時間付き π 計算によるリアルタイムオブジェクト指向言語の形式的記述

桑原寛明[†] 結縁祥治[†,††] 阿草清滋[†]

本論文では実時間システム開発にオブジェクト指向開発技術を適用する基礎とするために，形式計算モデルである π 計算に基づきリアルタイムオブジェクト指向言語の振舞いを定式化する．π 計算に離散時間の振舞いを拡張し，単純なリアルタイムオブジェクト指向言語 OOL_{RT} の振舞いを記述する．実時間システムは一般に時間制約を持つ複数のオブジェクトの並行動作によって実現される．OOL_{RT} によって実時間システムの特性を直接的に記述し，時間拡張された π 計算によってその振舞いを厳密に定義することで，システムの動作の解析や検証を形式的に行うための枠組みを与える．

A Formal Description of a Real-Time Object-Oriented Language by the π-Calculus with Time

Hiroaki Kuwabara,[†] Shoji Yuen[†,††] and Kiyoshi Agusa[†]

In this paper, we aim at providing a foundational framework of the object- oriented technique for system development with timing constraints. We formalize timed behavior of objects via the behavior of π-calculus extended with time. It is common to model a real-time system by composing concurrent objects with timing constraints. To capture the features of real-time objects, we define a simple programming language OOL_{RT} to give the operational semantics by translating a program of OOL_{RT} into a term of our timed π-calculus. By this translation, we obtain an abstract behavioral model for real-time objects to analyze and verify the behavioral properties of real-time systems.

1. はじめに

携帯電話や交通管理システム，ブレーキングシステムなど様々な実時間システムが開発され利用されている．実時間システムの多くは組み込みシステムであり，停止することなく動作し続けなければならない．ペースメーカのように人命にかかわるために誤動作が絶対に許されないシステムもあり，多くの実時間システムには高い信頼性が要求される．実時間システムはセンサなどを通して外部環境と相互作用することが多く，時間経過によっても動作が変わるため，テストによってつねに正常動作することを確かめることは難しい．そのため，コストが大きくなっても形式的な検証手法によってシステムの動作の正しさを示すことが必要である．オブジェクト指向開発技法では，システムを相互作用により計算を行う複数のオブジェクトの集合であるとしてモデル化する．相互に通信を行う単位として 1 つのプロセスを 1 つのオブジェクトと見なす．オブジェクト指向開発技法を適用することで，ソフトウェアの部品化による再利用性の向上，変更に対する影響範囲の限定による保守性の向上といった利点が得られる．

実時間システムは動作に時間制約を持つ並行システムであり，与えられた時間制約を満たすことが要求される[3]．システムの動作の正しさは，結果の正しさだけではなく結果が得られるまでに経過した時間の長さにも依存する．動作の実行時間は実行時のオブジェクト生成やメッセージ通信，並行性など動的な要因に影響される．そのため実装後に実際に動作させてテストを行うだけでシステムがつねに正しく動作することを確認することは難しい．実時間システムが正常に動作することを確認するためにシステムの時間を含む動作を記号として定式化する．形式的な厳密さを導入することで動作の正しさの検証の基礎とすることができる．

この問題に対し，実時間システムの動作を制御する

† 名古屋大学大学院情報科学研究科情報システム学専攻
Department of Information Engineering, Graduate School of Information Science, Nagoya University
†† 科学技術振興機構さきがけ研究 21
PRESTO/JST

ソフトウェアとしてのリアルタイムオブジェクト指向言語から π 計算による記述への変換規則を定義する．実装段階で作成されたプログラムから π 計算による記述に変換することで，実時間システムの特にソフトウェアの抽象的な形式モデルを得る．形式的なモデルにより実時間システムの解析と検証の基礎を与える．仕様に対する形式的記述とプログラムから変換して得られる記述との振舞い等価性を調べることが可能となり，実装が仕様に従っているか確認できる．

π 計算[8),10)] はプロセス代数の一種で並行計算モデルである．プロセス間通信に利用するリンク自身をメッセージとする通信とプロセスの動的生成を表現することが可能であり，高い表現能力を持っている．しかし，π 計算は振舞いに関して時間の概念がなく，時間に関する性質を表現できない．プロセス代数に導入されている時間概念を導入して π 計算を拡張する．離散時間の経過を表すプリミティブを導入し，選択演算子を用いてタイムアウトをモデル化する．時間に依存した振舞いはタイムアウトによって表現できることが知られており[7)]，この拡張によって時間に依存する振舞いを π 計算で記述できるようになる．本論文では，拡張による時間の概念を並行オブジェクト指向プログラムに反映させ，Walker の変換手法[11)] を時間に拡張する．この変換によって，本論文における時間拡張が実際のプログラミングモデルへの厳密な意味で時間の導入に応用可能であることを示し，プログラムの振舞いの性質の解析の基礎技法とできることを示す．

本論文の構成は以下のとおりである．2 章で π 計算に対する時間拡張について述べる．3 章で簡単なリアルタイムオブジェクト指向言語 OOL_{RT} を示し，OOL_{RT} から π 計算による記述への変換規則を与える．4 章で OOL_{RT} プログラムとその変換の簡単な具体例をあげる．5 章で関連研究に触れ，最後にまとめと今後の課題を述べる．

2. π 計算に対する時間拡張

π 計算ではプロセス間通信に用いるリンクを**名前**と呼び，名前が π 計算の最も基本的な要素である．名前自身をメッセージとして送受信することにより計算を表す．本章では，時間遷移を表すための構文と動作意味を与える．

本拡張では，実時間システムの時間を含む動作を定式化し記述することを目指す．システムの動作は環境に対する入出力動作であり，連続する 2 つの入出力動作の間で経過する時間の長さを記述できれば，いつどれだけ時間が経過するか表現できる．そこで時間の経過をアクションと見なし入出力動作と同じように記述する．システムが動作する計算機上では，経過する時間の長さは離散的に数えられるため，離散時間によるモデル化を行う．

2.1 構　　文

Milner らの π 計算[9)] に自然数によって添字付けされたプレフィックス t を導入する．n 単位時間でタイムアウトする時間待ちを $t[n]$ と記述し，時間経過アクションと呼ぶ．

以下では，$\mathcal{N}ame$ を名前の集合，$\mathcal{I}$ を自然数を表す名前の集合とする．$\mathcal{I} = \{0, 1, \ldots\} \subset \mathcal{N}ame$ である．さらに $\mathcal{N} = \mathcal{N}ame - \mathcal{I}$ とし，$\overline{\mathcal{N}} = \{\overline{x} | x \in \mathcal{N}\}$，$\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}}$，$a, x \in \mathcal{N}$ とする．$\widetilde{y}$ は名前のリストを表し，i 番目の名前を y_i と書く．π 計算のプロセス式全体の集合を $\mathcal{P}$ と書き，アクションの集合を $Act = \mathcal{L} \cup \{\tau\}$ と記す．

プロセス式を以下のように定義する．

定義 1　時間拡張した π 計算のプロセス式 P は以下の構文によって定義される．

$$\pi ::= x(\widetilde{y}) \mid \overline{x}\langle\widetilde{y}\rangle \mid \tau \mid t[n]$$
$$P ::= M \mid P_1 | P_2 \mid \boldsymbol{\nu} a\, P \mid !\, P$$
$$M ::= \mathbf{0} \mid \pi.P \mid M_1 + M_2$$

