 佐藤一郎，所真理雄：時間的特性を考慮した並列プロセスの形式的記述，情報処理学会論文誌，Vol. 34, No. 4, pp. 540-548 (1993).
12) Satoh, I. and Tokoro, M.: A Timed Calculus for Distributed Objects with Clocks, *Proceedings of ECOOP '93, LNCS 707*, pp. 326-345, Springer-Verlag (1993).
13) 佐藤一郎，所真理雄：分散計算のための局所時間性に基づく形式系，コンピュータソフトウェア，Vol. 11, No. 2, pp. 32-44 (1994).
14) Shade, E. and Narayana, K. T.: Real-Time Semantics for Shared Variable Concurrency, *Information and Computation*, Vol. 102, No. 1, pp. 56-82 (1993).
15) Takashio, K. and Tokoro, M.: DROL: An Object-Oriented Programming Language for

Distributed Real-time Systems, *Proceedings of OOPSLA '92*, pp. 276-294 (1992).
16) 戸村 哲, 石川 裕, 二木厚吉: プロセス代数モデルに基づく並行オブジェクト指向言語の意味定義, コンピュータソフトウェア, Vol. 7, No. 2, pp. 12-29 (1990).
17) Walker, D.: π-Calculus Semantics of Object-Oriented Programming Languages, *Proceedings of Theoretical Aspects of Computer Software '91*, *LNCS 526*, pp. 532-547, Springer-Verlag (1991).
18) Yonezawa, A. and Tokoro, M. (ed.): *Object-Oriented Concurrent Programming*, MIT Press (1987).

付 録

ここでは，図8に示された$\mathcal{R}_{\mathcal{L}}$の読み手書き手プログラムの例を通じて，変換・解析の方法を簡単に例示する．これはバッファクラス (**Buffer**) のインスタンス，書き手クラス (**Writer**) のインスタンス，読み手クラス (**Reader**) のインスタンスから構成される．書き手（読み手）インスタンスは，メソッド **start** の呼び出しにより，バッファクラスのインスタンスのメソッド **put**（または **get**）を呼び，バッファにデータを入れる（または取り出す）ものである*．

このプログラムを第4章で与えた変換規則により RtCCS の動作式に変換すると**図9**のようになる．ただし，Nはプログラムの最大並列動作数で並列プロセッサの数に相当するものになる．また簡単化のため，動作式Pがt単位時間だけ遅延することを"$(t).P \triangleq \langle \mathbf{0}, P\rangle_t$"と略記する．

まず **Buffer** クラス定義の変換について考える．**Buffer** クラス定義は図10のような動作式に変換される．われわれはこの動作式を通じて **Buffer** クラス定義の動作内容と時間性が解析できる．例えばインスタンスを生成する部分に相当する変換部分"$\mathbf{new}_{Buffer}\cdots Body_{Buffer,i}$"では，二つの"**run**$\cdots(t_{stat})\cdots$**idle**"列を含んでいることから，この部分の実行時間が$2t_{stat}$単位時間であることがわかる．

ただし，変換された動作式は複雑となることがあることから，動作式を簡単化する方法として以下の等価関係**を導入する．

定義 二つのプロセス $P, Q \in \mathcal{P}$ 上の関係 $\mathcal{S}$ が時間的弱双模倣であるとは，$(P,Q)\in\mathcal{S}$ ならば任意の $\mu \in Act_{\mathcal{L}}$について次の二つの条件が成立することである．ただし，$P \overset{\hat{\mu}}{\Rightarrow} P'$は，$\mu \in \mathcal{L}\cup\{\surd\}$ のとき $P(\xrightarrow{\tau})^*\xrightarrow{\mu}(\xrightarrow{\tau})^* P'$ とし，μがτのとき $P(\xrightarrow{\tau})^* P'$ とする．

（i） $\forall P' : P \xrightarrow{\mu} P' \supset \exists Q' : Q \overset{\hat{\mu}}{\Rightarrow} Q' \wedge (P', Q') \in \mathcal{S}$.

（ii） $\forall Q' : Q \xrightarrow{\mu} Q' \supset \exists P' : P \overset{\hat{\mu}}{\Rightarrow} P' \wedge (P', Q') \in \mathcal{S}$.

