通信プロセスモデルに対する時間拡張と
実時間ソフトウェア開発への応用

楽原　寛明

概要

本論文では，実時間システムの振舞い解析と検証の基礎となる形式モデルを与えることを目的として，通信プロセスモデルであるπ計算に時間の概念を導入して拡張した体系を提案する．また，その体系に基づいて

- 振舞いの等価性を表す関係
- 振舞いの時間的タイミングの違いを表す関係
- タイムアウトに着目した時間待ち動作の抽象化手法
- 実時間システムを構成するソフトウェアのモデル化手法

を提案する．

実時間システムは動作に時間制約が存在する並行システムであり，与えられた時間制約を満たしながら動作することが要求される．実時間システムの動作の正しさは，計算結果の正しさだけでなく，結果を得るまでに経過した時間の長さが時間制約を満たしているか否かに依存する．実時間システムの動作は，実時間性と並行性により非決定的である．そのため，いつでも正常に動作することを確認することは容易ではない．例えば，動作テストによってすべての動作をテストすることは，動作の非決定性により簡単ではない．また，実時間性により時間の経過を考慮したテストを行う必要があり，行うべきテストの数が非常に多くなる．

実時間システムは携帯電話，交通管理システム，自動車のブレーキングシステムなど多岐に渡る分野で広く利用されている．実時間システムの多くは機器に組み込まれた組込みシステムであり，停止することなく動作することが要求される．また，ブレーキングシステムやペースメーカのように誤動作が人命に重大な影響を及ぼすシステムもあり，高い信頼性が要求される．そのため，動作テストを行うだけではなく，コストが大きくなっても形式的な検証手法によりシステムの動作の正しさを網羅的に示し保証することが必要である．

並行システムを対象とする形式手法として通信プロセスモデルがよく知られている．通信プロセスモデルは並行計算のモデルであり，計算を入出力ではなく外

部環境との相互作用や通信に着目してモデル化する．通信に着目することで入出力に基づくモデルよりも動作を詳細にモデル化することが可能であるのと同時に，相互にデータを送受信しながら全体としての計算を進める並行システムのモデルに適している．Milner による CCS，Hoare による CSP，Milner, Parrow, Walker による π 計算といった通信プロセスモデルが知られている．

実時間システムは並行システムでもあるため，通信プロセスモデルによってモデル化することが可能である．しかし，従来の通信プロセスモデルは時間の概念を扱う仕組みを持たないため，時間に関する動作や時間の長さを表現できない．そこで本論文では，実時間システムの動作を時間に関する動作も含めて精密にモデル化するために，通信プロセスモデルに時間を導入する拡張を提案する．特に，通信プロセスモデルの中でも高い表現能力を持つ π 計算を対象に拡張を行う．センサやアクチュエータ，制御ユニットをオブジェクトとみなすと実時間システムはオブジェクト指向システムと考えられ，π 計算にはオブジェクトのモデル化に適した特徴が存在する．

時間拡張された π 計算による実時間システムのモデルを用いて振舞い解析を行うために，振舞いの等価性を表す関係と，振舞いのタイミングの違いを表す関係を定義する．振舞いの等価性を表す関係は，従来の π 計算における双模倣関係に基づき時間に関する動作の等価性を表現できるよう拡張した関係である．振舞いのタイミングの違いを表す関係も双模倣関係に基づいているが，時間の待ち方が異なり一方が他方よりも常に早く動作する場合も関係付けられる．システム全体を対象とする振舞い解析はコストが高いため，システムを分割し部分ごとに解析できることが望ましい．そのためには関係が合同的性質を持つことが重要である．本論文で提案する 2 種類の関係はいずれも前提条件のない合同的性質は持たないが，合同的性質が成立するための十分条件を示す．

本論文で提案する π 計算の時間拡張は時間に関して非常に細かい意味論を提供する．そのため，実時間システムの時間に関する動作を詳細に調べることが可能であり，デッドラインなどの時間制約に関する性質の解析に有用である．しかし，細かい意味論は詳細な解析を可能にする一方で，状態を細かく分割しているため解析のコストを増大させる．時間に関する動作はタイムアウトによってモデル化できることが知られており，時間に関する動作の特徴としてタイムアウトの発生を残しながら動作を抽象化する手法を提案する．抽象化を行うことにより状態数を削減することができる．抽象化を行って振舞いが等価であれば，抽象化する前のモデルも時間の関係しない動作とタイムアウトについては振舞いが等価である

ことを示す．このことにより，時間の関係しない性質を示すためには抽象化を行ってから示せばよい．提案する抽象化は，実時間システムの動作をタイムアウトを捉えられる形で従来のπ計算によってモデル化することを可能にし，従来のπ計算に対する等価性判定手法，検証手法およびツールの応用を可能にする．

実時間システムの動作を最も詳細かつ直接的に表現しているのはシステムを構成するソフトウェアである．そこで，実時間システムの動作を時間拡張されたπ計算によってモデル化する一つの手法として，プログラムから時間拡張されたπ計算による表現へ変換する規則を与える．時間に関する動作を直接的に記述するための構文を持つリアルタイムオブジェクト指向言語の意味論を変換規則の集合として定義する．変換規則によりプログラムを変換することで，実時間システムの動作を表現した形式的記述が得られる．実時間システム開発の設計段階においてシステムの動作を時間拡張されたπ計算を用いて記述されていれば，実装したプログラムを変換して得られる記述と振舞いを比較することで正しく実装が行われたか確認することができる．

目 次

図 目 次

第1章　序論

1.1　背景

今日，様々な実時間システムが開発され利用されている．携帯電話，交通管理システム，乗り物の制御システム，カーナビゲーションシステム，ペースメーカなど，実時間システムは多岐に渡る分野において活用されている．多くの実時間システムは機器に組み込まれて動作する組込みシステムであり，停止することなく動作することが要求される．また，自動車のブレーキングシステムやペースメーカのように人命にかかわるため誤動作が絶対に許されないシステムもあり，高い信頼性が要求される．

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 [Gom00]．システムの動作の正しさは，計算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が時間制約を満たしたか否かに依存する．例えば，カーナビゲーションシステムは GPS(Global Positioning System) により取得した位置情報に基づいて進行方向，目的地までの距離や経路などを計算してモニタに表示する．しかし，走行中の車は自車位置が刻々と変化するため，一定時間内に計算を完了しなければカーナビゲーションシステムが提供する情報と実際の状況の誤差が大きくなり有用性が失われてしまう．

多くの場合，実時間システムは外部環境と相互作用するために複数のセンサやアクチュエータ，制御ユニットを備えた組込みシステム，あるいは様々なサーバとクライアントが組み合わされて動作するネットワークシステムである．そのため，システムを制御するソフトウェアは時間制約を持つ複数のコンポーネントが組み合わされ，各コンポーネントが状態を持ちながら並行に動作するプログラムとして構成される．それぞれのコンポーネントの動作は外部環境や他のコンポーネントの影響を受け，さらに動作に時間制約が存在する．そのため，本質的に実時間的かつ並行的な実時間システムが正常に動作することを保証するためには，システムの動作を時間を含めて詳細に記述し解析できる基礎技法が必要である．組込み型の実時間システムは長時間にわたって連続的に正常に動作し続けることが要

求される．高い信頼性を確保するためにも形式的な厳密さを導入することが求められる．

近年，実時間システムの開発には多くの場合オブジェクト指向開発手法が適用されている [Dou99]．オブジェクト指向開発には，ソフトウェアの部品化による再利用性の向上や，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存在する．また，実時間システムは相互作用を行う複数のコンポーネントが組み合わされた並行システムであり，オブジェクト指向は本質的に並行的であるので，オブジェクト指向開発との親和性が高い．オブジェクト指向開発ではオブジェクトを基本要素としてシステムを設計し，オブジェクト指向言語を用いて実装を行う．

オブジェクト指向言語はプログラムをオブジェクトの集合として構成するための機構を備えたプログラミング言語である．オブジェクトの定義と生成，オブジェクト間の相互作用などを記述するための構文と意味論が定義されている．また，実時間システム向けのオブジェクト指向言語として Real-Time Java[Wel04] などがある．

システムの振舞いはシステムを構成するソフトウェアプログラムが最も詳細に表現していると考えられる．そのためプログラムに基づいて形式化を行うことで，システムの振舞いの詳細なモデルを得ることができる．しかし，ほとんどのオブジェクト指向言語の意味論は自然言語で与えられており，形式性，厳密性に欠ける．システムの振舞いを正確にモデル化し解析するためには，オブジェクト指向言語の意味論を形式手法を用いて厳密に定義する必要がある．

1.2　目的

本論文では，実時間システムの振舞い解析と検証の基礎となる形式モデルを与えることを目指して，通信プロセスモデルである π 計算に時間の概念を導入して拡張した体系を提案する．さらに，その体系に基づいて

- 振舞いの等価性を表す等価関係
- 振舞いの時間的タイミングの違いを表す擬順序関係
- タイムアウトに着目した時間待ち動作の抽象化手法
- 実時間システムを構成するソフトウェアのモデル化手法

を提案する．本論文では，時間制約を必ず満たさなければならず要求が明確なハード実時間システム [But97] を主な対象とし，そのモデル化と振舞い解析や検証のための基礎技法の確立を目指す．

通信プロセスモデルは並行計算のモデルである．計算を入出力ではなく外部環境との相互作用や通信に着目してモデル化する．通信プロセスモデルでは計算を行うオブジェクトをプロセスと呼び，相互に通信を行いながら並行に動作する複数のプロセスが集まって一つのプロセスを構成する．通信を計算の基本要素とすることで入出力に基づくモデルよりも動作を詳細にモデル化することが可能であり，かつ，相互にデータを送受信しながら全体としての計算が行われる並行計算を自然に表現できる．また，通信プロセスモデルに基づいてどのような計算が行われるか解析する手法や，正しく計算が行われているか検証する手法が知られている．

実時間システムは並行システムでもあるため，通信プロセスモデルによってモデル化することが可能である．しかし，従来の通信プロセスモデルは時間の概念を扱う仕組みを持たないため，時間に関する動作や時間の長さを表現することができない．そこで本論文では，実時間システムの動作を時間に関する動作も含めて精密にモデル化するために，通信プロセスモデルに時間の概念を導入する拡張を提案する．特に，表現能力の高い通信プロセスモデルである π 計算を対象に拡張を行う．オブジェクト指向開発ではオブジェクトが基本要素であるが，π 計算にはオブジェクトのモデル化に適した特徴が存在する．

時間拡張された π 計算により，実時間システムをタイムアウトなどの時間に関する動作も含めてモデル化できる．このモデルを用いてシステムの振舞い解析を行うために，プロセス間の関係を 2 種類提案する．一つは振舞いの等価性を表す等価関係であり，時間に関する動作も含めて双方のプロセスの振舞いが等しいことを表す．もう一つは振舞いのタイミングの違いを表す擬順序関係であり，一方のプロセスが他方のプロセスより常に遅いタイミングで同じように振舞うことを表す．擬順序関係は動作のタイミングの違いを表すため，デッドラインに関する仕様の検証に適用できる．システム全体を対象とする振舞い解析はコストが高いため，システムを分割して部分ごとに解析できることが望ましい．そのためには関係が合同的性質を持つことが望ましい．合同的性質が成り立てば，システムの一部のコンポーネントを振舞いが等価な異なるコンポーネントに交換することが可能になり，部分ごとに振舞いを解析することができる．本論文で提案する 2 種類の関係はいずれも前提条件のない合同的性質は持たないが，合同的性質が成り