ここで $!P$ における P は以下で定義されるアクションガード付きプロセスである．

$\pi ::= x(\widetilde{y}) \mid \overline{x}\langle\widetilde{y}\rangle \mid \tau$ とするとき，

- 任意のプロセス P に対し $\pi.P$ はアクションガード付きプロセスである
- P_1, P_2 がアクションガード付きプロセスであるとき

  $P_1|P_2,\ P_1 + P_2,\ \boldsymbol{\nu} a\, P_1,\ !\, P_1$

  はアクションガード付きプロセスである　□

定義 2　$P = t[x].P'$ に対し，$x \in \mathcal{N}$ かつ x が P を内部に含むプロセスにおいて入力アクションによって束縛されていない場合，P は動作不能であるといい $P\!\uparrow$ と書く．また $P\!\uparrow$ のとき，

$$(\overline{x}\langle\widetilde{y}\rangle.P)\!\uparrow, (x(\widetilde{y}).P)\!\uparrow, (\tau.P)\!\uparrow, (t[n].P)\!\uparrow,$$
$$(P+Q)\!\uparrow, (P \mid Q)\!\uparrow, (\boldsymbol{\nu} z\, P)\!\uparrow, (!\, P)\!\uparrow$$

とする．動作不能でない場合，動作可能であるといい $P\!\downarrow$ と書く．　□

プレフィックス $t[n]$ は n 単位時間の待機を表す．たとえばプロセス $t[5].P$ は 5 単位時間後に P のように振る舞う．時間経過アクションと非決定的選択を用いることで，一定時間以内にイベントが発生したか否かで動作が異なる，典型的なタイムアウトを以下のように記述できる．プロセス $a.P + t[5].\tau.Q$ はアクション

$$\text{OUT}: \frac{}{\overline{x}\langle\widetilde{y}\rangle.P \xrightarrow{\overline{x}\langle\widetilde{y}\rangle} P} \qquad \text{INPUT}: \frac{}{x(\widetilde{y}).P \xrightarrow{x(\widetilde{z})} P\{\widetilde{z}/\widetilde{y}\}} \qquad \text{TAU}: \frac{}{\tau.P \xrightarrow{\tau} P}$$

$$\text{SUM-L}: \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad \text{SUM-R}: \frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$$

$$\text{PAR-L}: \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset$$

$$\text{PAR-R}: \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ \mathsf{bn}(\alpha) \cap \mathsf{fn}(P) = \emptyset$$

$$\text{COMM-L}: \frac{P \xrightarrow{\overline{x}(\widetilde{y})} P' \quad Q \xrightarrow{x(\widetilde{y})} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{COMM-R}: \frac{P \xrightarrow{x(\widetilde{y})} P' \quad Q \xrightarrow{\overline{x}(\widetilde{y})} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\text{RES}: \frac{P \xrightarrow{\alpha} P'}{\boldsymbol{\nu} x\, P \xrightarrow{\alpha} \boldsymbol{\nu} x\, P'}\ \text{if } \alpha \notin \{x, \overline{x}\}$$

$$\text{REP-ACT}: \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \qquad \text{REP-COMM}: \frac{P \xrightarrow{\overline{x}(\widetilde{y})} P' \quad P \xrightarrow{x(\widetilde{y})} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P}$$

$$\text{TIMEOUT}: \frac{P \xrightarrow{\alpha} P'}{t[0].P \xrightarrow{\alpha} P'} \qquad \text{ABORT}: \frac{}{P \xrightarrow{abort}}\ \text{if } P\!\uparrow$$

図 1 遷移規則
Fig. 1 The transition rules.

$$\text{PASS}_\text{T}: \frac{}{t[n].P \rightsquigarrow t[n-1].P}\ \text{if } n > 0 \qquad \text{INACT}_\text{T}: \frac{}{\mathbf{0} \rightsquigarrow \mathbf{0}}$$

$$\text{OUT}_\text{T}: \frac{}{\overline{x}\langle\widetilde{y}\rangle.P \rightsquigarrow \overline{x}\langle\widetilde{y}\rangle.P} \qquad \text{IN}_\text{T}: \frac{}{x(\widetilde{y}).P \rightsquigarrow x(\widetilde{y}).P}$$

$$\text{SUM}_\text{T}: \frac{P \rightsquigarrow P' \quad Q \rightsquigarrow Q'}{P+Q \rightsquigarrow P'+Q'} \qquad \text{PAR}_\text{T}: \frac{P \rightsquigarrow P' \quad Q \rightsquigarrow Q'}{P \mid Q \rightsquigarrow P' \mid Q'}\ \text{if } P \mid Q \not\xrightarrow{\tau}$$

$$\text{RES}_\text{T}: \frac{P \rightsquigarrow P'}{\boldsymbol{\nu} x\, P \rightsquigarrow \boldsymbol{\nu} x\, P'} \qquad \text{REP}_\text{T}: \frac{P \rightsquigarrow P'}{!P \rightsquigarrow !P'}\ \text{if } P \mid P \not\xrightarrow{\tau}$$

$$\text{STRUCT}_\text{T}: \frac{P \rightsquigarrow P'}{Q \rightsquigarrow Q'}\ \text{if } P \equiv Q,\ P' \equiv Q'$$

$$\text{TIMEOUT}_\text{T}: \frac{P \rightsquigarrow P'}{t[0].P \rightsquigarrow P'}$$

図 2 時間経過規則
Fig. 2 The rules for time progression.

a の発生まで最大で 5 単位時間待機する．5 単位時間経過する前に a が発生すれば P へ遷移する．5 単位時間経過したときにアクション a が実行可能であれば P へ，実行不可能であればタイムアウトし τ により Q へ遷移する．

プロセス $P = \overline{a}\langle m\rangle.\mathbf{0} \mid a(n).t[n].\overline{b}.\mathbf{0}$ は τ 遷移により $Q = \mathbf{0} \mid t[m].\overline{b}.\mathbf{0}$ となる．このとき $P \not\downarrow$ であるが，$m \in \mathcal{N}$ とすると定義 2 より $Q\!\uparrow$ となる．

他のプレフィックスと演算子の直観的な意味を以下に示す．

- （入力）$x(\widetilde{y})$：x を通して $\widetilde{y}$ を受信．
- （出力）$\overline{x}\langle\widetilde{y}\rangle$：$x$ を通して $\widetilde{y}$ を送信．
- （τ 動作）τ：外部からは不可視な内部アクション．
- （選択）$P+Q$：プロセス P，Q のうち実行可能なプロセスを選択し実行する．いずれのプロセスも実行可能であれば一方を非決定的に選択する．
- （並行）$P|Q$：プロセス P，Q が並行に動作する．
- （制限）$\boldsymbol{\nu} a\, P$：プロセス P 中の自由な変数 a を束縛する．
- （複製）$!P$：無限個のプロセス P が ｜ 演算子によって結合していると見なす．

2.2 時間動作意味

従来の π 計算の動作意味を時間に関して拡張する．時間の経過は時間経過規則に基づく遷移によってのみ発生し，他の規則による遷移では時間は経過しないとする．

$\mathcal{P}$ 上の遷移関係 $\{\xrightarrow{\alpha} \mid \alpha \in Act\} \cup \rightsquigarrow$ は**図 1** の遷移規則および**図 2** の時間経過規則によって定義される．ABORT 規則以外のすべての規則において $P \not\downarrow$，$Q \not\downarrow$ とする．また，図 1，図 2 において $x \in \mathcal{N}$，$n \in \mathcal{I}$，$y \in \mathcal{N}ame$，$z \in \mathcal{N}ame$ とする．