時間的弱双模倣 $\mathcal{S}$ の最大の集合を時間的観測等価 $\approx_{\mathcal{L}}$ とする．

時間的観測等価となる二つの動作式は，互いの外部観測可能な動作内容とその実行タイミングを模倣し合うことになる．特にオブジェクト指向計算ではカプセル化によりインスタンスの内部的計算が外部に影響を

```
program
  class Writer is
      var Y
    method start(X) in
      return true ;
      Y := X::put(1) timeout 3 ;
        ......
    endm
  endc
  class Reader is
      var Y
    method start(X) in
      return true ;
      Y := X::get(0) timeout 3 ;
        ......
    endm
  endc
  class Buffer is
      var Y
    new
      Y := error
    method put(X) in
      Y := X ;
      return true
    endm
    method get(X) in
      return Y
    endm
  endc
is
  var B var R var W var X var Y ;
in
  B := Buffer::new ;
  W := Writer::new ;
  R := Reader::new ;
  X := W::start(B) timeout 5 ;
  Y := R::start(B) timeout 5
endp
```

図 8 読み手・書き手プログラム
Fig. 8 Reader/Writer program.

$$[\![\mathbf{program}\cdots\mathbf{is}\cdots\mathbf{endp}]\!] = Prog \text{ where}$$
$$Prog \overset{\text{def}}{=} ([\![\mathbf{class}\,\mathtt{Buffer}\cdots\mathbf{endc}]\!]|[\![\mathbf{class}\,\mathtt{Writer}\cdots\mathbf{endc}]\!]|$$
$$[\![\mathbf{class}\,\mathtt{Reader}\cdots\mathbf{endc}]\!]|([\![\mathbf{var}\,\mathtt{B}\cdots\mathbf{var}\,\mathtt{Y}]\!]\ Before$$
$$[\![\mathtt{B:=Buffer::new};\cdots\mathbf{timeout}\,5]\!])|Obj\mathit{Id}_0|Machine_N)$$

図 9 読み手・書き手プログラムの変換
Fig. 9 Translation of Reader/Writer program.

* $\mathcal{R}_{\mathcal{L}}$ の各インスタンスは高々1個の並列度（シングルスレッド実行）をもつことから，バッファのインスタンスにおいて **put**, **get** メソッドが同時に実行されないことに注意されたい．
** RtCCS の等価関係の詳細は別稿[10]にゆずる．

$$
\begin{aligned}
&[\![\textbf{class}\ \mathtt{Buffer}\ \textbf{var}\ Y\ \cdots \textbf{endc}]\!] = \mathit{ClassDec}_{\mathit{Buffer}} \quad \textit{where}\\
&\mathit{ClassDec}_{\mathit{Buffer}} \stackrel{\text{def}}{=} \mathsf{new}_{\mathit{Buffer}}.\mathsf{getid}(i).\overline{\mathsf{id}_{\mathit{Buffer}}}(i).(\mathit{ClassDec}_{\mathit{Buffer}}|\\
&\qquad \mathsf{run}.((Loc_Y|\overline{\mathsf{done}}.\mathbf{0})[\mathsf{b/done}]|\mathsf{b}.\overline{(t_{stat})}.\overline{\mathsf{idle}}.\\
&\qquad \mathsf{run}.((\overline{\mathsf{res}}(\mathit{error}).\mathbf{0}|\mathsf{res}(x).\overline{\mathsf{write}_Y}(x).\overline{\mathsf{done}}.\mathbf{0}) \setminus \{\mathsf{res}\}[\mathsf{b/done}]|\mathsf{b}.\overline{(t_{stat})}.\overline{\mathsf{idle}}.\\
&\qquad \mathit{Body}_{\mathit{Buffer},i}) \setminus \{\mathsf{b}\}) \setminus \{\mathsf{b}\} \setminus L_Y)\\
&\mathit{Body}_{\mathit{Buffer},i} \stackrel{\text{def}}{=} (\mathsf{call}_{i,\mathit{put}}(x).(\mathsf{ans}(y).\overline{\mathsf{ret}_{i,\mathit{put}}}(y).\mathbf{0}|\mathit{Loc}_X|\overline{\mathsf{write}_X}(x).(\\
&\qquad \mathsf{run}.((\mathsf{read}_X(z).\overline{\mathsf{res}}(z).\mathbf{0}|\mathsf{res}(u).\overline{\mathsf{write}_Y}(u).\overline{\mathsf{done}}.\mathbf{0}) \setminus \{\mathsf{res}\}[\mathsf{b/done}]|\mathsf{b}.\overline{(t_{stat})}.\overline{\mathsf{idle}}.\\
&\qquad \mathsf{run}.((\overline{\mathsf{res}}(\mathit{true}).\mathbf{0}|\mathsf{res}(w).\overline{\mathsf{ans}}(w).\overline{\mathsf{done}}.\mathbf{0}) \setminus \{\mathsf{res}\}[\mathsf{b/done}]|\mathsf{b}.\overline{(t_{stat})}.\overline{\mathsf{idle}}.\\
&\qquad \overline{\mathsf{endm}}.\mathbf{0}) \setminus \{\mathsf{b}\}) \setminus \{\mathsf{b}\})) \setminus L_X \setminus \{\mathsf{ans}\}\\
&\qquad + \mathsf{call}_{i,\mathit{get}}(x).(\mathsf{ans}(y).\overline{\mathsf{ret}_{i,\mathit{get}}}(y).\mathbf{0}|\mathit{Loc}_X|\overline{\mathsf{write}_X}(x).(\\
&\qquad \mathsf{run}.((\mathsf{read}_Y(z).\overline{\mathsf{res}}(z).\mathbf{0}|\mathsf{res}(u).\overline{\mathsf{ans}}(u).\overline{\mathsf{done}}.\mathbf{0}) \setminus \{\mathsf{res}\}[\mathsf{b/done}]|\mathsf{b}.\overline{(t_{stat})}.\overline{\mathsf{idle}}.\\
&\qquad \overline{\mathsf{endm}}.\mathbf{0}) \setminus \{\mathsf{b}\})) \setminus L_X \setminus \{\mathsf{ans}\}\,|\,\mathsf{endm}.\mathit{Body}_{\mathit{Buffer},i}) \setminus \{\mathsf{endm}\}
\end{aligned}
$$