立つための十分条件を明らかにし，システムをどのように分割することができるか示す．

提案する体系は時間に関して非常に細かい意味論を提供するため，実時間システムの時間に関する動作を詳細に調べることが可能である．デッドラインなどの時間制約に関する性質の解析に有用である．一方，時間によって状態を細かく分割しているため，状態爆発の問題が時間を考慮しないモデルに比べて容易に引き起こされる．この問題に対処するため，時間に関する動作を抽象化する手法を提案する．この手法により，時間を導入して拡張された体系によるモデルを時間の概念を含まない従来の体系によるモデルに変換できる．時間に関する情報を単に捨象するのではなく，即座にタイムアウトが発生する状態，時間がいくらか経過するとタイムアウトが発生する状態，時間経過が発生しない状態を区別して時間経過を抽象化する．これによりタイムアウトの発生順序を考慮しながらモデルの状態数を削減して検証コストを下げるとともに，時間の概念を含まない従来の体系に対する解析手法や検証ツールの応用を可能にする．

また，実時間システムの動作をモデル化する一つの手法として，実時間システムを構成するソフトウェアのモデル化手法を提案する．システムの振舞いはプログラムが最も直接的に表現していると考えられる．そこで，時間に関する動作を直接的に記述するための構文を持つリアルタイムオブジェクト指向言語の意味論を時間拡張された π 計算によって定義する．意味論は言語の各構文要素から時間拡張された π 計算による記述への変換規則の集合として定義される．この変換規則を用いて，実時間システムを制御するソフトウェアの形式モデルを得ることができる．振舞いの等価関係，擬順序関係，時間に関する動作の抽象化を用いて，プログラムの振舞いを解析することが可能になる．

1.3　π 計算

通信プロセスモデルは，通信を計算の基本要素であると考えて並行システムをモデル化し，並行計算を形式的に扱うための基盤を与える．CCS[Mil80, Mil89] や CSP[Hoa85]，π 計算 [MPW92, Mil99, SW01] を初めとして数多くの研究が行われている．特に π 計算は CCS に対してプロセス間の接続関係を動的に変更できるよう拡張が行われた体系であり，動的プロセスを記述できる高い表現能力を持っている．リンクの動的生成や，生成されたリンクをメッセージとするプロセス間通信を表現できるため，例えばオブジェクト指向におけるクラスのインスタンス生

成の表現に適している．また，π 計算によって λ 計算がエンコードできることが示されており [Mil92]，チューリング完全であることが知られている．

本節では提案する体系の基礎である π 計算の構文，動作意味，等価関係についてその概要を述べる．初めに π 計算の構文と動作意味の定義を示し，直観的な説明を加える．π 計算の基本要素は名前である．以下では x, y, z は名前を表す．

定義 1.1 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．

$$\begin{aligned} \pi &::= x(y) \mid \overline{x}\langle z\rangle \mid \tau \\ P &::= M \mid P \mid P \mid \boldsymbol{\nu} x\, P \mid !\, P \\ M &::= \mathbf{0} \mid \pi.P \mid M + M \end{aligned}$$

□

定義 1.2 π 計算の動作意味は図 *1.1* の遷移規則からなるラベル付き遷移系によって定義される．図 *1.1* では *SUM-L* 規則，*COMM-L* 規則，*PAR-L* 規則，*CLOSE-L* 規則においてそれぞれ P と Q を入れ替えた *SUM-R* 規則，*COMM-R* 規則，*PAR-R* 規則，*CLOSE-R* 規則が省略されている．□

これらの定義の直観的な説明を以下に示す．

- （終了）$\mathbf{0}$：プロセスの動作終了状態を表す．これ以上遷移することはない．
- $\pi.P$ は動作 π を実行し，その後プロセス P として振舞う．動作 π の種類に応じて次の 3 通りの場合がある．
  - （出力）$\overline{x}\langle z\rangle.P$：名前 x を通して名前 z を送信する．
  - （入力）$x(y).P$：名前 x を通して受信した名前を y に束縛する．
  - （τ 動作）$\tau.P$：外部から不可視な内部動作によって P に遷移する．
- （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いずれか一方を非決定的に選択する．
- （並行）$P \mid Q$：プロセス P, Q が並行に動作する．
- （制限）$\boldsymbol{\nu} x\, P$：プロセス P 中の自由な名前 x を束縛する．
- （複製）$!P$：無限個のプロセス P が並行合成演算子によって結合されている．

π 計算における等価関係として双模倣関係を挙げる．双模倣関係は次のように定義される．

$$\text{OUT}: \frac{}{\overline{x}\langle z\rangle.P \xrightarrow{\overline{x}\langle z\rangle} P}$$

$$\text{IN}: \frac{}{x(y).P \xrightarrow{x(z)} P\{z/y\}}$$

$$\text{TAU}: \frac{}{\tau.P \xrightarrow{\tau} P}$$

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

$$\text{COMM-L}: \frac{P \xrightarrow{\overline{x}\langle z\rangle} P' \quad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\text{PAR-L}: \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \; z \notin \mathsf{fn}(Q) \text{ if } \alpha = \overline{x}(z)$$

$$\text{CLOSE-L}: \frac{P \xrightarrow{\overline{x}(z)} P' \quad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu} z\,(P' \mid Q')} \; z \notin \mathsf{fn}(Q)$$

$$\text{RES}: \frac{P \xrightarrow{\alpha} P'}{\boldsymbol{\nu} x\, P \xrightarrow{\alpha} \boldsymbol{\nu} x\, P'} \; x \notin \mathsf{n}(\alpha)$$

$$\text{OPEN}: \frac{P \xrightarrow{\overline{x}\langle z\rangle} P'}{\boldsymbol{\nu} z\, P \xrightarrow{\overline{x}(z)} P'} \; z \neq x$$

$$\text{REP-ACT}: \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P}$$

$$\text{REP-COMM}: \frac{P \xrightarrow{\overline{x}\langle z\rangle} P' \quad P \xrightarrow{x(z)} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P}$$

$$\text{REP-CLOSE}: \frac{P \xrightarrow{\overline{x}(z)} P' \quad P \xrightarrow{x(z)} P''}{!P \xrightarrow{\tau} (\boldsymbol{\nu} z\,(P' \mid P'')) \mid !P} \; z \notin \mathsf{fn}(P)$$

図 1.1: π 計算の遷移規則

定義 1.3 以下を満たす関係 $\mathcal{R}$ を双模倣関係と呼ぶ.
$(P, Q) \in \mathcal{R}$ の時,

- $P \xrightarrow{\alpha} P' \ \Rightarrow \ \exists Q'.\ Q \xrightarrow{\alpha} Q' \wedge (P', Q') \in \mathcal{R}$
- $Q \xrightarrow{\alpha} Q' \ \Rightarrow \ \exists P'.\ P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathcal{R}$ □

定義 1.4 $\sim \ = \ \bigcup\{\mathcal{R} \mid \mathcal{R}$ は双模倣関係 $\}$ □

補題 1.1 $\sim$ は等価関係である [SW01]. □

双模倣関係を満たす 2 つのプロセスはいずれのプロセスも他方のプロセスと同じ動作を行い続けることができる．そのため，2 つのプロセスをその動作によって区別することは不可能である．双模倣関係は π 計算における振舞いの等価性の基本的な関係である．例えば，2 つのプロセス $\overline{x}.\mathbf{0} \mid y.\mathbf{0}$ および $\overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0}$ について

$$\begin{aligned}\mathcal{R} \ = \ \{&(\overline{x}.\mathbf{0} \mid y.\mathbf{0},\ \overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0}),\\ &(\mathbf{0} \mid y.\mathbf{0},\ y.\mathbf{0}),\\ &(\overline{x}.\mathbf{0} \mid \mathbf{0},\ \overline{x}.\mathbf{0}),\\ &(\mathbf{0} \mid \mathbf{0},\ \mathbf{0})\}\end{aligned}$$

なる関係 $\mathcal{R}$ が双模倣関係であることから

$$\overline{x}.\mathbf{0} \mid y.\mathbf{0} \ \sim \ \overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0}$$

であることがわかる．いずれのプロセスも x から出力して次に y から入力するか，あるいは y から入力して次に x から出力するか，いずれかの動作が実行できて，かつ，実行可能な動作はこの 2 つのみである．

ここで，双方のプロセスに対し，先行する動作として入力動作 $z(y)$ を付け加えると

$$z(y).(\overline{x}.\mathbf{0} \mid y.\mathbf{0}) \ \nsim \ z(y).(\overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0})$$

である．なぜなら，

$$z(y).(\overline{x}.\mathbf{0} \mid y.\mathbf{0}) \xrightarrow{z(x)} \overline{x}.\mathbf{0} \mid x.\mathbf{0} \xrightarrow{\tau} \mathbf{0} \mid \mathbf{0}$$

に対して

$$z(y).(\overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0}) \xrightarrow{z(x)} \overline{x}.x.\mathbf{0} + x.\overline{x}.\mathbf{0} \xrightarrow{\tau}\!\!\!\!\!\!\rightarrow$$

であり，初めに動作 $z(x)$ によって遷移するとその後の振舞いが模倣できないからである．

定理 1.1 $\sim$ は入力プレフィックス以外の演算子に対して合同である [SW01]. □

1.4　本論文の構成

本論文では，第 2 章で π 計算に対して時間の概念を導入して拡張を行う．拡張された構文，動作意味を定義し，動作中に不正な状態に移らないことを保証するために型システムを与える．時間に関する基本的な性質として，時間決定性，最大進行性，可能動作不変性が成り立つことを示す．

第 3 章では，時間拡張された π 計算によるモデルの振舞い解析技法として，振舞いの等価性を表す時間付き双模倣関係と，動作の実行タイミングの違いを表す遅延時間順関係を定義し，直観的説明を与える．これらの関係が合同的性質を持つための十分条件を示す．

第 4 章では，時間拡張された π 計算によるモデルを時間に関して抽象化する手法を提案する．第 3 章で述べる双模倣関係よりも詳細な双模倣関係のもとで展開定理を示し，時間に関する動作を抽象化する手法を述べる．また，抽象化を行っても時間以外の動作には影響しないことを示す．

第 5 章では，実時間システムのモデル化手法として，リアルタイムオブジェクト指向言語 OOL_{RT} の意味論を時間拡張された π 計算によって与える．OOL_{RT} の構文要素に対して時間拡張された π 計算による記述を定義する．小規模なプログラムに対する変換の例を示す．

最後に，第 6 章で本論文をまとめ，今後の課題を述べる．

なお，第 2，3 章の内容は [KYA03a, KYA04b, KYA05, KYA06] で，第 4 章の内容は [KYA07] で，第 5 章の内容は [KYA03b, KYA04a] で公表済みである．

第2章　π計算に対する時間拡張

2.1　はじめに

π計算 [MPW92, Mil99, SW01] は並行システムの抽象計算モデルの一種である．並行に動作するプロセス間の通信に利用されるリンク自体をメッセージとする通信が表現可能であり，高い表現能力を持つ．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.「ボタンが押されたらメッセージを表示する」といった動作の順序は記述できるが,「ボタンが押されてから10秒後にメッセージを表示する」のような2つの動作間で経過する時間の長さを記述することは不可能である．実時間システムの振舞いを直接的にモデル化するためには時間に依存する振舞いがモデル化できる必要があり，何らかの動作が「いつ」発生するかを記述できる能力が求められる．

実時間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．また，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．

本章では，離散時間の経過を表すためのプリミティブを導入して拡張されたπ計算を提案する．本拡張では，時間の経過も一種の動作とみなし入出力動作と同様にプレフィックスとして記述する．導入したプレフィックスと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 [Hen92]，この拡張により時間に依存する振舞いが表現可能になる．本論文では拡張したπ計算を時間付きπ計算と呼ぶ．

本章の構成は以下の通りである．2.2節で時間付きπ計算の構文を，2.3節で動作意味を定義する．2.4節では型システムについて述べる．時間付きπ計算では通信のメッセージとして，プロセス間のリンクを表すチャネル名と時間待ちの長さを表す自然数に対応する名前がある．通信を行っても常に時間経過を表すプレフィックスと時間待ちの長さを表す自然数が関連付けられることを型システムを用いて保証する．2.5節で時間付きπ計算の時間的性質を示す．最後に関連研究を挙げて