ラベル付き遷移は

- プロセスが他のプロセスとどのように通信するか，
- プロセスが単位時間ごとにどのように遷移するか，

を示す．$P \xrightarrow{\alpha} P'$ は入力，出力，内部アクションのいずれかのアクション α により P から P' に遷移することを表し，$P \rightsquigarrow P'$ は P が 1 単位時間の経過により P' に遷移することを表す．

SUM_T 規則，PAR_T 規則，REP_T 規則により時間はすべてのプロセスにおいて同時に経過する．PASS_T 規則から時間経過によりプレフィックス $t[n]$ は $t[n-1]$ となり，タイムアウトまでの時間が 1 単位時間短くなったことを示す．PAR_T 規則と REP_T 規則から τ による遷移は時間経過による遷移に優先して発生する．

2.3 時間的性質

時間拡張した π 計算が時間決定性および最大進行性を満たすことを示す．

定理 1（時間決定性） プロセス P に対し，$P \rightsquigarrow P'$ かつ $P \rightsquigarrow P''$ ならば $P' = P''$ である．

証明： P の構造に関する帰納法による．ここでは $P = Q \mid R$ および $P = !Q$ の場合について示す．

- $P = Q \mid R$ のとき，$P \rightsquigarrow P' = Q' \mid R'$，$P \rightsquigarrow P'' = Q'' \mid R''$ とすると帰納法の仮定より $Q' = Q'', R' = R''$ ゆえ $P' = P''$．
- $P = !Q$ のとき，P はアクションガード付きプロセスであるため $P \rightsquigarrow P$． □

時間決定性は時間の経過による遷移が決定的であり，プロセスにおいて時間経過による遷移先が一意に決まることを示す．時間の経過方法が 1 通りであることを反映している．

定理 2（最大進行性） プロセス P に対し，$P \xrightarrow{\tau}$ ならば $P \not\rightsquigarrow$ である．

証明： P の構造に関する帰納法による．ここでは $P = Q \mid R$ の場合について示す．

$P \xrightarrow{\tau} P'$ とする．$Q \xrightarrow{\tau} Q'$ あるいは $R \xrightarrow{\tau} R'$ ならば帰納法の仮定より $Q \not\rightsquigarrow$ あるいは $R \not\rightsquigarrow$．ここで $P \rightsquigarrow P''$ とすると $\exists Q''\ Q \rightsquigarrow Q''$ かつ $\exists R''\ R \rightsquigarrow R''$ でなければならないが，これは帰納法の仮定に反する．よって $P \not\rightsquigarrow$ である．$Q \xrightarrow{\alpha} Q', R \xrightarrow{\overline{\alpha}} R'$ ならば $Q \mid R \xrightarrow{\tau} Q' \mid R'$ であり，$\mathrm{PAR_T}$ 規則の条件により $P \not\rightsquigarrow$ である． □

最大進行性は τ 遷移が可能であれば時間経過が発生しないことを示す．この性質によりプロセス間通信は実行可能になればすぐに実行される．

最大進行性により $\tau.P$ はアージェントプロセスである．そのため，たとえば以下の 2 つのプロセス

$$P \stackrel{def}{=} a.\mathbf{0} + t[1].b.\mathbf{0}$$
$$Q \stackrel{def}{=} a.\mathbf{0} + t[1].\tau.b.\mathbf{0}$$

の振舞いは異なる．$R \stackrel{def}{=} t[2].\overline{a}.\mathbf{0}$ とすると

$$\begin{aligned} P \mid R &\rightsquigarrow a.\mathbf{0} + t[0].b.\mathbf{0} \mid t[1].\overline{a}.\mathbf{0} \\ &\rightsquigarrow a.\mathbf{0} + b.\mathbf{0} \mid t[0].\overline{a}.\mathbf{0} \\ &\xrightarrow{\tau} \mathbf{0} \mid \mathbf{0} \\ Q \mid R &\rightsquigarrow a.\mathbf{0} + t[0].\tau.b.\mathbf{0} \mid t[1].\overline{a}.\mathbf{0} \\ &\xrightarrow{\tau} b.\mathbf{0} \mid t[1].\overline{a}.\mathbf{0} \end{aligned}$$

となり，$P \mid R$ では通信が発生するが $Q \mid R$ では最大進行性により通信は発生しない．

2.4 双模倣関係

時間拡張された π 計算における双模倣関係を以下のように定義する．

定義 3 以下を満たす対称な関係 $\mathcal{S}$ を**時間付き強双模倣関係**と呼び，最大の関係を $\sim_{\mathcal{T}}$ と書く．
$(P,Q) \in \mathcal{S}$ のとき，

- $P\!\uparrow \ \Rightarrow \ Q\!\uparrow$
- $P \xrightarrow{\alpha} P' \ \Rightarrow \ \exists Q' \,.\, Q \xrightarrow{\alpha} Q' \,.\, (P',Q') \in \mathcal{S}$
- $P \rightsquigarrow P'' \ \Rightarrow \ \exists Q'' \,.\, Q \rightsquigarrow Q'' \,.\, (P'',Q'') \in \mathcal{S}$ □

定義 4 以下を満たす対称な関係 $\mathcal{S}$ を**時間付き弱双模倣関係**と呼び，最大の関係を $\approx_{\mathcal{T}}$ と書く．
$(P,Q) \in \mathcal{S}$ のとき，

- $P\!\uparrow \ \Rightarrow \ Q\!\uparrow$
- $P \xrightarrow{\alpha}_{\tau} P' \ \Rightarrow \ \exists Q' \,.\, Q \xrightarrow{\alpha}_{\tau} Q' \,.\, (P',Q') \in \mathcal{S}$
- $P \rightsquigarrow_{\tau} P'' \ \Rightarrow \ \exists Q'' . Q \rightsquigarrow_{\tau} Q''. (P'',Q'') \in \mathcal{S}$

ただし $\xrightarrow{\alpha}_{\tau} = (\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^*$，$\rightsquigarrow_{\tau} = (\xrightarrow{\tau})^* \rightsquigarrow (\xrightarrow{\tau})^*$ とする． □

定理 3 $\sim_{\mathcal{T}}, \approx_{\mathcal{T}}$ は等価関係である．

証明： 反射性と対称性については容易に証明できる．ここでは $\sim_{\mathcal{T}}$ の推移性について示す．$\sim_{\mathcal{T}}\sim_{\mathcal{T}} \subseteq \sim_{\mathcal{T}}$ であることを示せばよい．ここでは時間経過遷移についてのみ述べる．

$$\begin{aligned} Pdec &::= \mathsf{program}\ P\ \mathsf{is}\ Cdec_1, \ldots, Cdec_n \\ &\quad\ \mathsf{with}\ E \\ Cdec &::= \mathsf{class}\ A\ \mathsf{is}\ Vdecs,\ Mdecs \\ Mdecs &::= Mdec_1, \ldots, Mdec_m \\ Mdec &::= \mathsf{method}\ M(X_1, \ldots, X_l)\ \mathsf{is}\ S \\ Vdecs &::= \mathsf{var}\ X_1, \ldots, X_k \\ S &::= E \mid X := E \mid S_1; S_2 \\ &\mid \mathsf{wait}\ E \\ &\mid \mathsf{return}\ E \\ &\mid \mathsf{if}\ E\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{endif} \\ &\mid \mathsf{while}\ E\ \mathsf{do}\ S\ \mathsf{done} \\ &\mid \mathsf{within}\ E\ \mathsf{do}\ S_1\ \mathsf{timeout}\ S_2\ \mathsf{done} \\ E &::= X \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{nil} \\ &\mid Numeral \\ &\mid \mathsf{new}(A) \\ &\mid E!M(E_1, \ldots, E_l) \end{aligned}$$

図 3 OOL_{RT} の構文
Fig. 3 The syntax of OOL_{RT}.