図 10 Buffer クラスの変換
Fig. 10 Translation of **Buffer** class declaration.

$$
\begin{aligned}
&\mathit{ClassDec}_{\mathit{Buffer}} \approx_{\mathcal{T}} \mathit{ClassDec}'_{\mathit{Buffer}}\\
&\mathit{ClassDec}'_{\mathit{Buffer}} \stackrel{\text{def}}{=} \mathsf{new}_{\mathit{Buffer}}.\mathsf{getid}(i).\overline{\mathsf{id}_{\mathit{Buffer}}}(i).(\mathit{ClassDec}'_{\mathit{Buffer}}|\mathsf{run}.(\mathit{Loc}_Y|\overline{(t_{stat})}.\overline{\mathsf{idle}}.\\
&\qquad \mathsf{run}.\overline{\mathsf{write}_Y}(\mathit{error}).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathit{Body}'_{\mathit{Buffer},i}) \setminus L_Y)\\
&\mathit{Body}'_{\mathit{Buffer},i} \stackrel{\text{def}}{=} \mathsf{call}_{i,\mathit{put}}(x).\mathsf{run}.\overline{\mathsf{write}_Y}(x).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathsf{run}.\overline{\mathsf{ret}_{i,\mathit{put}}}(\mathit{true}).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathit{Body}'_{\mathit{Buffer},i}\\
&\qquad + \mathsf{call}_{i,\mathit{get}}(x).\mathsf{run}.\mathsf{read}_Y(y).\overline{\mathsf{ret}_{i,\mathit{put}}}(y).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathit{Body}'_{\mathit{Buffer},i}
\end{aligned}
$$

図 11 Buffer クラスの時間的観測等価式
Fig. 11 Equivalent expression of **Buffer** class declaration.

$$
\begin{aligned}
&[\![\textbf{class}\ \mathtt{Writer}\ \textbf{var}\ Y\ \cdots \textbf{endc}]\!] \approx_{\mathcal{T}} \mathit{ClassDec}'_{\mathit{Writer}} \quad \text{where}\\
&\mathit{ClassDec}'_{\mathit{Writer}} \stackrel{\text{def}}{=} \mathsf{new}_{\mathit{Writer}}.\mathsf{getid}(i).\overline{\mathsf{id}_{\mathit{Writer}}}(i).(\mathit{ClassDec}'_{\mathit{Writer}}|\\
&\qquad \mathsf{run}.(\mathit{Loc}_Y|\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathit{Body}'_{\mathit{Writer},i})) \setminus L_Y\\
&\mathit{Body}'_{\mathit{Writer},i} \stackrel{\text{def}}{=} \mathsf{call}_{i,\mathit{start}}(j).(\mathit{Loc}_X|\overline{\mathsf{write}_X}(j).\mathsf{run}.\overline{\mathsf{ret}_{i,\mathit{start}}}(\mathit{true}).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\\
&\qquad \mathsf{run}.(\langle\overline{\mathsf{call}_{j,\mathit{put}}}(\hat{1}).\mathsf{ret}_{j,\mathit{put}}(x).\overline{\mathsf{reply}}(x).\mathbf{0}\,,\ \overline{\mathsf{abort}}.\mathbf{0}\rangle_3\,|\\
&\qquad\quad \langle\mathsf{reply}(y).\overline{\mathsf{write}_Y}(y).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\cdots\cdots \mathit{Body}'_{\mathit{Writer},i}\,,\ \mathsf{reply}(y).\mathbf{0}+\mathsf{abort}.\mathbf{0}\,|\\
&\qquad\quad \overline{\mathsf{write}_Y}(\mathit{error}).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\cdots\cdots \mathit{Body}'_{\mathit{Writer},i}\rangle_3) \setminus \{\mathsf{reply},\mathsf{abort}\}) \setminus L_X
\end{aligned}
$$