まとめを述べる．

2.2　構文

π 計算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．m 単位時間の長さの時間待ちを $t[\underline{m}]$ と記述し，時間経過動作と呼ぶ．ここで t は以下に示す名前には含まれない特別なシンボルとする．

$\mathcal{N}ame$ を名前の集合，$\mathcal{I}$ を自然数を表す名前の集合，$\mathbb{N}$ を自然数の集合とし，$\mathcal{N} = \mathcal{N}ame - \mathcal{I}$ とする．ここで，$m \in \mathcal{I}$ に対し $i_m \in \mathbb{N}$ が存在して m は $\underline{i_m}$ と書ける．$\underline{i_m}$ は i_m 単位時間を表し，$i_m > 0$ に対して $\underline{i_m - 1}$ は $\underline{i_m}$ より一単位時間短い時間長を表す．以下では $n, x, y, z \in \mathcal{N}ame$ とする．

定義 2.1 時間付き 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．

$$\begin{array}{rcl} \pi & ::= & x(y) \mid \overline{x}\langle z\rangle \mid \tau \mid t[n] \\ P & ::= & M \mid P \,|\, P \mid \boldsymbol{\nu} x\, P \mid !\, P \\ M & ::= & \mathbf{0} \mid \pi.P \mid M + M \end{array}$$ □

時間付き π 計算のプロセス全体の集合を $\mathcal{P}$ と書く．

定義 2.2 プロセス $x(y).P$ における名前 y および $\boldsymbol{\nu} x\, P$ における名前 x のスコープは P に制限される．この時，名前 y および x は束縛されるという．束縛されない名前を自由であるという．プロセス P に含まれる自由な名前の集合を $\mathsf{fn}(P)$，束縛される名前の集合を $\mathsf{bn}(P)$ と書く．$\mathsf{n}(\alpha)$ を動作 α に出現する名前の集合とする． □

時間経過動作 $t[n]$ の n も入力プレフィックス $x(n)$ や制限演算 $\boldsymbol{\nu} n$ によって束縛される．次に名前に対する代入を定義する．

定義 2.3 代入は $\mathcal{N}ame$ から $\mathcal{N}ame$ への関数である．プロセス P に代入 σ を適用

して得られるプロセス $P\sigma$ を以下のように定義する.

$$\begin{aligned}
\mathbf{0}\sigma &\stackrel{def}{=} \mathbf{0} \\
(x(y).P)\sigma &\stackrel{def}{=} x\sigma(y).P\sigma_{-\{y\}} \\
(\overline{x}\langle z\rangle.P)\sigma &\stackrel{def}{=} \overline{x\sigma}\langle z\sigma\rangle.P\sigma \\
(\tau.P)\sigma &\stackrel{def}{=} \tau.P\sigma \\
(t[n].P)\sigma &\stackrel{def}{=} t[n\sigma].P\sigma \\
(P \mid Q)\sigma &\stackrel{def}{=} P\sigma \mid Q\sigma \\
(P + Q)\sigma &\stackrel{def}{=} P\sigma + Q\sigma \\
(\boldsymbol{\nu} x\, P)\sigma &\stackrel{def}{=} \boldsymbol{\nu} x\, P\sigma_{-\{x\}} \\
(!\, P)\sigma &\stackrel{def}{=} !\, P\sigma
\end{aligned}$$

ここで，$x\sigma$ は名前 x に σ を適用して得られる名前を表し，σ_{-D} は D に含まれる名前については置換を行わず，その他の名前については σ と同じ置換を行う代入を表す. □

代入 σ は $\{y_1, \ldots, y_n/x_1, \ldots, x_n\}$ とも書く．これは x_i に対する y_i の代入を表す.

定義 2.4 構造合同性は α 変換による関係 $\equiv_\alpha$[1]と以下の公理を満たす最小の合同関係であり $\equiv$ と書く.

$$\begin{aligned}
M_1 + (M_2 + M_3) &\equiv (M_1 + M_2) + M_3 \\
M_1 + M_2 &\equiv M_2 + M_1 \\
M_1 + \mathbf{0} &\equiv M_1 \\
P_1 \mid (P_2 \mid P_3) &\equiv (P_1 \mid P_2) \mid P_3 \\
P_1 \mid P_2 &\equiv P_2 \mid P_1 \\
P_1 \mid \mathbf{0} &\equiv P_1 \\
\boldsymbol{\nu} z\, \boldsymbol{\nu} w\, P &\equiv \boldsymbol{\nu} w\, \boldsymbol{\nu} z\, P \\
\boldsymbol{\nu} z\, \mathbf{0} &\equiv \mathbf{0} \\
\boldsymbol{\nu} z\, (P \mid Q) &\equiv P \mid \boldsymbol{\nu} z\, Q \quad z \in \mathcal{I} \vee z \notin \mathsf{fn}(P) \\
!\, P &\equiv P \mid !\, P
\end{aligned}$$

□

[1] P に対し α 変換のみを適用することによって Q が得られるならば $P \equiv_\alpha Q$

時間経過動作 $t[\underline{m}]$ は直観的には m 単位時間の時間待ちを表す．プロセス $t[\underline{m}].P$ は m 単位時間後にタイムアウト遷移を行い，その後 P として振舞う．その他のプレフィックスと演算子の直観的な意味は 1.3 節に示した従来の π 計算における意味と同様である．

2.3　動作意味

時間付き π 計算の動作意味はラベル付き遷移系によって定義される．ラベルとして以下に定義する動作を利用する．

定義 2.5 動作 β は以下の構文によって定義される．

$$\beta \ ::= \ x(y) \mid \overline{x}\langle z\rangle \mid \overline{x}(z) \mid \tau$$

ここで，$\overline{x}(z)$ は $(\boldsymbol{\nu} z)\,\overline{x}\langle z\rangle$ の略記であり，新しい名前 z を x を通して出力する動作を表す．動作全体の集合を Act と書く． □

定義 2.6 $\mathcal{P}$ 上の遷移関係 $\{\stackrel{\alpha}{\rightharpoonup} \mid \alpha \in Act \cup \{\bullet\}\} \cup \{\rightharpoonup\}$ は図 *2.1*，図 *2.2* の遷移規則および図 *2.3* の時間経過規則によって定義される．ここで $\bullet$ は Act に含まれない特別なシンボルである．図 *2.2* では *T-SUM-L* 規則，*SUM-L* 規則，*T-PAR-L* 規則，*PAR-L* 規則，*COMM-L* 規則，*CLOSE-L* 規則においてそれぞれ P と Q を入れ替えた *T-SUM-R* 規則，*SUM-R* 規則，*T-PAR-R* 規則，*PAR-R* 規則，*COMM-R* 規則，*CLOSE-R* 規則が省略されている． □

RES 規則，REP-ACT 規則においては $\alpha = \bullet$ の場合も含む．

$\alpha \neq \bullet$ の場合，$P \stackrel{\alpha}{\rightharpoonup} P'$ は入力，出力，内部動作のいずれかの動作 α により P から P' に遷移することを表す．$\overline{x}\langle z\rangle.P$ は名前 x を通して名前 z を出力する．一方，$x(y).P$ は x を通して受信した名前を P 中の自由な名前 y に束縛する．選択演算子によって結合されたプロセスは，入力，出力，内部動作によっていずれか一方が非決定的に選択され他方は捨てられる．並行合成されたプロセスは一方のプロセスのみが遷移するか，あるいは，同じ名前を通して出力を行うプロセスと入力を行うプロセスが存在する場合はそれらのプロセスの間で通信が行われる．通信は τ 遷移によって表される．CLOSE-L, CLOSE-R 規則は束縛された名前をメッセージとした通信を行うと受信側のプロセスも束縛名のスコープに含まれるようになることを表している．

$$\text{OUT}: \frac{}{\overline{x}\langle z\rangle.P \xrightarrow{\overline{x}\langle z\rangle} P}$$

$$\text{IN}: \frac{}{x(y).P \xrightarrow{x(z)} P\{z/y\}}$$

$$\text{TAU}: \frac{}{\tau.P \xrightarrow{\tau} P}$$

$$\text{TIMEOUT}: \frac{}{t[\underline{0}].P \xrightarrow{\bullet} P}$$

$$\text{RES}: \frac{P \xrightarrow{\alpha} P'}{\boldsymbol{\nu}x\, P \xrightarrow{\alpha} \boldsymbol{\nu}x\, P'}\; x \notin \mathsf{n}(\alpha)$$

$$\text{OPEN}: \frac{P \xrightarrow{\overline{x}\langle z\rangle} P'}{\boldsymbol{\nu}z\, P \xrightarrow{\overline{x}(z)} P'}\; z \neq x$$

$$\text{REP-ACT}: \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P}$$

$$\text{REP-COMM}: \frac{P \xrightarrow{\overline{x}\langle z\rangle} P' \quad P \xrightarrow{x(z)} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P}$$

$$\text{REP-CLOSE}: \frac{P \xrightarrow{\overline{x}(z)} P' \quad P \xrightarrow{x(z)} P''}{!P \xrightarrow{\tau} (\boldsymbol{\nu}z\,(P' \mid P'')) \mid !P}\; z \notin \mathsf{fn}(P)$$

図 2.1: 遷移規則

$$\text{T-SUM-L}: \frac{P \overset{\bullet}{\rightharpoonup} P'}{P + Q \overset{\bullet}{\rightharpoonup} P'}$$

$$\text{SUM-L}: \frac{P \overset{\alpha}{\rightharpoonup} P' \quad Q \not\overset{\bullet}{\rightharpoonup}}{P + Q \overset{\alpha}{\rightharpoonup} P'} \alpha \neq \bullet$$

$$\text{T-PAR-L}: \frac{P \overset{\bullet}{\rightharpoonup} P'}{P \mid Q \overset{\bullet}{\rightharpoonup} P' \mid Q}$$

$$\text{PAR-L}: \frac{P \overset{\alpha}{\rightharpoonup} P' \quad Q \not\overset{\bullet}{\rightharpoonup}}{P \mid Q \overset{\alpha}{\rightharpoonup} P' \mid Q} \alpha \neq \bullet \wedge ((z \in \mathcal{I} \vee z \notin \mathsf{fn}(Q)) \text{ if } \alpha = \overline{x}(z))$$

$$\text{COMM-L}: \frac{P \overset{\overline{x}\langle z \rangle}{\rightharpoonup} P' \quad Q \overset{x(z)}{\rightharpoonup} Q'}{P \mid Q \overset{\tau}{\rightharpoonup} P' \mid Q'}$$

$$\text{CLOSE-L}: \frac{P \overset{\overline{x}(z)}{\rightharpoonup} P' \quad Q \overset{x(z)}{\rightharpoonup} Q'}{P \mid Q \overset{\tau}{\rightharpoonup} \boldsymbol{\nu} z\,(P' \mid Q')} z \in \mathcal{I} \vee z \notin \mathsf{fn}(Q)$$

図 2.2: 遷移規則（続き）

$$\mathrm{PASS_T}: \frac{}{t[\underline{k}].P \rightharpoonup t[\underline{k-1}].P} \text{ if } k > 0$$

$$\mathrm{INACT_T}: \frac{}{\mathbf{0} \rightharpoonup \mathbf{0}}$$

$$\mathrm{OUT_T}: \frac{}{\overline{x}\langle z\rangle.P \rightharpoonup \overline{x}\langle z\rangle.P}$$

$$\mathrm{IN_T}: \frac{}{x(y).P \rightharpoonup x(y).P}$$

$$\mathrm{SUM_T}: \frac{P \rightharpoonup P' \quad Q \rightharpoonup Q'}{P + Q \rightharpoonup P' + Q'}$$

$$\mathrm{PAR_T}: \frac{P \rightharpoonup P' \quad Q \rightharpoonup Q'}{P \mid Q \rightharpoonup P' \mid Q'} \text{ if } P \mid Q \overset{\tau}{\not\rightarrow}$$

$$\mathrm{RES_T}: \frac{P \rightharpoonup P'}{\boldsymbol{\nu} x\, P \rightharpoonup \boldsymbol{\nu} x\, P'}$$

$$\mathrm{REP_T}: \frac{P \rightharpoonup P'}{!P \rightharpoonup !P'} \text{ if } P \mid P \overset{\tau}{\not\rightarrow}$$