$(P,R) \in \sim_{\mathcal{T}}\sim_{\mathcal{T}}$ とすると，$(P,Q) \in \sim_{\mathcal{T}}$ かつ $(Q,R) \in \sim_{\mathcal{T}}$ となる Q が存在する．このとき $P \rightsquigarrow P''$ とすると，$(P,Q) \in \sim_{\mathcal{T}}$ より $Q \rightsquigarrow Q''$ かつ $(P'',Q'') \in \sim_{\mathcal{T}}$ を満たす Q'' が存在し，さらに $(Q,R) \in \sim_{\mathcal{T}}$ ゆえ Q'' に対し $R \rightsquigarrow R''$ かつ $(Q'',R'') \in \sim_{\mathcal{T}}$ となる R'' が存在する．よって $P'' \sim_{\mathcal{T}} Q'' \sim_{\mathcal{T}} R''$ であるので $\sim_{\mathcal{T}}\sim_{\mathcal{T}} \subseteq \sim_{\mathcal{T}}$． □

3. OOL_{RT} から π 計算への変換規則

3.1 リアルタイムオブジェクト指向言語 OOL_{RT}

OOL_{RT} は簡単なオブジェクト指向言語 OOL [10] からデータ型の取扱いを省き，時間待ちとタイムアウトのための構文を追加した言語である．

OOL_{RT} の構文を**図 3** に示す．ここで P はプログラム名，E は式，A はクラス名，M はメソッド名，S は文，X は変数名を表す．

3.2 変換規則

OOL_{RT} の構文要素それぞれに対し時間拡張した π 計算による記述への変換規則を与える．構文要素 A に対応する変換規則を $[\![A]\!]$ と表す．

プログラム $Pdec$ に対する π 計算記述への変換規則を**図 4** に示す．$\widetilde{k}$ はプログラム中で定義されるクラスの名前のリストを示す．Boolean はブール値を表すクラスでありプログラムには標準で組み込まれる．名前 k_{Bool} は Boolean クラスを表し $k_{Bool} \in \widetilde{k}$ とする．

クラス定義 $Cdec$ に対する π 計算記述への変換規則

$$[\![Pdec]\!] \equiv (\boldsymbol{\nu}\widetilde{k})([\![Cdec_1]\!](\widetilde{k}) \mid \cdots \mid [\![Cdec_n]\!](\widetilde{k}) \mid \mathrm{Boolean}(k_{Bool}) \mid (\boldsymbol{\nu}l)[\![E]\!](l, \mathsf{nil}, \widetilde{k}))$$
$$[\![Cdec]\!](\widetilde{k}) \equiv !\,(\boldsymbol{\nu}a)(k(l).\overline{l}\langle a\rangle.(\boldsymbol{\nu}\widetilde{x}, \widetilde{m})([\![Vdecs]\!](\widetilde{x}, \widetilde{c} \cup \{a\}) \mid [\![Mdecs]\!](\widetilde{m}, a, \widetilde{x}, \widetilde{k})))$$
$$[\![Vdecs]\!](\widetilde{x}, \widetilde{c}) \equiv [\![\mathsf{var}\ X_1]\!](x_1, c_1) \mid \cdots \mid [\![\mathsf{var}\ X_k]\!](x_k, c_k)$$
$$[\![\mathsf{var}\ X]\!](x, c) \equiv (\boldsymbol{\nu}l)(x(r, u).(\overline{r}\langle c\rangle.\overline{l}\langle c\rangle \mid u(c').\overline{l}\langle c'\rangle) \mid !\,l(c).x(r, u).(\overline{r}\langle c\rangle.\overline{l}\langle c\rangle \mid u(c').\overline{l}\langle c'\rangle))$$

図 4 プログラム，クラス定義，変数宣言に対する π 計算記述への変換規則
Fig. 4 Translation rules for $Pdec$, $Cdec$ and $Vdecs$.

$$[\![Mdecs]\!](\widetilde{m}, a, \widetilde{x}, \widetilde{k}) \equiv !\,a(n).\overline{n}\langle\widetilde{m}\rangle.([\![Mdec_1]\!](m_1, \widetilde{x}, \widetilde{k}) \mid \cdots \mid [\![Mdec_m]\!](m_n, \widetilde{x}, \widetilde{k}))$$
$$[\![Mdec]\!](m, \widetilde{x}, \widetilde{k}) \equiv (\boldsymbol{\nu}g, \widetilde{y}, \widetilde{r}, \widetilde{u})$$
$$(m(\widetilde{v}, t, l).\overline{y_1}\langle r_1, u_1\rangle.\overline{u_1}\langle v_1\rangle.\cdots.\overline{y_l}\langle r_l, u_l\rangle.\overline{u_l}\langle c_l\rangle.\overline{g}\langle t, l\rangle$$
$$\mid [\![\mathsf{var}\ X_1]\!](y_1, \mathsf{nil}) \mid \cdots \mid [\![\mathsf{var}\ X_l]\!](y_l, \mathsf{nil}) \mid g(t, l).[\![S]\!](l, t, \widetilde{x} \cup \widetilde{y}, \widetilde{k}))$$

図 5 メソッド定義に対する π 計算記述への変換規則
Fig. 5 Translation rules for $Mdecs$.

を図 4 に示す．名前 k がここで定義されているクラスを示し，名前 a がこのクラスのインスタンスの 1 つを示す．インスタンス生成のとき，以下のように動作する．

$$\begin{aligned}
&[\![Cdec]\!](\widetilde{k})\\
&\xrightarrow{k(l)} (\boldsymbol{\nu}a)\,(\overline{l}\langle a\rangle.(\boldsymbol{\nu}\widetilde{x}, \widetilde{m})\,(\cdots)) \mid [\![Cdec]\!](\widetilde{k})\\
&\xrightarrow{\overline{l}\langle a\rangle} (\boldsymbol{\nu}\widetilde{x}, \widetilde{m})\,(\cdots) \mid [\![Cdec]\!](\widetilde{k})
\end{aligned}$$

k によるメッセージの受信がインスタンスの生成要求を意味する．このときに渡される名前 l を通して新しいインスタンスを示す名前 a を返す．a を用いてメソッドにアクセスする．$\boldsymbol{\nu}$ 演算子により個々のインスタンスを示す名前はすべて異なる．$\widetilde{x}$ はメンバ変数の名前のリスト，$\widetilde{m}$ はメソッドの名前のリスト，$\widetilde{c}$ は各メンバ変数の初期値のリストを表す．メンバ変数にはインスタンス自身を示す $this$ が含まれる．

変数宣言 $Vdecs$ に対する π 計算記述への変換規則を図 4 に示す．$\widetilde{x}$ が各変数の名前のリスト，$\widetilde{c}$ が各変数の初期値のリストを示す．それぞれの変数にアクセスするには変数を示す名前 x を通して 2 つの名前 r，u を渡す．値を参照する場合は r を通して値を受信する．値を更新する場合は u を通して新しい値を送信する．