$$
\begin{aligned}
&[\![\textbf{class}\ \mathtt{Reader}\ \textbf{var}\ Y\ \cdots \textbf{endc}]\!] \approx_{\mathcal{T}} \mathit{ClassDec}'_{\mathit{Reader}} \quad \text{where}\\
&\mathit{ClassDec}'_{\mathit{Reader}} \stackrel{\text{def}}{=} \mathsf{new}_{\mathit{Reader}}.\mathsf{getid}(i).\overline{\mathsf{id}_{\mathit{Reader}}}(i).(\mathit{ClassDec}'_{\mathit{Reader}}|\\
&\qquad \mathsf{run}.(\mathit{Loc}_Y|\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathit{Body}'_{\mathit{Reader},i})) \setminus L_Y\\
&\mathit{Body}'_{\mathit{Reader},i} \stackrel{\text{def}}{=} \mathsf{call}_{i,\mathit{start}}(j).(\mathit{Loc}_X|\overline{\mathsf{write}_X}(j).\mathsf{run}.\overline{\mathsf{ret}_{i,\mathit{start}}}(\mathit{true}).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\\
&\qquad \mathsf{run}.(\langle\overline{\mathsf{call}_{j,\mathit{get}}}(\hat{0}).\mathsf{ret}_{j,\mathit{get}}(x).\overline{\mathsf{reply}}(x).\mathbf{0}\,,\ \overline{\mathsf{abort}}.\mathbf{0}\rangle_3\,|\\
&\qquad\quad \langle\mathsf{reply}(y).\overline{\mathsf{write}_Y}(y).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\cdots\cdots \mathit{Body}'_{\mathit{Reader},i}\,,\ \mathsf{reply}(y).\mathbf{0}+\mathsf{abort}.\mathbf{0}\,|\\
&\qquad\quad \overline{\mathsf{write}_Y}(\mathit{error}).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\cdots\cdots \mathit{Body}'_{\mathit{Reader},i}\rangle_3) \setminus \{\mathsf{reply},\mathsf{abort}\}) \setminus L_X
\end{aligned}
$$

$$
\begin{aligned}
&[\![\textbf{var}\ \mathtt{B}\ \textbf{var}\ \mathtt{W}\ \textbf{var}\ \mathtt{R}\ \textbf{var}\ \mathtt{X}\ \textbf{var}\ \mathtt{Y}]\!]\ \mathit{Before}\ [\![\mathtt{B}{:=}\mathtt{Buffer}::\textbf{new};\cdots\ \mathtt{Y}{:=}\mathtt{R}::\textbf{start}(\mathtt{B})\ \textbf{timeout}\ 5]\!] \approx_{\mathcal{T}} \mathit{Initial}'\\
&\text{where}\quad \mathit{Initial}' \stackrel{\text{def}}{=} \mathsf{run}.\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathsf{run}.\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathsf{run}.\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathsf{run}.\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathsf{run}.\overline{(t_{stat})}.\overline{\mathsf{idle}}.\\
&\qquad \mathsf{run}.\overline{\mathsf{new}_{\mathit{Buffer}}}.\mathsf{id}_{\mathit{Buffer}}(i).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathsf{run}.\overline{\mathsf{new}_{\mathit{Writer}}}.\mathsf{id}_{\mathit{Writer}}(j).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\\
&\qquad \mathsf{run}.\overline{\mathsf{new}_{\mathit{Reader}}}.\mathsf{id}_{\mathit{Reader}}(k).\overline{(t_{stat})}.\overline{\mathsf{idle}}.A\\
&A \stackrel{\text{def}}{=} \mathsf{run}.(\langle\overline{\mathsf{call}_{j,\mathit{start}}}(i).\mathsf{ret}_{j,\mathit{start}}(x).\overline{\mathsf{reply}}(x).\mathbf{0}\,,\ \overline{\mathsf{abort}}.\mathbf{0}\rangle_5\,|\\
&\qquad \langle\mathsf{reply}(y).\overline{(t_{stat})}.\overline{\mathsf{idle}}.B\,,\ \mathsf{reply}(y).\mathbf{0}+\mathsf{abort}.\mathbf{0}\,|\,\overline{(t_{stat})}.\overline{\mathsf{idle}}.B\rangle_5) \setminus \{\mathsf{reply},\mathsf{abort}\}\\
&B \stackrel{\text{def}}{=} \mathsf{run}.(\langle\overline{\mathsf{call}_{k,\mathit{start}}}(i).\mathsf{ret}_{k,\mathit{start}}(x).\overline{\mathsf{reply}}(x).\mathbf{0}\,,\ \overline{\mathsf{abort}}.\mathbf{0}\rangle_5\,|\\
&\qquad \langle\mathsf{reply}(y).\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathbf{0}\,,\ \mathsf{reply}(y).\mathbf{0}+\mathsf{abort}.\mathbf{0}\,|\,\overline{(t_{stat})}.\overline{\mathsf{idle}}.\mathbf{0}\rangle_5) \setminus \{\mathsf{reply},\mathsf{abort}\}
\end{aligned}
$$