図 2.3: 時間経過規則

$P \overset{\bullet}{\rightharpoonup} P'$ はタイムアウトによって P から P' に遷移することを表し，入出力動作や時間経過に優先して実行される．選択演算子 $+$ はタイムアウトによって選択が行われる．例えば，$a.P + t[0].Q \overset{\bullet}{\rightharpoonup} Q$ である．この時，必ずタイムアウトが選択され，動作 a は選択されない．$t[3].P + t[5].Q$ では $t[3].P$ が先にタイムアウトするため必ず $t[3].P$ が選択され，3 単位時間後に P に遷移する．

$P \rightharpoonup P'$ 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．プレフィックスが時間経過動作であればタイムアウトまでの時間が 1 単位時間短くなる．$\mathrm{PAR_T}$ 規則と $\mathrm{REP_T}$ 規則は τ 動作による遷移が時間経過による遷移に優先して発生することを表す．

2.4　型システム

時間付き π 計算では時間経過動作 $t[n]$ の n を入力プレフィックスによって束縛して時間待ちの長さを動的に決定することができる．例えば，プロセス $\overline{x}\langle m\rangle.\mathbf{0} \mid x(n).t[n].\mathbf{0}$ では

$$\overline{x}\langle m\rangle.\mathbf{0} \mid x(n).t[n].\mathbf{0} \quad \overset{\tau}{\rightharpoonup} \quad \mathbf{0} \mid t[m].\mathbf{0}$$

のように通信を行った後で時間待ちの長さが m に決定される．この時 $m \in \mathcal{I}$ であることは構文的には保証されない．もし $m \notin \mathcal{I}$ ならばプロセス $\mathbf{0} \mid t[m].\mathbf{0}$ の動作は未定義である．そのため時間経過動作の添字が $\mathcal{I}$ の要素となることを型システムを用いて保証する．すなわち，型付け可能なプロセスでは時間経過動作の添字は必ず自然数と対応付けられる名前であることが保証される．

Sangiorgi らの simply-typed π-calculus[SW01] を基礎として，自然数を表す名前の型 I を導入する．型の構文を図 2.4 に示す．ここで，B は基本型，$\sharp V$ は型 V の値を送受信する名前の型を表す．L をリンク型と呼ぶ．

名前 x に型 T を割り当てることを $x : T$ と書く．名前に対する型の割当ての集合を型環境と呼び Γ と書く．型環境に含まれる名前はすべて異なるとする．一つの名前に複数の異なる型を割り当てることはできない．名前に対する型判定を $\Gamma \vdash x : T$ と書く．型環境 Γ において名前 x の型が T であるか判定する．型環境 Γ における名前 x の型を一般的に $\Gamma(x)$ と書く．また，$n \in \mathcal{I}$ に対して $\Gamma(n) = I$ とする．プロセスに対する型判定を $\Gamma \vdash P : \diamond$ と書く．P がプロセス式であることが明らかな場合は $\diamond$ を省略して $\Gamma \vdash P$ とも書く．

定義 2.7 型環境 Γ にリンク型のみが含まれる時 Γ は閉じているという．　□

$$
\begin{array}{rcl}
T & ::= & V \mid \diamond \\
V & ::= & B \mid I \mid L \\
L & ::= & \sharp V \\
\Gamma & ::= & \Gamma, x : V \mid \emptyset
\end{array}
$$

図 2.4: 型の構文

型付け規則を図 2.5 に示す．T-TIME 規則により時間経過動作 $t[n]$ の n の型は I でなければならない．型が I の名前は $\mathcal{I}$ に含まれる自然数を表す名前のみである．よって時間経過動作の添字は常に自然数を表す名前であることが保証される．

本節で定義した型システムの基本的な性質を示す．

補題 2.1 $\Gamma \vdash E : T$ ならば，任意の型 S と Γ で型が割り当てられない任意の名前 x に対し $\Gamma, x : S \vdash E : T$. □

補題 2.2 $\Gamma, x : S \vdash E : T$ かつ $x \notin \mathsf{fn}(E)$ ならば $\Gamma \vdash E : T$. □

補題 2.3 $\Gamma \vdash E : T$ かつ $\Gamma(x) = S$ かつ $\Gamma \vdash v : S$ ならば $\Gamma \vdash E\{v/x\} : T$.

証明： $\Gamma \vdash E : T$ の型判定木の高さに関する帰納法による．最後に適用された規則によって場合分けする．

- T-NAT 規則：$\Gamma \vdash n : I$ であり，明らかに $\Gamma(n) = I$．$\Gamma \vdash v : I$ とする．この時，$n\{v/n\} = v$ で仮定から $\Gamma \vdash v : I$ ゆえ $\Gamma \vdash n\{v/n\} : I$ である．

- T-PAR 規則：$E = P \,|\, Q$ とすると T-PAR 規則の定義より $\Gamma \vdash P : \diamond$ かつ $\Gamma \vdash Q : \diamond$ である．$\Gamma(x) = S$，$\Gamma \vdash v : S$ とする．帰納法の仮定より $\Gamma \vdash P\{v/x\} : \diamond$，$\Gamma \vdash Q\{v/x\} : \diamond$ であり，$P\{v/x\} \mid Q\{v/x\} = (P \,|\, Q)\{v/x\}$ ゆえ T-PAR 規則から $\Gamma \vdash (P \,|\, Q)\{v/x\} : \diamond$ である．

- T-RES 規則：$E = \boldsymbol{\nu} y\, P$ とすると T-RES 規則の定義より $\Gamma, y : L \vdash P : \diamond$．$\Gamma(x) = S$，$\Gamma \vdash v : S$ とする．帰納法の仮定より $\Gamma \vdash P\{v/x\} : \diamond$ である．もし $x \neq y$ ならば $\boldsymbol{\nu} y\, P\{v/x\} = (\boldsymbol{\nu} y\, P)\{v/x\}$ ゆえ T-RES 規則から $\Gamma \vdash (\boldsymbol{\nu} y\, P)\{v/x\} : \diamond$ である．一方 $x = y$ ならば $(\boldsymbol{\nu} y\, P)\{v/x\} = \boldsymbol{\nu} y\, P$ ゆえ明らかに $\Gamma \vdash (\boldsymbol{\nu} y\, P)\{v/x\} : \diamond$ である．

$$\text{T-BASE}: \frac{}{\Gamma \vdash val : B}\, val \in B$$

$$\text{T-NAME}: \frac{}{\Gamma,\, x : T \vdash x : T}$$

$$\text{T-NAT}: \frac{}{\Gamma \vdash n : I}\, n \in \mathcal{I}$$

$$\text{T-PAR}: \frac{\Gamma \vdash P : \diamond \quad \Gamma \vdash Q : \diamond}{\Gamma \vdash P \mid Q : \diamond}$$

$$\text{T-SUM}: \frac{\Gamma \vdash P : \diamond \quad \Gamma \vdash Q : \diamond}{\Gamma \vdash P + Q : \diamond}$$

$$\text{T-RES}: \frac{\Gamma, x : L \vdash P : \diamond}{\Gamma \vdash \boldsymbol{\nu} x\, P : \diamond}$$

$$\text{T-REP}: \frac{\Gamma \vdash P : \diamond}{\Gamma \vdash !\, P : \diamond}$$

$$\text{T-TAU}: \frac{\Gamma \vdash P : \diamond}{\Gamma \vdash \tau.P : \diamond}$$

$$\text{T-IN}: \frac{\Gamma \vdash x : \sharp T \quad \Gamma,\, y : T \vdash P : \diamond}{\Gamma \vdash x(y).P : \diamond}$$

$$\text{T-OUT}: \frac{\Gamma \vdash x : \sharp T \quad \Gamma \vdash z : T \quad \Gamma \vdash P : \diamond}{\Gamma \vdash \overline{x}\langle z \rangle.P : \diamond}$$

$$\text{T-TIME}: \frac{\Gamma \vdash n : I \quad \Gamma \vdash P : \diamond}{\Gamma \vdash t[n].P : \diamond}$$

$$\text{T-INACT}: \frac{}{\Gamma \vdash \mathbf{0} : \diamond}$$

図 2.5: 型付け規則

- T-TIME 規則： $E = t[n].P$ とすると T-TIME 規則の定義より $\Gamma \vdash n : I$ かつ $\Gamma \vdash P : \diamond$．$\Gamma(x) = S$，$\Gamma \vdash v : S$ とする．もし $x = n$ ならば $S = I$ であり，帰納法の仮定より $\Gamma \vdash n\{v/n\} : I$ かつ $\Gamma \vdash P\{v/n\} : \diamond$．$t[n\{v/n\}].P\{v/n\} = (t[n].P)\{v/n\}$ ゆえ T-TIME 規則から $\Gamma \vdash (t[n].P)\{v/n\} : \diamond$ である．$x \neq n$ ならば $n\{v/x\} = n$ であり，帰納法の仮定より $\Gamma \vdash P\{v/x\} : \diamond$．よって T-TIME 規則から $\Gamma \vdash (t[n].P)\{v/x\} : \diamond$ である．

- 他の場合も同様である． □

補題 2.1 より型環境には新しい名前の型割当てを自由に追加することができる．補題 2.2 より型判定の対象となっているプロセス式において束縛される名前の型割当てを型環境から除くことができる．補題 2.3 は型が同じ名前の代入を行っても型判定が保存されることを表す．

プロセスは意味論に従って動作する．初期状態で型付け可能なプロセスはその実行過程において常に型付け可能であることが望まれる．次に示す定理は本型システムで型付け可能であれば，動作意味による遷移の前後で型が保存されることを示す．

定理 2.1 Γ は閉じた型環境であり，$\Gamma \vdash P$ とする．

- $P \overset{\alpha}{\rightarrowtriangle} P'$ に対し，

  1. $\alpha = \tau$ あるいは $\alpha = \bullet$ ならば $\Gamma \vdash P'$．
  2. $\alpha = x(y)$ ならば以下を満たす型 T が存在する．
     - $\Gamma \vdash x : \sharp T$
     - $\Gamma \vdash y : T$ ならば $\Gamma \vdash P'$
  3. $\alpha = \overline{x}\langle z \rangle$ ならば以下を満たす型 T が存在する．
     - $\Gamma \vdash x : \sharp T$
     - $\Gamma \vdash z : T$
     - $\Gamma \vdash P'$
  4. $\alpha = \overline{x}(z)$ ならば以下を満たす型 T が存在する．
     - $\Gamma \vdash x : \sharp T$
     - $\Gamma, z : T \vdash P'$
     - T はチャネル型

- $P \rightharpoonup P'$ に対し $\Gamma \vdash P'$.

証明： $P \overset{\alpha}{\rightharpoonup} P'$，$P \rightharpoonup P'$ の導出木の高さに関する帰納法によって示す.

- INPUT 規則：$P = x(y).Q, \alpha = x(z), P' = Q\{z/y\}$
  $\Gamma \vdash P$ ゆえ T-IN 規則より $\Gamma \vdash x : \sharp T$，$\Gamma, y : T \vdash Q$ である．補題 2.1 より $\Gamma, y : T \vdash x : \sharp T$ とできて，$(\Gamma, y : T)(y) = T$ である．ここで $\Gamma \vdash z : T$ とすると $\Gamma, y : T \vdash z : T$ とできて，補題 2.3 より $\Gamma, y : T \vdash Q\{z/y\}$．$y \notin \mathsf{fn}(Q\{z/y\})$ であるので補題 2.2 より $\Gamma \vdash Q\{z/y\}$.