メソッド定義 $Mdecs$ および $Mdec$ に対する π 計算記述への変換規則を**図 5** に示す．すべてのメソッドは必ずいずれかのインスタンスに属している．そのインスタンスをクラス定義 $[\![Cdec]\!]$ において新たに生成される名前 a によって示す．メソッド呼び出し時の動作例を以下に示す．

$$\begin{aligned}
&[\![Mdecs]\!](\widetilde{m}, a, \widetilde{x}, \widetilde{k})\\
&\xrightarrow{a(n)} \overline{n}\langle\widetilde{m}\rangle.([\![Mdec_1]\!](m_1, \widetilde{x}, \widetilde{k}) \mid \cdots)\\
&\quad \mid [\![Mdecs]\!](\widetilde{m}, a, \widetilde{x}, \widetilde{k})\\
&\xrightarrow{\overline{n}\langle\widetilde{m}\rangle} ([\![Mdec_1]\!](m_1, \widetilde{x}, \widetilde{k}) \mid \cdots)\\
&\quad \mid [\![Mdecs]\!](\widetilde{m}, a, \widetilde{x}, \widetilde{k})\\
&\xrightarrow{m(\widetilde{v}, r, l)} \cdots\cdots
\end{aligned}$$

a を通して要求を受けるとそのインスタンスに属するメソッドを示す名前すべてを返す．その中から呼び出し元が希望するメソッドを示す名前 m を選択し，その名前を通して引数 $\widetilde{v}$ などを送信することでメソッド呼び出しとする．$\widetilde{y}$ は実引数を表す名前のリスト，$\widetilde{r}$ は実引数の値の読み出し，$\widetilde{u}$ は実引数の値の更新を行う名前のリストである．m を通して渡される名前 r は return 文により値を返すために利用される．

文 文要素 S に対する π 計算記述への変換規則 $[\![S]\!](l, r, \widetilde{x}, \widetilde{k})$ を**図 6** に示す．l は文の実行完了を示す名前，r は return 文による値の返却を示す名前，$\widetilde{x}$ はメンバ変数，$\widetilde{k}$ はプログラム中で定義されているクラスを示す名前である．

$[\![\mathsf{wait}\ E]\!]$ では E を評価するサブプロセスと時間待ちを行うサブプロセスが並行に動作する．E の評価値をローカルな名前 l' を通して時間待ちを行うサブプロセスに送信し時間待ちが開始する．$[\![\mathsf{within}\ E\ \mathsf{do}\ S_1\ \mathsf{timeout}\ S_2\ \mathsf{done}]\!]$ では初めに E を評価しその値をローカルな名前 l'' を通して送信する．受信側では S_1 を実行するサブプロセスとタイムアウトまで時間を計測するサブプロセスが並行に動作する．時間内に S_1 が完了しなければタイムアウトして S_2 の実行が選択される．

$$
\begin{aligned}
[\![X := E]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l')([\![E]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)(l'(v).\overline{x_X}\langle r, u\rangle.\overline{u}\langle v\rangle.\overline{l})) \\
[\![E]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l')([\![E]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\overline{l}) \\
[\![S_1; S_2]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l')([\![S_1]\!](l', r, \widetilde{x}, \widetilde{k}) \mid l'.[\![S_2]\!](l, r, \widetilde{x}, \widetilde{k})) \\
[\![\mathsf{wait}\ E]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l')([\![E]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(n).t[n].\overline{l}) \\
[\![\mathsf{return}\ E]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l')([\![E]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\overline{r}\langle v\rangle.\overline{l}) \\
[\![\mathsf{if}\ E\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{endif}]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l', l_1, l_2) \\
&\quad ([\![E]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} t, f, \widetilde{n})(l'(v).\overline{v}\langle\widetilde{n}\rangle.\overline{n_1}\langle t, f\rangle.(t.\overline{l_1} \mid f.\overline{l_2})) \\
&\quad \mid l_1.[\![S_1]\!](l, r, \widetilde{x}, \widetilde{k}) \mid l_2.[\![S_2]\!](l, r, \widetilde{x}, \widetilde{k})) \\
[\![\mathsf{while}\ E\ \mathsf{do}\ S\ \mathsf{done}]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} g, l', l_1, l_2) \\
&\quad (\overline{g} \mid !\, g.([\![E]\!](l', \widetilde{x}, \widetilde{k}) \\
&\qquad \mid (\boldsymbol{\nu} t, f, \widetilde{n})(l'(v).\overline{v}\langle\widetilde{n}\rangle.\overline{n_1}\langle t, f\rangle.(t.\overline{l_1} \mid f.\overline{l_2})) \\
&\qquad \mid l_1.[\![S]\!](g, r, \widetilde{x}, \widetilde{k}) \mid l_2.\overline{l})) \\
[\![\mathsf{within}\ E\ \mathsf{do}\ S_1\ \mathsf{timeout}\ S_2\ \mathsf{done}]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l', l'', r') \\
&\quad ([\![E]\!](l'', \widetilde{x}, \widetilde{k}) \\
&\quad \mid l''(n).([\![S_1]\!](l', r', \widetilde{x}, \widetilde{k}) \\
&\qquad \mid ((r'(v).\overline{r}\langle v\rangle \mid l'.\overline{l}) + t[n].\tau.[\![S_2]\!](l, r, \widetilde{x}, \widetilde{k}))))
\end{aligned}
$$

図 6 文要素に対する π 計算記述への変換規則
Fig. 6 Translation rules for statements.

$$
\begin{aligned}
[\![\mathsf{true}]\!](l, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l', t, f)(\overline{k_{Bool}}\langle l', t, f\rangle.\overline{t}.l'(b).\overline{l}\langle b\rangle) \\
[\![\mathsf{false}]\!](l, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l', t, f)(\overline{k_{Bool}}\langle l', t, f\rangle.\overline{f}.l'(b).\overline{l}\langle b\rangle) \\
[\![\mathsf{nil}]\!](l, \widetilde{x}, \widetilde{k}) &\equiv \overline{l} \\
[\![Numeral]\!](l, \widetilde{x}, \widetilde{k}) &\equiv \overline{l}\langle Numeral\rangle \\
[\![\mathsf{new}(A)]\!](l, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l')(\overline{k_A}\langle l'\rangle.l'(c).\overline{l}\langle c\rangle) \\
[\![X]\!](l, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} r, u)(\overline{x_X}\langle r, u\rangle.r(v).\overline{l}\langle v\rangle) \\
[\![E!m(E_1, \ldots, E_n)]\!](l, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} h, h_1, \ldots, h_n) \\
&\quad ([\![E]\!](h, \widetilde{x}, \widetilde{k}) \mid [\![E_1]\!](h_1, \widetilde{x}, \widetilde{k}) \mid \cdots \mid [\![E_n]\!](h_n, \widetilde{x}, \widetilde{k}) \\
&\quad \mid h(v).h_1(v_1).\cdots.h_n(v_n).(\boldsymbol{\nu} r, l', n)(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_M}\langle v_1, \ldots, v_n, r, l'\rangle \\
&\qquad \mid (r(v).\overline{l}\langle v\rangle \ +\ l'.\overline{l})))
\end{aligned}
$$

図 7 式要素に対する π 計算記述への変換規則
Fig. 7 Translation rules for expressions.