図 12 Reader, Writer クラスの時間的観測等価式
Fig. 12 Equivalent expressions of **Reader, Writer** class and initial program.

$$Prog \approx_{\mathcal{T}} Prog' \text{ where}$$
$$Prog' \stackrel{\text{def}}{=} (ClassDec'_{Buffer} | ClassDec'_{Writer} | ClassDec'_{Reader} | ObjId_0 | Machine_N | Initial')$$

図 13 読み手・書き手プログラムの時間的観測等価式
Fig. 13 Equivalent expression of Reader/Writer Program.

与えたり逆に与えられることはないことから，メソッド呼出し・返答などの外部的特性とその実行タイミングだけを考えれば十分なことが多い．このため，図10のバッファクラスの定義（$ClassDec_{Buffer}$）は，それと時間的観測等価になる簡単化された動作式 $ClassDec'_{Buffer}$（**図 11**）を通じてより容易に解析することができるようになる．同様に他のクラス定義と初期化プログラムに関しても時間的観測等価となる動作式が存在し，それらは**図 12** のようになる．

また，時間的観測等価は任意のプロセス $P_1, P_2, Q \in \mathcal{P}$ ただし $P_1 \approx_{\mathcal{T}} P_2$ とするとき，次の関係を満足する．

（1） $\alpha.P_1 \approx_{\mathcal{T}} \alpha.P_2$
（2） $P_1|Q \approx_{\mathcal{T}} P_2|Q$
（3） $P_1\backslash L \approx_{\mathcal{T}} P_2\backslash L$
（4） $P_1[f] \approx_{\mathcal{T}} P_2[f]$
（5） $\langle Q, P_1\rangle_t \approx_{\mathcal{T}} \langle Q, P_2\rangle_t$

これは時間的観測等価が，$\mathcal{R}_{\mathcal{T}}$ のクラス定義や，インスタンス，プログラム文を変換した動作式を結合する演算子に関して保存されることを示している．これにより，これらの構文要素を表す動作式は，それと時間的観測等価になる動作式に替えても，プログラム全体の動作内容と時間性は等価になることが保証される．よって図8のプログラムは，**図 13** に示された簡単化された動作式を合成した式で代用することができる．そしてこの簡単化された式を展開することにより，もとのプログラムがもつオブジェクト間相互作用の動作内容的・時間的特性が解析できるようになる．

（平成6年4月11日受付）
（平成6年7月14日採録）

佐藤 一郎

1991年慶應義塾大学理工学部電気工学科卒業．1993年同大学大学院理工学研究科計算機科学専攻修士課程修了．現在，同大学大学院博士課程在学中．並行・分散計算モデル，オブジェクト指向計算システム，プログラミング言語処理系，プログラミング環境に興味を持っている．日本ソフトウェア科学会，ACM 各会員．

所 真理雄（正会員）

1970年慶應義塾大学工学部電気工学科卒業．1975年同大学博士課程修了．工学博士．同大学電気工学科助手，専任講師，助教授を経て現在教授．その間，1979年ウォータールー大学訪問助教授，1980年カーネギーメロン大学訪問助教授．1988年よりソニーコンピュータサイエンス研究所副所長を兼務．計算モデル，プログラミング言語，分散・開放型システム，人工知能などに興味を持っている．主要著書に「計算システム入門」（岩波書店），「Object Oriented Concurrent Programming」（MIT Press, 編書）などがある．日本ソフトウェア科学会，電子情報通信学会，ACM, IEEE ほか各会員．