- TIMEOUT 規則：$P = t[\underline{0}].Q, \alpha = \bullet, P' = Q$
  $\Gamma \vdash P$ ゆえ T-TIME 規則より $\Gamma \vdash Q'$.

- COMM-L 規則：$P = Q_1 \,|\, Q_2, Q_1 \overset{x(z)}{\rightharpoonup} Q_1', Q_2 \overset{\overline{x}\langle z\rangle}{\rightharpoonup} Q_2', \alpha = \tau, P' = Q_1' \,|\, Q_2'$
  $\Gamma \vdash P$ ゆえ T-PAR 規則より $\Gamma \vdash Q_1$，$\Gamma \vdash Q_2$ である．帰納法の仮定より $\Gamma \vdash x : \sharp T$ かつ $\Gamma \vdash z : T \Rightarrow \Gamma \vdash Q_1'$ なる型 T が存在する．同じく帰納法の仮定より $\Gamma \vdash x : \sharp S$，$\Gamma \vdash z : S$，$\Gamma \vdash Q_2'$ なる型 S が存在する．ここで，型環境に含まれる名前はすべて異なることから名前の型は一意に定まるため $T = S$ である．よって $\Gamma \vdash z : T$ より $\Gamma \vdash Q_1'$ であり，$\Gamma \vdash Q_2'$ と T-PAR 規則より $\Gamma \vdash Q_1' \,|\, Q_2'$ である.

- OPEN 規則：$P = \boldsymbol{\nu} z\, Q, Q \overset{\overline{x}\langle z\rangle}{\rightharpoonup} Q', \alpha = \overline{x}(z), P' = Q'$
  $\Gamma \vdash P$ ゆえ T-RES 規則よりあるチャネル型 T が存在して $\Gamma, z : T \vdash Q$ である．帰納法の仮定より $\Gamma, z : T \vdash x : \sharp S$，$\Gamma, z : T \vdash z : S$，$\Gamma, z : T \vdash Q'$ なる型 S が存在する．型環境に含まれる名前の型は一意に定まるため $T = S$ である．$x \neq z$ ゆえ補題 2.2 より $\Gamma \vdash x : \sharp T$ である.

- $\mathrm{PASS}_{\mathrm{T}}$ 規則：$P = t[\underline{k}].Q, P' = t[\underline{k-1}].Q, k > 0$
  $\Gamma \vdash P$ ゆえ T-TIME 規則より $\Gamma \vdash Q$ である．$\underline{k-1} \in \mathcal{I}$ より $\Gamma \vdash \underline{k-1} : I$ なので $\Gamma \vdash P'$.

- 他の場合も同様. □

以下，本論文では特に明示しない限り本節で示した型システムで型付け可能なプロセスを対象とする.

2.5 時間的性質

時間付きπ計算が時間決定性 (Time Determinacy)[UY97]，最大進行性 (Maximal Progress)[UY97]，可能動作不変性 (Constancy of Offers)[UY04] を満たすことを示す.

定理 2.2 (**時間決定性**) 任意のプロセス P に対し，$P \rightharpoonup P'$ かつ $P \rightharpoonup P''$ ならば $P = P''$ である.

証明： P の構造に関する帰納法による.

- $P = \mathbf{0},\ \overline{x}\langle z\rangle.P',\ x(y).P$ の時，$P \rightharpoonup P$ ゆえ明らか.
- $P = \tau.P'$ の時，$P \not\rightharpoonup$.
- $P = P_1 + P_2$ の時，$P \rightharpoonup P_1' + P_2'$ かつ $P \rightharpoonup P_1'' + P_2''$ とすると帰納法の仮定より $P_1' = P_2'$ かつ $P_1'' = P_2''$ であるので $P_1' + P_2' = P_1'' + P_2''$.
- 他の場合も同様. □

時間決定性は時間経過による遷移が決定的であり，遷移先が一意に決まることを表す．時間の経過は一通りであることを意味している.

定理 2.3 (**最大進行性**) 任意のプロセス P に対し，$P \overset{\tau}{\rightharpoonup}$ ならば $P \not\rightharpoonup$ である.

証明： $P \overset{\tau}{\rightharpoonup} P'$ の導出木の高さに関する帰納法による．導出の最後に適用された規則によって場合分けする.

- TAU 規則：遷移規則の定義より明らか.
- SUM-L 規則 $(\alpha = \tau)$：$P+Q \overset{\tau}{\rightharpoonup} P'$ とすると規則より $P \overset{\tau}{\rightharpoonup} P'$ ゆえ帰納法の仮定から $P \not\rightharpoonup$ である．この時，$\mathrm{SUM_T}$ 規則の前件を満たさないため $P+Q \not\rightharpoonup$ である.
- COMM-L 規則：$\mathrm{PAR_T}$ 規則の定義より明らか.
- 他の場合も同様. □

最大進行性はτ動作による遷移が可能であれば時間経過が発生しないことを表す．内部動作は不必要に待機させられないことを意味する．また，プロセス間通信は実行可能になればすぐに実行される.

定理 2.4 (**可能動作不変性**) 任意のプロセス P に対し，$P \rightharpoonup P'$ かつ $P' \not\xrightarrow{\bullet}$ ならば $P \xrightarrow{\alpha} \iff P' \xrightarrow{\alpha}$ である．

証明： $P \rightharpoonup P'$ の導出木の高さに関する帰納法による．導出の最後に適用された規則によって場合分けする．

- $\mathrm{PASS_T}$ 規則：$P = t[\underline{k}].P_0$(ただし $k > 0$) であり，$P' = t[\underline{k-1}].P_0 \not\xrightarrow{\bullet}$ であることから $k - 1 > 0$，すなわち $k > 1$ である．この時，遷移規則の定義より明らかに $P \not\xrightarrow{\alpha}$ かつ $P' \not\xrightarrow{\alpha}$ である．
- $\mathrm{INACT_T}$ 規則：$P = P' = \mathbf{0}$ ゆえ明らかに $P \not\xrightarrow{\alpha}$ かつ $P' \not\xrightarrow{\alpha}$ である．
- $\mathrm{OUT_T}$ 規則，$\mathrm{IN_T}$ 規則：$P = P'$ ゆえ明らかに $P \xrightarrow{\alpha}$ ならば $P' \xrightarrow{\alpha}$ かつ $P' \xrightarrow{\alpha}$ ならば $P \xrightarrow{\alpha}$ である．
- $\mathrm{SUM_T}$ 規則：$P = P_1 + P_2, P_1 \rightharpoonup P_1', P_2 \rightharpoonup P_2', P' = P_1' + P_2'$ とする．$P \not\xrightarrow{\bullet}$ ゆえ SUM-L, SUM-R 規則から $P_1' \not\xrightarrow{\bullet}$ かつ $P_2' \not\xrightarrow{\bullet}$ である．ここで，帰納法の仮定から $P_1 \xrightarrow{\alpha} \iff P_1' \xrightarrow{\alpha}$ および $P_2 \xrightarrow{\alpha} \iff P_2' \xrightarrow{\alpha}$ であるため，$P \xrightarrow{\alpha} \iff P' \xrightarrow{\alpha}$ である．
- 他の場合も同様. □

可能動作不変性は，時間経過後にタイムアウトしない場合，時間経過前後で実行できる入出力動作が変化しないことを表す.

2.6　関連研究

時間の概念を導入して拡張された π 計算はいくつか提案されている [BH00, Ber02, Ber04, Che04, DLP96, LŽ02]．[BH00, Ber02, Ber04] では非同期 π 計算 [HT91, Bou92] に対してタイマ，メッセージ消失，プロセスの実行失敗などを導入して拡張した体系による二相コミットプロトコルの記述が示されている．タイマは $\mathsf{timer}^t(P, Q)$ のようにプロセスとして記述され，t 単位時間経過するとタイムアウトして Q として振舞う．動作は時間を進める時間ステップ関数と動作意味定義によって定義されており，時間ステップ関数の適用として時間の経過を表現する．時間経過を表す動作はなく，入出力動作や τ 動作による 1 ステップの遷移で 1 単位時間が経過する．[Che04] では π 計算に対し稠密時間を導入した拡張が提案されて

いる．[DLP96] では，環境として時計を持ち各動作に完了した時刻を記録することで時間動作を表現できるよう拡張している．これらの体系では，時間待ちの長さをプロセス間通信によって直接送受信して動的に決定する仕組みがない．[LŽ02] では離散時間を導入して拡張している．本論文における体系ではタイムアウトの発生が明示されるが，これらの体系ではタイムアウトは暗黙のうちに発生し捉えることはできない．

π 計算の基礎となった CCS[Mil80, Mil89] に対する時間拡張も多く提案されている [HR95, Hen92, ST92, Che92a, Che92b, MT90, Yi91a, Yi91b]．時間領域を離散時間 [HR95, Hen92, ST92, MT90] あるいは稠密時間 [Yi91a, Yi91b] とし，遅延演算子 [MT90, Yi91a, Yi91b] やタイムアウト演算子 [HR95, Hen92, ST92] を導入して CCS の拡張を行っている．[Che92a, Che92b] では入出力動作に対して時間制約の上限と下限を付加する形で拡張している．また時間領域は特に定めていない．

本論文ではこれらの先行研究を踏まえ，実時間システムの振舞いのモデル化を行うために π 計算に対して離散時間を導入し，タイムアウトを明示的に表現できる意味論を定義して拡張を行った．

2.7 おわりに

本章では，実時間システム開発への応用を目的として，π 計算に離散時間の経過を表す時間経過動作を導入して拡張した時間付き π 計算を提案した．時間経過動作と選択演算子を用いることでタイムアウトをモデル化し，時間に依存する動作を表現する．提案する体系は以下に挙げる特徴を持つ．

- 時間待ちの長さをメッセージとした通信
- タイムアウトによる遷移が明示される意味論

これらの特徴はシステムの振舞いの柔軟な記述と詳細な振舞い解析を可能にする．

時間付き π 計算の構文は従来の π 計算の構文に対しプレフィックスとして時間経過動作が追加されたものである．時間経過動作には時間待ちの長さを示す自然数を表す名前が添字として付けられる．この添字は $x(n).t[n].P$ のように入力プレフィックスによって束縛することが可能であり，これにより時間待ちの長さを通信によって実行時に動的に決定することができる．この時，時間経過動作の添字が必ず自然数を表す名前であることを保証するために型システムを定義した．定理

2.1 により，型付け可能なプロセスはその遷移先においても型付け可能である．つまり，型付け可能であれば正常に動作することが保証されている．

動作意味は時間経過に関する規則とタイムアウトに関する規則が追加されたラベル付き遷移系によって定義される．時間付き π 計算の意味論では，通信プロセスモデルに対する従来の時間拡張では明示されず暗黙のうちに発生するタイムアウトを一つの遷移として明示する．システムの振舞いとして入出力動作に加え，タイムアウトの発生と 1 単位時間ごとの時間経過が明示的に表現されるため，振舞いを詳細に解析することができる．

第3章　時間付き双模倣関係および遅延時間順関係

3.1　はじめに

並行計算の動作に関する検証技法として，双模倣関係に基づく並行プロセスの等価関係がよく知られている [Mil89]．双模倣関係を満たすプロセスは，いずれのプロセスも他方のプロセスの動作を模倣することが可能であり，双方のプロセスの動作は等価であると考えることができる．本章では，π 計算における双模倣関係 [MPW92, Mil99, SW01] を時間に関して拡張し，入出力動作に加えて実時間性に関しても等価判定を行えるようにする．

実時間システムのデッドラインに関する仕様は，対象となる処理の実行時間の上限として与えられる．仕様として与えられた時間内に実行されることを示せばデッドラインに関する仕様を満たすことがわかる．そこで，動作を実行するタイミングの違いを表す擬順序関係を提案する．提案する擬順序関係は，入出力動作についてはいずれのプロセスも他方のプロセスを模倣できるが，ある入出力動作から次の入出力動作までに経過する時間が一方のプロセスの方が常に短いことを表す．このことにより，遅い方のプロセスをデッドライン仕様とみなせば，擬順序関係が成り立つ時に仕様が満たされているといえる．