$$
\begin{aligned}
\mathrm{Boolean}(k) &\equiv (\boldsymbol{\nu} b_t, b_f)(\mathrm{BoolClass}(k, b_t, b_f) \mid \mathrm{Bool}_t(b_t, k) \mid \mathrm{Bool}_f(b_f, k)) \\
\mathrm{BoolClass}(k, b_t, b_f) &\equiv !\, k(l, t, f).(t.\overline{l}\langle b_t\rangle \mid f.\overline{l}\langle b_f\rangle) \\
\mathrm{Bool}_v(b, k) &\equiv (\boldsymbol{\nu}\widetilde{n})(!\,\mathrm{BoolBody}(b, \widetilde{n}) \mid !\,\mathrm{BoolVal}_v(n_1) \\
&\quad \mid !\,\mathrm{BoolNot}(n_2, n_1, k) \mid !\,\mathrm{BoolAnd}(n_3, n_1, k)) \\
\mathrm{BoolBody}(b, \widetilde{n}) &\equiv b(\widetilde{n'}).(n'_1(t, f).\overline{n_1}\langle t, f\rangle \mid n'_2(l).\overline{n_2}\langle l\rangle \mid n'_3(b', l).\overline{n_3}\langle b', l\rangle) \\
\mathrm{BoolVal}_t(n_1) &\equiv n_1(t, f).\overline{t} \\
\mathrm{BoolVal}_f(n_1) &\equiv n_1(t, f).\overline{f} \\
\mathrm{BoolNot}(n_2, n_1, k) &\equiv (\boldsymbol{\nu} t, f)\ (n_2(l).\overline{n_1}\langle t, f\rangle.(t.\overline{k}\langle l, t, f\rangle.\overline{f} \mid f.\overline{k}\langle l, t, f\rangle.\overline{t})) \\
\mathrm{BoolAnd}(n_3, n_1, k) &\equiv (\boldsymbol{\nu} t, f)\ (n_3(b', l).\overline{n_1}\langle t, f\rangle.(f.\overline{k}\langle l, t, f\rangle.\overline{f} \\
&\quad \mid t.(\boldsymbol{\nu} t', f')\ (b'(\widetilde{n'}).\overline{n'_1}\langle t', f'\rangle.(t'.\overline{k}\langle l, t, f\rangle.\overline{t} \mid f'.\overline{k}\langle l, t, f\rangle.\overline{f}))))
\end{aligned}
$$

図 8 Boolean クラスに対する π 計算記述への変換規則
Fig. 8 Translation rules for Boolean class.

式 式要素 E に対する π 計算記述への変換規則 $[\![E]\!](l, \widetilde{x}, \widetilde{k})$ を**図 7** に示す．名前 l，$\widetilde{x}$，$\widetilde{k}$ は文要素と同じである．$[\![\mathsf{new}(A)]\!]$ の k_A はクラス A を示す名前，$[\![X]\!]$ の x_X は変数 X を示す名前である．

インスタンス生成 $[\![\mathsf{new}(A)]\!]$ では k_A を通してクラス定義のプロセスに名前 l' を渡し，l' を通してインスタンス c を受信する．

メソッド呼び出し $[\![E!m(E_1, \ldots, E_n)]\!]$ では，初めに呼び出し先のインスタンス E と引数 E_i を評価する．次に呼び出し先のインスタンスを示す名前 v を通して名前 n を送信し，n を通してそのインスタンスが持つメソッドの名前のリスト $\widetilde{m}$ を受信する．そして $\widetilde{m}$ に含まれる目的のメソッドを示す名前 m_M に対し引数を送信する．return 文によって返される値は r を通して受信する．値を返さないメソッドの完了は l' を通して通知される．

Boolean **クラス** Boolean クラスに対する π 計算記述への変換規則を**図 8** に示す．k が Boolean クラスを示す名前，b_t が真，b_f が偽を示す名前である．BoolClass は Boolean クラスにアクセスするためのインタフェースであり，名前 k を通してアクセスする．

4. 変 換 例

ここでは簡単なタイマのプログラムの例を示す．OOL_{RT} によるプログラムを**図 9** に示す．*start* メソッドにより実行を開始し，一定時間ごとに *fire* メソッドを呼び出す．処理内容は *fire* メソッドに実装する．*fire* メソッド本体の振舞いについては本質的ではないので省略する．

```
 1: program P is
 2:
 3:   class Timer is
 4:     method start() is
 5:       while true do
 6:         wait 10;
 7:         this!fire()
 8:       done
 9:
10:     method fire() is
11:       ....
12:
13: with
14:   new(Timer)!start()
```

図 9 タイマのプログラム
Fig. 9 A program of timer.

図 9 のプログラムを図 4，図 5，図 6，図 7 の変換規則を用いて時間拡張した π 計算による記述に変換する．変換結果を**図 10** に示す．

一定時間後に *fire* メソッドが呼ばれることを変換結果を用いて確認する．プログラム上では `wait 10; this!fire()` の部分であり，ここの変換結果である $[\![\mathsf{wait}\ 10; this!fire()]\!]$ を展開した式と遷移の経過を**図 11** に示す．図中の $\rightsquigarrow^n$ は n 回連続して $\rightsquigarrow$ による遷移が発生したことを表す．α は τ などの適切なアクションを示すとする．

初めの状態から τ 遷移が 1 回発生し 2 つのプロセス $t[10].\overline{l'}$，$l'.(\boldsymbol{\nu} l'')\,(P \mid l''(v).\bar{l})$ が並行に動作する．このとき，2 つ目のプロセスは l' でブロックされているため実際に動作するのは 1 つ目のプロセスだけであり，10 単位時間の時間待ちとなる．その後，x_{this}，r，h，v といった名前による遷移が発生しさらに $\overline{m_{fire}}$ による遷移が発生する．m_{fire} は *fire* メソッドを表しており，一定時間経過した後に呼び出されることが分かる．

図 11 はプログラムから変換して得られた式であるため複雑だが，システムの設計段階では同じ動作を

$$t[10].\overline{m_{fire}}.Body^s_{fire}$$

のようにより直接的に表現する．ここで $Body^s_{fire}$ は *fire* メソッドで実装される仕様を表す．この場合遷移は

$$\begin{aligned} & t[10].\overline{m_{fire}}.Body^s_{fire} \\ \rightsquigarrow^{10}\ & t[0].\overline{m_{fire}}.Body^s_{fire} \\ \xrightarrow{\overline{m_{fire}}}\ & Body^s_{fire} \end{aligned}$$