本章では，時間付き π 計算のプロセス間の関係として等価関係と擬順序関係を定義し，それらの性質について述べる．また，いずれの関係についても内部動作 τ を抽象して外部から観測可能な動作のみに着目した関係を与える．すべての関係は制限されたコンテキストに対して保存される．合同的性質を持つための十分条件を示す．

本章の構成は以下の通りである．3.2 節では等価関係として時間付き双模倣関係について述べる．関係を定義し，入力プレフィックスを除くコンテキストに対して合同であることを示す．3.3 節では擬順序関係として遅延時間順関係を提案する．プロセスの振舞いが時間経過に影響されないことを表す時間独立の概念を導入し，

遅延時間順関係が入力プレフィックスと時間独立ではないプロセスの並行合成を除くコンテキストに対して合同であることを示す．3.4 節でストリーミング配信システムを用いて例を挙げ，最後に関連研究を挙げてまとめを述べる．

3.2　等価関係

π 計算における双模倣関係を基礎として，時間付き π 計算のプロセス式の等価関係を定義する．

3.2.1　時間付き強双模倣関係

時間付き強双模倣関係は直観的には以下を満たす関係である．

- 双方のプロセスの入出力動作および内部動作の列が同じ
- 双方のプロセスにおいて時間待ちが発生するタイミングと時間待ちの長さが同じ

時間付き強双模倣関係では内部動作と時間経過を含むすべての動作を対象として双模倣性を定義する．双模倣なプロセスをその振舞いによって区別することは不可能である．

定義 3.1 $\alpha \in Act$ に対して強遷移関係を以下のように定義する．

- $\xrightarrow{\alpha} \stackrel{def}{=} \stackrel{\bullet}{\rightarrow}^{*} \stackrel{\alpha}{\rightarrow} \stackrel{\bullet}{\rightarrow}^{*}$
- $\rightsquigarrow \stackrel{def}{=} \stackrel{\bullet}{\rightarrow}^{*} \rightarrow \stackrel{\bullet}{\rightarrow}^{*}$ □

定義 3.2 以下を満たす対称な関係 $\mathcal{R}$ を時間付き強双模倣関係と呼ぶ．
$(P, Q) \in \mathcal{R}$ の時，

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

$(P, Q) \in \mathcal{R}$ なる時間付き強双模倣関係 $\mathcal{R}$ が存在する時 $P \sim_T Q$ と書く．

定義 3.3 $\sim_T \ = \bigcup\{\mathcal{R} \mid \mathcal{R}$ は時間付き強双模倣関係 $\}$ □

命題 3.1 $\sim_{\mathcal{T}}$ は最大の時間付き強双模倣関係である．□

命題 3.2 $\sim_{\mathcal{T}}$ は等価関係である．□

時間付き強双模倣関係はプロセスの振舞等価性の基礎である．もしプロセス P と Q が時間付き強双模倣関係にあれば，実行系列から P と Q を区別することは不可能である．時間付き強双模倣関係では，入出力動作，内部動作，時間経過動作を等しく扱う．例えば，プロセス $t[\underline{1}].\overline{x}.\mathbf{0} \mid t[\underline{1}].y.\mathbf{0}$ と $t[\underline{1}].(\overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0})$ は，以下に示す関係 $\mathcal{R}$ から時間付き強双模倣関係であることがわかる．

$$
\begin{aligned}
\mathcal{R} \;=\; \{&(t[\underline{1}].\overline{x}.\mathbf{0} \mid t[\underline{1}].y.\mathbf{0},\ t[\underline{1}].(\overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0})),\\
&(t[\underline{0}].\overline{x}.\mathbf{0} \mid t[\underline{0}].y.\mathbf{0},\ t[\underline{0}].(\overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0})),\\
&(\overline{x}.\mathbf{0} \mid t[\underline{0}].y.\mathbf{0},\ t[\underline{0}].(\overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0})),\\
&(t[\underline{0}].\overline{x}.\mathbf{0} \mid y.\mathbf{0},\ t[\underline{0}].(\overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0})),\\
&(\overline{x}.\mathbf{0} \mid y.\mathbf{0},\ t[\underline{0}].(\overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0})),\\
&(t[\underline{0}].\overline{x}.\mathbf{0} \mid t[\underline{0}].y.\mathbf{0},\ \overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0}),\\
&(\overline{x}.\mathbf{0} \mid t[\underline{0}].y.\mathbf{0},\ \overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0}),\\
&(t[\underline{0}].\overline{x}.\mathbf{0} \mid y.\mathbf{0},\ \overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0}),\\
&(\overline{x}.\mathbf{0} \mid y.\mathbf{0},\ \overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0}),\\
&(\mathbf{0} \mid y.\mathbf{0},\ y.\mathbf{0}),\\
&(\overline{x}.\mathbf{0} \mid \mathbf{0},\ \overline{x}.\mathbf{0}),\\
&(\mathbf{0} \mid \mathbf{0},\ \mathbf{0})\}
\end{aligned}
$$

一方，これらのプロセスに対して代入 $\{y/x\}$ を適用すると時間付き強双模倣関係が成り立たなくなる．つまり

$$(t[\underline{1}].\overline{x}.\mathbf{0} \mid t[\underline{1}].y.\mathbf{0})\{y/x\} \quad \not\sim_{\mathcal{T}} \quad (t[\underline{1}].(\overline{x}.y.\mathbf{0} + y.\overline{x}.\mathbf{0}))\{y/x\}$$

である．なぜなら左辺のプロセスは $t[\underline{1}].\overline{y}.\mathbf{0} \mid t[\underline{1}].y.\mathbf{0}$ であるため 1 単位時間後に τ 遷移できるが，右辺のプロセスは $t[\underline{1}].(\overline{y}.y.\mathbf{0} + y.\overline{y}.\mathbf{0})$ であるため不可能だからである．

時間付き強双模倣関係は入力プレフィックスを除く演算子について保存される．

定義 3.4 任意のプロセス式 P と $n \in \mathcal{I}$ に対し，構文

$$P_g \;::=\; x(y).P \mid \overline{x}\langle z\rangle.P \mid \tau.P \mid t[n].P \mid P_g + P_g$$

によって定義できるプロセス式 P_g をガード付きプロセスという．　□

定義 3.5 入力プレフィックスを含まないコンテキスト $C_{ni}[\cdot]$ を以下のように定義する．

$$\begin{aligned} C_{ni}[\cdot] \;::=\; & [\cdot] \mid \overline{x}\langle z\rangle.C_{ni}[\cdot] \mid \tau.C_{ni}[\cdot] \mid t[n].C_{ni}[\cdot] \\ & \mid \overline{x}\langle z\rangle.C_{ni}[\cdot]+R \mid \tau.C_{ni}[\cdot]+R \mid t[n].C_{ni}[\cdot]+R \\ & \mid C_{ni}[\cdot] \mid S \mid \boldsymbol{\nu} x\, C_{ni}[\cdot] \mid !\,C_{ni}[\cdot] \end{aligned}$$

ここで R はガード付きプロセス，S は任意のプロセスとする．　□

補題 3.1 $P_1 \sim_{\mathcal{T}} P_2$ かつ $Q_1 \sim_{\mathcal{T}} Q_2$ ならば $P_1 \mid Q_1 \sim_{\mathcal{T}} P_2 \mid Q_2$ である．

証明： $\mathcal{R} = \{(P_1 \mid Q_1,\, P_2 \mid Q_2) \mid P_1 \sim_{\mathcal{T}} P_2,\, Q_1 \sim_{\mathcal{T}} Q_2\}$ に対し $\mathcal{R} \subseteq \sim_{\mathcal{T}}$ を示せばよい．$(P_1 \mid Q_1,\, P_2 \mid Q_2) \in \mathcal{R}$ とする．

- $P_1 \mid Q_1 \xrightarrow{\alpha} P_1' \mid Q_1$（ただし $P_1 \xrightarrow{\alpha} P_1'$）の時，$P_1 \sim_{\mathcal{T}} P_2$ ゆえ P_1' に対し $P_2 \xrightarrow{\alpha} P_2'$ かつ $P_1' \sim_{\mathcal{T}} P_2'$ なる P_2' が存在する．この時，$P_2 \mid Q_2 \xrightarrow{\alpha} P_2' \mid Q_2$ とできて $P_1' \sim_{\mathcal{T}} P_2'$ ゆえ $(P_1' \mid Q_1,\, P_2' \mid Q_2) \in \mathcal{R}$ である．
- $P_1 \mid Q_1 \xrightarrow{\alpha} P_1 \mid Q_1'$（ただし $Q_1 \xrightarrow{\alpha} Q_1'$）の時も同様である．
- $P_1 \mid Q_1 \xrightarrow{\tau} P_1' \mid Q_1'$（ただし $P_1 \xrightarrow{\alpha} P_1'$，$Q_1 \xrightarrow{\overline{\alpha}} Q_1'$）の時，$P_1'$ に対し $P_2 \xrightarrow{\alpha} P_2' \sim_{\mathcal{T}} P_1'$ なる P_2' が存在し，Q_1' に対し $Q_2 \xrightarrow{\overline{\alpha}} Q_2' \sim_{\mathcal{T}} Q_1'$ なる Q_2' が存在する．よって $P_2 \mid Q_2 \xrightarrow{\tau} P_2' \mid Q_2$ とできて $(P_1' \mid Q_1',\, P_2' \mid Q_2') \in \mathcal{R}$ である．
- $P_1 \mid Q_1 \rightsquigarrow P_1' \mid Q_1'$（ただし $P_1 \rightsquigarrow P_1'$，$Q_1 \rightsquigarrow Q_1'$）の時，$P_1'$ に対し $P_2 \rightsquigarrow P_2' \sim_{\mathcal{T}} P_1'$ なる P_2' が存在し，Q_1' に対し $Q_2 \rightsquigarrow Q_2' \sim_{\mathcal{T}} Q_1'$ なる Q_2' が存在する．よって $P_2 \mid Q_2 \rightsquigarrow P_2' \mid Q_2$ とできて $(P_1' \mid Q_1',\, P_2' \mid Q_2') \in \mathcal{R}$ である．
- 他の場合も同様である．

以上より $\mathcal{R} \subseteq \sim_{\mathcal{T}}$ である．　□

補題 3.2 $P \sim_{\mathcal{T}} Q$ ならば $!P \sim_{\mathcal{T}} !Q$ である．

証明： 初めに $\mathcal{R} = \{(!P \mid R,\, !Q \mid S) \mid P \sim_{\mathcal{T}} Q,\, R \sim_{\mathcal{T}} S\}$ に対し $\mathcal{R} \subseteq \sim_{\mathcal{T}}$ を示す．

- $!P \mid R \xrightarrow{\alpha} P' \mid R$（ただし $!P \xrightarrow{\alpha} P'$）の時，

- $\alpha \neq \tau$ の時，$P \xrightarrow{\alpha} P''$ とすると $P' = !P \mid P''$ である．$P \sim_{\mathcal{T}} Q$ ゆえ P'' に対し $Q \xrightarrow{\alpha} Q'' \sim_{\mathcal{T}} P''$ なる Q'' が存在する．この時 $!Q \xrightarrow{\alpha} !Q \mid Q''$ とできる．$P'' \sim_{\mathcal{T}} Q''$ かつ $R \sim_{\mathcal{T}} S$ ゆえ補題 3.1 より $P'' \mid R \sim_{\mathcal{T}} Q'' \mid S$ であるので，$(!P \mid P'' \mid R, !Q \mid Q'' \mid S) \in \mathcal{R}$ である．
- $\alpha = \tau$ の時，$P \xrightarrow{\beta} P''$，$P \xrightarrow{\overline{\beta}} P'''$ とすると $P' = !P \mid P'' \mid P'''$ である．$P \sim_{\mathcal{T}} Q$ ゆえ P'' に対し $Q \xrightarrow{\beta} Q'' \sim_{\mathcal{T}} P''$ なる Q'' が存在し，P''' に対し $Q \xrightarrow{\overline{\beta}} Q''' \sim_{\mathcal{T}} P'''$ なる Q''' が存在する．この時 $!Q \xrightarrow{\tau} !Q \mid Q'' \mid Q'''$ とできる．$P'' \sim_{\mathcal{T}} Q''$，$P''' \sim_{\mathcal{T}} Q'''$，$R \sim_{\mathcal{T}} S$ ゆえ補題 3.1 より $P'' \mid P''' \mid R \sim_{\mathcal{T}} Q'' \mid Q''' \mid S$ であるので，$(!P \mid P'' \mid P''' \mid R, !Q \mid Q'' \mid Q''' \mid S) \in \mathcal{R}$ である．

- $!P \mid R \xrightarrow{\alpha} !P \mid R'$（ただし $R \xrightarrow{\alpha} R'$）の時，$R \sim_{\mathcal{T}} S$ ゆえ R' に対し $S \xrightarrow{\alpha} S' \sim_{\mathcal{T}} R'$ なる S' が存在する．よって $(!P \mid R', !Q \mid S') \in \mathcal{R}$ である．

- $!P \mid R \xrightarrow{\tau} P' \mid R'$（ただし $!P \xrightarrow{\alpha} P'$，$R \xrightarrow{\overline{\alpha}} R'$）の時，$P \xrightarrow{\alpha} P''$ とすると $P' = !P \mid P''$ である．$P \sim_{\mathcal{T}} Q$ ゆえ P'' に対し $Q \xrightarrow{\alpha} Q'' \sim_{\mathcal{T}} P''$ なる Q'' が存在し，$!Q \xrightarrow{\alpha} !Q \mid Q''$ とできる．また R' に対し $S \xrightarrow{\overline{\alpha}} S' \sim_{\mathcal{T}} R'$ なる S' が存在する．この時，$!Q \mid S \xrightarrow{\tau} !Q \mid Q'' \mid S'$ とできて，補題 3.1 より $P'' \mid R' \sim_{\mathcal{T}} Q'' \mid S'$ であるので，$(!P \mid P'' \mid R', !Q \mid Q'' \mid S') \in \mathcal{R}$ である．

- $!P \mid R \rightsquigarrow P' \mid R'$（ただし $!P \rightsquigarrow P'$，$R \rightsquigarrow R'$）の時，$P \rightsquigarrow P''$ とすると $P' = !P''$ である．P'' に対し $Q \rightsquigarrow Q'' \sim_{\mathcal{T}} P''$ なる Q'' が存在し，$!Q \rightsquigarrow !Q''$ とできる．R' に対し $S \rightsquigarrow S' \sim_{\mathcal{T}} R'$ なる S' が存在する．この時，$!Q \mid S \rightsquigarrow !Q'' \mid S'$ とできて $(!P'' \mid R', !Q'' \mid S') \in \mathcal{R}$ である．

- 他の場合も同様である．

以上より $\mathcal{R} \subseteq \sim_{\mathcal{T}}$ である．ここで，$R = S = \mathbf{0}$ とすると明らかに $R \sim_{\mathcal{T}} S$ であるので $(!P, !Q) \in \mathcal{R}$ である．よって $!P \sim_{\mathcal{T}} !Q$ である． □

定理 3.1 $P \sim_{\mathcal{T}} Q$ ならば $C_{ni}[P] \sim_{\mathcal{T}} C_{ni}[Q]$ である．

証明： コンテキストの構造に関する帰納法による．

- $C_{ni}[\cdot] = [\cdot]$ の場合，明らか．
- $C_{ni}[\cdot] = \overline{x}\langle z \rangle.C'_{ni}[\cdot]$ の場合，$C_{ni}[P] \xrightarrow{\overline{x}\langle z \rangle} C'_{ni}[P]$，$C_{ni}[Q] \xrightarrow{\overline{x}\langle z \rangle} C'_{ni}[Q]$ であり，帰納法の仮定より $C_{ni}[P] \sim_{\mathcal{T}} C_{ni}[Q]$．また，$C_{ni}[P] \rightsquigarrow C_{ni}[P]$，$C_{ni}[Q] \rightsquigarrow C_{ni}[Q]$ ゆえ $C_{ni}[P] \sim_{\mathcal{T}} C_{ni}[Q]$ である．

- $C[\cdot] = t[n].C'_{ni}[\cdot]$ の場合，$n = \underline{k}$ とすると
  - $k = 0$ の時，$C_{ni}[P] \stackrel{\bullet}{\rightharpoonup} C'_{ni}[P]$，$C_{ni}[Q] \stackrel{\bullet}{\rightharpoonup} C'_{ni}[Q]$ である．帰納法の仮定より $C'_{ni}[P] \sim_{\mathcal{T}} C'_{ni}[Q]$ であり，強遷移関係では $\stackrel{\bullet}{\rightharpoonup}$ は抽象されるため $C_{ni}[P] \sim_{\mathcal{T}} C_{ni}[Q]$ である．
  - $k = 1$ の時，$C_{ni}[P] \stackrel{\alpha}{\nrightarrow}$ かつ $C_{ni}[Q] \stackrel{\alpha}{\nrightarrow}$ である．$C_{ni}[P] \rightsquigarrow t[\underline{0}].C'_{ni}[P]$ に対して帰納法の仮定から $C_{ni}[Q] \rightsquigarrow t[\underline{0}].C'_{ni}[Q] \sim_{\mathcal{T}} t[\underline{0}].C'_{ni}[P]$ が，$C_{ni}[P] \rightsquigarrow C'_{ni}[P]$ に対しても帰納法の仮定から $C_{ni}[Q] \rightsquigarrow C'_{ni}[Q] \sim_{\mathcal{T}} C'_{ni}[P]$ が成り立つため，$C_{ni}[P] \sim_{\mathcal{T}} C_{ni}[Q]$ である．
  - $k > 1$ の時，$C_{ni}[P] \stackrel{\alpha}{\nrightarrow}$ かつ $C_{ni}[Q] \stackrel{\alpha}{\nrightarrow}$ である．$C_{ni}[P] \rightsquigarrow t[\underline{k-1}].C'_{ni}[P]$，$C_{ni}[Q] \rightsquigarrow t[\underline{k-1}].C'_{ni}[Q]$ であり，帰納法の仮定から $t[\underline{k-1}].C'_{ni}[P] \sim_{\mathcal{T}} t[\underline{k-1}].C'_{ni}[Q]$ ゆえ $C_{ni}[P] \sim_{\mathcal{T}} C_{ni}[Q]$ である．
- $C_{ni}[\cdot] = C'_{ni}[\cdot] \mid S$ の場合，帰納法の仮定より $C'_{ni}[P] \sim_{\mathcal{T}} C'_{ni}[Q]$ である．$S \sim_{\mathcal{T}} S$ であるので補題 3.1 より $C_{ni}[P] \sim_{\mathcal{T}} C_{ni}[Q]$ である．
- $C_{ni}[\cdot] =! C'_{ni}[\cdot]$ の場合，帰納法の仮定より $C'_{ni}[P] \sim_{\mathcal{T}} C'_{ni}[Q]$ である．補題 3.2 より $C_{ni}[P] \sim_{\mathcal{T}} C_{ni}[Q]$ である．
- 他の場合も同様である． □

時間付き強双模倣関係は従来の π 計算における強双模倣関係と同様に入力プレフィックスに対して保存されない．従来の π 計算における強双模倣関係の定義を時間経過動作に関して拡張しただけであり，入出力動作や内部動作に関する性質はそのまま継承している．

強遷移関係はタイムアウト遷移を抽象するため次の性質が成り立つ．

補題 3.3 $t[\underline{0}].P \sim_{\mathcal{T}} P$

証明： 関係

$$\mathcal{R} \;=\; \{(t[\underline{0}].P,\ P) \mid P \text{ は任意のプロセス}\} \cup Id$$

が時間付き強双模倣関係であることを示せばよい． □

時間付き強双模倣関係は選択演算子そのものに対しては保存されない．例えば，補題 3.3 より $t[\underline{0}].\overline{a}.P \sim_{\mathcal{T}} \overline{a}.P$ であるが，$t[\underline{0}].\overline{a}.P + b.Q \not\sim_{\mathcal{T}} \overline{a}.P + b.Q$ である．$t[\underline{0}].\overline{a}.P \stackrel{\bullet}{\rightharpoonup} \overline{a}.P$ ゆえ $t[\underline{0}].\overline{a}.P + b.Q$ の遷移は $t[\underline{0}].\overline{a}.P + b.Q \stackrel{\overline{a}}{\rightarrow} P$ のみである．しかし，$\overline{a}.P + b.Q$

の遷移は $\overline{a}.P + b.Q \xrightarrow{\overline{a}} P$ と $\overline{a}.P + b.Q \xrightarrow{b} Q$ の 2 通り存在する．定理 3.1 において時間付き強双模倣関係が選択演算子を付加するコンテキストに対して保存されるのは，選択演算子を付加するコンテキストを適用して得られるプロセスがガード付きプロセスになるように制限されているからである．$[\cdot] + b.Q$ のようなコンテキストは正しいコンテキストではない．$(P \mid R)$ に適用して得られる $(P \mid R) + b.Q$ は定義 2.1 で与えた時間付き π 計算のプロセス式の構文に反する．

3.2.2 時間付き弱双模倣関係

時間付き強双模倣関係ではプロセスの内部動作に関しても互いに模倣することが要求される．しかし，時間付き弱双模倣関係は内部動作を模倣することは要求しない．1 回の内部動作に対して 0 回を含む任意の回数の内部動作が対応する．内部動作は不可視であるため，プロセスの振舞いについて外部環境に対する動作を観測して等価判定を行う場合は時間付き弱双模倣関係が適している．

時間付き弱双模倣関係では内部動作を抽象するため，入出力動作の列および時間待ちが発生するタイミングと時間待ちの長さが同じプロセスが振舞い等価であるとされる．

定義 3.6 $\alpha \in Act$ に対して弱遷移関係を以下のように定義する．

- $\longrightarrow_\tau \stackrel{def}{=} \xrightarrow{\tau}{}^*$
- $\xrightarrow{\alpha}_\tau \stackrel{def}{=} \longrightarrow_\tau \xrightarrow{\alpha} \longrightarrow_\tau$
- $\rightsquigarrow_\tau \stackrel{def}{=} \longrightarrow_\tau \rightsquigarrow \longrightarrow_\tau$ □

以下では簡単のために，$\xrightarrow{\widehat{\tau}}_\tau$ は $\longrightarrow_\tau$ を，τ 以外の動作 β に対して $\xrightarrow{\widehat{\beta}}_\tau$ は $\xrightarrow{\beta}_\tau$ を表す記法を用いる．

定義 3.7 以下を満たす対称な関係 $\mathcal{R}$ を時間付き弱双模倣関係と呼ぶ．
$(P, Q) \in \mathcal{R}$ の時，

- $P \xrightarrow{\alpha} P' \;\Rightarrow\; \exists Q'.\ Q \xrightarrow{\widehat{\alpha}}_\tau Q' \wedge (P', Q') \in \mathcal{R}$
- $P \rightsquigarrow P'' \;\Rightarrow\; \exists Q''.\ Q \rightsquigarrow_\tau Q'' \wedge (P'', Q'') \in \mathcal{R}$ □

$(P, Q) \in \mathcal{R}$ なる時間付き弱双模倣関係 $\mathcal{R}$ が存在する時 $P \approx_{\mathcal{T}} Q$ と書く．

定義 3.8 $\approx_{\mathcal{T}} = \bigcup\{\mathcal{R} \mid \mathcal{R}$ は時間付き弱双模倣関係 $\}$ □

命題 3.3 $\approx_{\mathcal{T}}$ は最大の時間付き弱双模倣関係である. □

命題 3.4 $\approx_{\mathcal{T}}$ は等価関係である. □

定義より明らかに，時間付き強双模倣関係は時間付き弱双模倣関係でもある.