となり，10 単位時間経過した後に *fire* メソッドを呼び出していることが分かる．

図 11 の遷移と比較すると，10 単位時間経過した後に $\overline{m_{fire}}$ による遷移が発生する点が同じである．システム全体の遷移を考えると，図 11 の x_{this}，r などによる遷移は通信の相手となるプロセスがあるため τ 遷移である．そこで時間経過，m_{fire} のような注目するアクション，τ 以外のアクションによる遷移列が，設計段階に記述されるシステムの動作を表す式と，プログラムから変換して得られる式とで同じになるか，つまりそれらの式により表されるプロセスが時間付き弱双模倣関係にあるか調べることで動作の等価性を示すことができる．

$$
\begin{aligned}
P &= (\boldsymbol{\nu} k_{Timer}, k_{Bool}) \\
&\quad (\llbracket Timer \rrbracket(\widetilde{k}) \mid \mathrm{Boolean}(k_{Bool}) \mid (\boldsymbol{\nu} l)\, \llbracket \mathsf{new}(Timer)!start() \rrbracket(l, \mathsf{nil}, \widetilde{k})) \\
\llbracket Timer \rrbracket(\widetilde{k}) &= !(\boldsymbol{\nu} a)\, (k_{Timer}(l).\overline{l}\langle a \rangle.(\boldsymbol{\nu} x_{this}, m_{start}, m_{fire}) \\
&\qquad\qquad (\llbracket \mathsf{var}\ X_{this} \rrbracket(x_{this}, a) \\
&\qquad\qquad \mid (!a(n).\overline{n}\langle \widetilde{m} \rangle.(\llbracket Mdec_{start} \rrbracket(m_{start}, \widetilde{x}, \widetilde{k}) \\
&\qquad\qquad\qquad \mid \llbracket Mdec_{fire} \rrbracket(m_{fire}, \widetilde{x}, \widetilde{k}))))) \\
\llbracket \mathsf{var}\ X_{this} \rrbracket(x, c) &= (\boldsymbol{\nu} l)(x(r, u).(\overline{r}\langle c \rangle.\overline{l}\langle c \rangle \mid u(c').\overline{l}\langle c' \rangle) \\
&\qquad \mid !\, l(c).x(r, u).(\overline{r}\langle c \rangle.\overline{l}\langle c \rangle \mid u(c').\overline{l}\langle c' \rangle)) \\
\llbracket Mdec_{start} \rrbracket(m, \widetilde{x}, \widetilde{k}) &= (\boldsymbol{\nu} g)\, (m(r, l).\overline{g}\langle r, l \rangle \mid g(r, l).\llbracket Body_{start} \rrbracket(l, r, \widetilde{x}, \widetilde{k})) \\
\llbracket Body_{start} \rrbracket(l, r, \widetilde{x}, \widetilde{k}) &= (\boldsymbol{\nu} g, l', l_1, l_2) \\
&\qquad (\overline{g} \mid !g.(\llbracket \mathsf{true} \rrbracket(l', \widetilde{x}, \widetilde{k}) \\
&\qquad\qquad \mid (\boldsymbol{\nu} t, f)\, (l'(v).\overline{v}\langle \widetilde{n} \rangle.\overline{n_1}\langle t, f \rangle.(t.\overline{l_1} \mid f.\overline{l_2})) \\
&\qquad\qquad \mid l_1.\llbracket \mathsf{wait}\ 10;\ this!fire() \rrbracket(g, r, \widetilde{x}, \widetilde{k}) \\
&\qquad\qquad \mid l_2.\overline{l})) \\
\llbracket \mathsf{wait}\ 10;\ this!fire() \rrbracket(l, r, \widetilde{x}, \widetilde{k}) &= (\boldsymbol{\nu} l')\, (\llbracket \mathsf{wait}\ 10 \rrbracket(l', r, \widetilde{x}, \widetilde{k}) \mid l'.\llbracket this!fire() \rrbracket(l, r, \widetilde{x}, \widetilde{k})) \\
\llbracket \mathsf{wait}\ 10 \rrbracket(l, r, \widetilde{x}, \widetilde{k}) &= (\boldsymbol{\nu} l')\, (\llbracket 10 \rrbracket(l', \widetilde{x}, \widetilde{k}) \mid l'(n).t[n].\overline{l}) \\
\llbracket 10 \rrbracket(l, \widetilde{x}, \widetilde{k}) &= \overline{l}\langle 10 \rangle \\
\llbracket this!fire() \rrbracket(l, r, \widetilde{x}, \widetilde{k}) &= (\boldsymbol{\nu} l')\, (\llbracket this!fire() \rrbracket(l', \widetilde{x}, \widetilde{k}) \mid l'(v).\overline{l}) \\
\llbracket this!fire() \rrbracket(l, \widetilde{x}, \widetilde{k}) &= (\boldsymbol{\nu} h)\, (\llbracket this \rrbracket(h, \widetilde{x}, \widetilde{k}) \\
&\qquad \mid h(v).(\boldsymbol{\nu} r, l', n)\, (\overline{v}\langle n \rangle.n(\widetilde{m}).\overline{m_{fire}}\langle r, l' \rangle \mid r(v).\overline{l}\langle v \rangle \mid l'.\overline{l})) \\
\llbracket this \rrbracket(l, \widetilde{x}, \widetilde{k}) &= (\boldsymbol{\nu} r, u)\, (\overline{x_{this}}\langle r, u \rangle.r(v).\overline{l}\langle v \rangle) \\
\llbracket Mdec_{fire} \rrbracket(m, \widetilde{x}, \widetilde{k}) &= (\boldsymbol{\nu} g)\, (m(r, l).\overline{g}\langle r, l \rangle \mid g(r, l).\llbracket Body^{i}_{fire} \rrbracket(l, r, \widetilde{x}, \widetilde{k})) \\
\llbracket \mathsf{new}(Timer)!start() \rrbracket(l, \widetilde{x}, \widetilde{k}) &= (\boldsymbol{\nu} h)\, (\llbracket \mathsf{new}(Timer) \rrbracket(h, \widetilde{x}, \widetilde{k}) \\
&\qquad \mid h(v).(\boldsymbol{\nu} r, l', n)\, (\overline{v}\langle n \rangle.n(\widetilde{m}).\overline{m_{start}}\langle r, l' \rangle \mid r(v).\overline{l}\langle v \rangle \mid l'.\overline{l})) \\
\llbracket \mathsf{new}(Timer) \rrbracket(l, \widetilde{x}, \widetilde{k}) &= (\boldsymbol{\nu} l')\, (\overline{k_{Timer}}\langle l' \rangle.l'(c).\overline{l}\langle c \rangle)
\end{aligned}
$$

図 10 タイマのプログラムの π 計算による記述
Fig. 10 π-calculus expressions of a timer program.

$$
\begin{aligned}
& (\boldsymbol{\nu} l')\, ((\boldsymbol{\nu} l'')\, (\overline{l''}\langle 10 \rangle \mid l''(n).t[n].\overline{l'}) \mid l'.(\boldsymbol{\nu} l'')\, (P \mid l''(v).\overline{l})) \\
\xrightarrow{\tau}\ & (\boldsymbol{\nu} l')\, (t[10].\overline{l'} \mid l'.(\boldsymbol{\nu} l'')\, (P \mid l''(v).\overline{l})) \\
\rightsquigarrow^{10}\ & (\boldsymbol{\nu} l')\, (t[0].\overline{l'} \mid l'.(\boldsymbol{\nu} l'')\, (P \mid l''(v).\overline{l})) \\
\xrightarrow{\tau}\ & (\boldsymbol{\nu} l'')\, (P \mid l''(v).\overline{l}) \\
\xrightarrow{\alpha}{}^{3}\ & (\boldsymbol{\nu} l'')\, ((\boldsymbol{\nu} r, l', n)\, (\overline{v}\langle n \rangle.n(\widetilde{m}).\overline{m_{fire}}\langle r, l' \rangle \mid r(v).\overline{l''}\langle v \rangle \mid l'.\overline{l''}) \mid l''(v).\overline{l}) \\
\xrightarrow{\alpha}{}^{2}\ & (\boldsymbol{\nu} l'')\, ((\boldsymbol{\nu} r, l')\, (\overline{m_{fire}}\langle r, l' \rangle \mid r(v).\overline{l''}\langle v \rangle \mid l'.\overline{l''}) \mid l''(v).\overline{l}) \\
\xrightarrow{\overline{m_{fire}}}\ & (\boldsymbol{\nu} l'')\, ((\boldsymbol{\nu} r, l')\, (r(v).\overline{l''}\langle v \rangle \mid l'.\overline{l''}) \mid l''(v).\overline{l})
\end{aligned}
$$

ここで $P = (\boldsymbol{\nu} h)\, ((\boldsymbol{\nu} r, u)\, (\overline{x_{this}}\langle r, u \rangle.r(v).\overline{h}\langle v \rangle) \mid h(v).(\boldsymbol{\nu} r, l', n)\, (\overline{v}\langle n \rangle.n(\widetilde{m}).\overline{m_{fire}}\langle r, l' \rangle \mid r(v).\overline{l''}\langle v \rangle \mid l'.\overline{l''}))$

図 11 $\llbracket \mathsf{wait}\ 10; this!fire() \rrbracket$ の遷移
Fig. 11 The transition of $\llbracket \mathsf{wait}\ 10; this!fire() \rrbracket$.

5. 関 連 研 究

Berger らはメッセージ消失，タイマ，プロセスの実行失敗などを導入して拡張した π 計算を提案し，二相コミットプロトコルを記述している[6]．タイマは $\mathsf{timer}^t(Q, R)$ と表されるプロセスとして実現されている．動作は時間ステップ関数と動作意味定義によって定義される．時間の経過は入出力による遷移や τ 遷移にともなって発生し，時間ステップ関数 ϕ の適用として表現する．

この拡張では，時間の経過を特別な遷移として表現するのではなく，従来の入出力や τ による 1 ステップの遷移により 1 単位時間が経過するとしている．そのためこの拡張を利用した OOL_{RT} に対する変換を考え

ると，変数参照やメソッドの呼び出し，真偽値の評価などあらゆる動作において，動作の進行にともなって発生する遷移の数に応じた長さの時間が経過することになる．この体系では，動作に対して抽象的な同一の時間経過を仮定することになり，最大同期性（Maximal Synchrony）を仮定していない．

本論文における時間拡張はプロセス代数で行われている時間拡張に基づいている[5),12),13)]．本論文で提案する体系では，プロセス代数における基本的な時間的性質[4)]の，時間決定性，最大進行性を満たす．

6. おわりに

本論文では実時間システムの開発に適用するために π 計算を時間に関して拡張し，リアルタイムオブジェクト指向言語に対して時間拡張した π 計算による記述への変換規則を定義した．本論文の変換規則を用いて実時間システムのソフトウェアプログラムを形式的表現に変換する．この表現と設計段階で与えられた動作記述の遷移列を比較して等価性を調べることにより，実装が設計に従っているか確認することができる．本論文では OOL_{RT} に対し変換規則を与えた．しかし，この変換規則は OOL_{RT} に本質的に依存しているわけではない．クラス定義，インスタンス生成，メソッド呼び出し，制御構造など同等の記述能力を持つオブジェクト指向言語に対する一般的な規則となっている．

本論文における拡張はプロセス代数に対する時間拡張と同様に非常に細かい意味論を提供する．そのため，複雑なシステムにおいてプログラムから導かれる状態数が多くなる．検証対象となるシステムをモデル化するために必要な抽象化技法については今後の課題である．

システム開発では設計と実装における動作の等価性だけではなく，モデル検査[2)]などの枠組みを用いて動作の性質を検証することで，システムの動作が仕様に従っているか確認する必要がある．型理論[1)]に基づく動作の性質の具体的な検証手法については今後の課題である．

謝辞 本研究を進めるにあたり熱心に議論していただいた阿草研究室の皆さんに感謝いたします．本研究の一部は，文部科学省科学技術研究費基盤研究（C）（2）課題番号 13680408，基盤研究（B）（2）課題番号 14380141，文部科学省リーディングプロジェクト e-Society「高信頼 WebWare 生成技術」，文部科学省 21 世紀 COE プログラム「社会情報基盤のための音声・映像の知的統合」の助成による．

参考文献

1) Cardelli, L.: Type Systems, *The Computer Science and Engineering Handbook*, Tucker, A.B.(Ed.), CRC Press (1997).
2) Clarke, E.M., Grunberg, O. and Peled, D.: *Model Checking*, The MIT Press (1999).
3) Gomma, H.: *Designing Concurrent, Distributed, and Real-Time Applications with UML*, Addison Wesley (2000).
4) Ulidowski, I. and Yuen, S.: Process languages for rooted eager bisimulation, *CONCUR 2000*, Lecture Notes in Computer Science, Vol.1877, pp.275–289 (2000).
5) Baeten, J.C.M. and Middelburg, C.A.: Process algebra with timing: real time and discrete time, *Handbook of Process Algebra*, chapter 10, North-Holland, pp.627–684 (2000).
6) Berger, M. and Honda, K.: The Two-Phase Commitment Protocol in an Extended π-Calculus, *Preliminary Proceedings of EXPRESS '00*, pp.105–130 (2000).
7) Hennessy, M.: Timed Process Algebras: A Tutorial, *Proc. International Summer School on Process Design Calculi*, Martoberdorf (1992).
8) Milner, R.: *Communicating and Mobile Systems: the π-Calculus*, Cambridge University Press (1999).
9) Milner, R., Parrow, J. and Walker, D.: A Calculus of mobile processes, Part I/II, *Information and Computation*, Vol.100, pp.1–77 (1992).
10) Sangiorgi, D. and Walker, D.: *The π-calculus: A Theory of Mobile Processes*, Cambridge University Press (2001).
11) Walker, D.: Objects in the π-Calculus, *Information and Computation*, Vol.116, No.2, pp.253–271 (1995).
12) Nicollin, X. and Shifaks, J.: An overview and synthesis on timed process algebras, *Real-time: Theory in Practice, REX Workshop*, Lecture Notes in Computer Science, Vol.600, pp.526–548, Springer (1991).
13) Yi, W.: A Calculus of Real Time Systems, Ph.D. Thesis, Chalmers University of Technology (1991).

(平成 15 年 10 月 14 日受付)
(平成 16 年 4 月 5 日採録)

桑原 寛明（学生会員）

1979 年生．2001 年名古屋大学工学部電気電子情報工学科卒業．2003 年同大学大学院工学研究科計算理工学専攻博士前期課程修了．現在，同大学院情報科学研究科情報システム学専攻博士後期課程在学中．プロセス代数，形式的手法を用いた実時間システム開発に関する研究に興味を持つ．

結縁 祥治

平成 2 年名古屋大学大学院博士課程満了．名古屋大学大学院工学研究科助手，平成 10 年同情報メディア教育センター助教授を経て，現在，同大学大学院情報科学研究科助教授．博士（工学）．並行計算モデルのソフトウェアへの応用面の観点で，通信プロセスモデルの研究に従事．形式的な手法によるモデル化に基づくソフトウェア検証に興味を持つ．

阿草 清滋（正会員）

1947 年生．1970 年京都大学工学部電気工学第二学科卒業．1972 年同大学大学院工学研究科電気工学第二専攻修士課程修了．同博士課程へ進学．1974 年より同情報工学科助手．同講師，助教授を経て 1989 年より名古屋大学教授．現在，同大学大学院情報科学研究科教授．工学博士．専門分野はソフトウェア工学，ソフトウェア開発方法論，知的開発環境，ソフトウェアデータベース，仕様化技法，再利用技法，マンマシンインタフェース．電子情報通信学会，ソフトウェア科学会，IEEE，ACM 各会員．