補題 3.4 $P \sim_{\mathcal{T}} Q$ ならば $P \approx_{\mathcal{T}} Q$ である.

証明： $\sim_{\mathcal{T}}$ および $\approx_{\mathcal{T}}$ の定義より明らか. □

時間付き強双模倣関係と同様に，時間付き弱双模倣関係も入力プレフィックスを除く演算子について保存される.

定理 3.2 $P \approx_{\mathcal{T}} Q$ ならば $C_{ni}[P] \approx_{\mathcal{T}} C_{ni}[Q]$ である.

証明： 定理 3.1 と同様. □

時間付き強双模倣関係と同様に，時間付き弱双模倣関係も選択演算子そのものに対して保存されない．補題 3.4 より時間付き強双模倣関係と同様な反例が挙げられる．また，弱遷移関係は τ 遷移を抽象するため次の性質が成り立つ.

補題 3.5 $\tau.P \approx_{\mathcal{T}} P$

証明： 関係

$$\mathcal{R} \;=\; \{(\tau.P,\ P) \mid P \text{ は任意のプロセス}\} \cup Id$$

が時間付き弱双模倣関係であることを示せばよい. □

この性質により $\tau.t[\underline{0}].\overline{a}.P \approx_{\mathcal{T}} t[\underline{0}].\overline{a}.P$ である．しかし，$\tau.t[\underline{0}].\overline{a}.P + b.Q \not\approx_{\mathcal{T}} t[\underline{0}].\overline{a}.P + b.Q$ である．$t[\underline{0}].\overline{a}.P + b.Q$ の遷移は $t[\underline{0}].\overline{a}.P + b.Q \xrightarrow{\overline{a}}_{\tau} P$ のみであり，$\tau.t[\underline{0}].\overline{a}.P + b.Q \xrightarrow{b}_{\tau} Q$ に対応する遷移が存在しない．時間付き強双模倣関係の場合と同様に，ガード付きプロセスになるようにコンテキストの形を制限することで，時間付き弱双模倣関係の合同性が成り立つ.

3.3　擬順序関係

時間付き π 計算のプロセス式に対し，入出力動作のタイミングの違いを表現するための擬順序関係を定義する.

3.3.1 遅延時間順関係

遅延時間順関係は直観的には以下を満たす関係である.

- 双方のプロセスの入出力動作の列が同じ
- 双方のプロセスにおいて時間待ちが発生するタイミングが同じ
- 一方のプロセスの方が常に時間待ちの長さが短い

入出力動作，内部動作に関しては相互に模倣することが可能であるが，一方のプロセスが他方のプロセスよりも速く動作することを表す.

定義 3.9 以下の条件を満たす関係 $\mathcal{R}$ を遅延時間順関係と呼ぶ.
$(P,Q) \in \mathcal{R}$ の時,

- $P \xrightarrow{\alpha} P' \Rightarrow \exists Q'.\ Q \rightsquigarrow^* \xrightarrow{\alpha} Q' \wedge (P',Q') \in \mathcal{R}$
- $P \rightsquigarrow P' \Rightarrow \exists Q'.\ Q \rightsquigarrow Q' \wedge (P',Q') \in \mathcal{R}$
- $Q \xrightarrow{\alpha} Q' \Rightarrow \exists P'.\ P \xrightarrow{\alpha} P' \wedge (P',Q') \in \mathcal{R}$
- $Q \rightsquigarrow Q' \Rightarrow (P,Q') \in \mathcal{R} \vee (\exists P'.\ P \rightsquigarrow P' \wedge (P',Q') \in \mathcal{R})$ □

$(P,Q) \in \mathcal{R}$ なる遅延時間順関係 $\mathcal{R}$ が存在する時 $P \precsim_{\mathcal{T}} Q$ と書く.

定義 3.10 $\precsim_{\mathcal{T}} = \bigcup\{\mathcal{R} \mid \mathcal{R}$ は遅延時間順関係 $\}$ □

命題 3.5 $\precsim_{\mathcal{T}}$ は最大の遅延時間順関係である. □

命題 3.6 $\precsim_{\mathcal{T}}$ は擬順序関係である. □

直観的には，$P \precsim_{\mathcal{T}} Q$ ならば時間経過動作による遷移の回数は P より Q の方が多いが，その他の動作による遷移は P と Q のいずれも同じである．もし P が時間経過以外の何らかの動作が実行可能ならば，Q は 0 単位時間以上後に同じ動作が実行できる．P が時間経過遷移するのであれば，時間経過遷移の回数は P の方が少ないので Q は P と同様に時間経過遷移できなければならない．同じ理由から Q が時間経過以外の動作を実行できるならば，P は同じ動作が即座に実行できなければならない．Q が時間経過遷移する時，P は遷移後に関係が保たれるのであれば時間経過遷移することができる.

例えば，$t[\underline{5}].\overline{a}.\mathbf{0} \precsim_{\mathcal{T}} t[\underline{10}].\overline{a}.\mathbf{0}$ である．左辺のプロセスは 5 単位時間後に動作 $\overline{a}$ を実行し，右辺のプロセスは 10 単位時間後である．動作 $\overline{a}$ を実行する点は同じであるが，その前に経過する時間の長さが異なり左辺のプロセスの方が早い．双方のプロセスの遷移は以下のようになる．

$$
\begin{array}{ccccccccccc}
t[\underline{5}].a.\mathbf{0} & \rightsquigarrow & t[\underline{4}].a.\mathbf{0} & \rightsquigarrow & \cdots & \rightsquigarrow & t[\underline{0}].a.\mathbf{0} & \rightsquigarrow & a.\mathbf{0} & \cdots \\
 & & & & & & \downarrow_{\wp} & & \downarrow_{\wp} & \\
 & & & & & & \mathbf{0} & & \mathbf{0} &
\end{array}
$$

$$
\begin{array}{ccccccccccc}
t[\underline{10}].a.\mathbf{0} & \rightsquigarrow & t[\underline{9}].a.\mathbf{0} & \rightsquigarrow & \cdots & \rightsquigarrow & t[\underline{5}].a.\mathbf{0} & \rightsquigarrow & t[\underline{4}].a.\mathbf{0} & \cdots \\
 & & & & & & \wr^{*} & & \wr^{*} & \\
 & & & & & & \downarrow_{\wp} & & \downarrow_{\wp} & \\
 & & & & & & \mathbf{0} & & \mathbf{0} &
\end{array}
$$

遅延時間順関係は，入力プレフィックスおよび時間経過によって状態が変化するプロセスの並行合成を除く演算子について保存される．

定義 3.11 $P \stackrel{\tau}{\not\rightarrow}$ かつ $P \rightsquigarrow P$ の時，P は時間安定であり，$TS(P)$ と書く．$derive(P) = \{P' \mid P(\stackrel{\alpha}{\rightarrow})^{*}P'\}$ とする．この時，$\forall Q \in derive(P)\,.\,TS(Q)$ ならば，P は時間独立であり，$TI(P)$ と書く． □

$TI(P)$ の定義より明らかに $TI(P)$ ならば $P \stackrel{\alpha}{\longrightarrow} P'$ なるすべての P' に対して $TI(P')$ である．

定義 3.12 入力プレフィックスを含まず，並行合成するプロセスが時間独立なプロセスに制限されるコンテキスト C_{ti} を以下のように定義する．

$$
\begin{array}{rcl}
C_{ti}[\cdot] & ::= & [\cdot] \mid \overline{x}\langle z\rangle.C_{ti}[\cdot] \mid \tau.C_{ti}[\cdot] \mid t[n].C_{ti}[\cdot] \\
 & & \mid \overline{x}\langle z\rangle.C_{ti}[\cdot]{+}R \mid \tau.C_{ti}[\cdot]{+}R \mid t[n].C_{ti}[\cdot]{+}R \\
 & & \mid C_{ti}[\cdot] \mid S \mid \boldsymbol{\nu} x\, C_{ti}[\cdot]
\end{array}
$$

ここで R はガード付きプロセス，S は時間独立なプロセスとする． □

定理 3.3 $P \precsim_{\mathcal{T}} Q$ ならば $C_{ti}[P] \precsim_{\mathcal{T}} C_{ti}[Q]$ である．

証明： コンテキストの構造に関する帰納法による．

- $C_{ti}[\cdot] = [\cdot]$ の場合，明らか．

- $C_{ti}[\cdot] = \overline{x}\langle z\rangle.C'_{ti}[\cdot]$ の場合，$C_{ti}[P] \xrightarrow{\overline{x}\langle z\rangle} C'_{ti}[P]$，$C_{ti}[Q] \xrightarrow{\overline{x}\langle z\rangle} C'_{ti}[Q]$ であり，帰納法の仮定より $C'_{ti}[P] \precsim_{\mathcal{T}} C'_{ti}[Q]$．また，$C_{ti}[P] \rightsquigarrow C_{ti}[P]$，$C_{ti}[Q] \rightsquigarrow C_{ti}[Q]$ ゆえ $C_{ti}[P] \precsim_{\mathcal{T}} C_{ti}[Q]$ である．

- $C_{ti}[\cdot] = t[n].C'_{ti}[\cdot]$ の場合，$n = \underline{k}$ とすると

  - $k = 0$ の時，$C_{ti}[P] \xrightarrow{\bullet}$ $C'_{ti}[P]$，$C_{ti}[Q] \xrightarrow{\bullet}$ $C'_{ti}[Q]$ である．帰納法の仮定より $C'_{ti}[P] \sim_{\mathcal{T}} C'_{ti}[Q]$ であり，強遷移関係では $\xrightarrow{\bullet}$ は抽象されるため $C_{ti}[P] \sim_{\mathcal{T}} C_{ti}[Q]$ である．
  - $k = 1$ の時，$C_{ti}[P] \overset{\alpha}{\nrightarrow}$ かつ $C_{ti}[Q] \overset{\alpha}{\nrightarrow}$ である．$C_{ti}[P] \rightsquigarrow t[\underline{0}].C'_{ti}[P]$ に対して帰納法の仮定から $C_{ti}[Q] \rightsquigarrow t[\underline{0}].C'_{ti}[Q] \sim_{\mathcal{T}} t[\underline{0}].C'_{ti}[P]$ が，$C_{ti}[P] \rightsquigarrow C'_{ti}[P]$ に対しても帰納法の仮定から $C_{ti}[Q] \rightsquigarrow C'_{ti}[Q] \sim_{\mathcal{T}} C'_{ti}[P]$ が成り立つため，$C_{ti}[P] \sim_{\mathcal{T}} C_{ti}[Q]$ である．
  - $k > 1$ の時，$C_{ti}[P] \overset{\alpha}{\nrightarrow}$ かつ $C_{ti}[Q] \overset{\alpha}{\nrightarrow}$ である．$C_{ti}[P] \rightsquigarrow t[\underline{k-1}].C'_{ti}[P]$，$C_{ti}[Q] \rightsquigarrow t[\underline{k-1}].C'_{ti}[Q]$ であり，帰納法の仮定から $t[\underline{k-1}].C'_{ti}[P] \sim_{\mathcal{T}} t[\underline{k-1}].C'_{ti}[Q]$ ゆえ $C_{ti}[P] \sim_{\mathcal{T}} C_{ti}[Q]$ である．

- $C_{ti}[\cdot] = C'_{ti}[\cdot] \mid S$ の場合，$\mathcal{R} = \{(C'_{ti}[P] \mid S,\ C'_{ti}[Q] \mid S) \mid P \precsim_{\mathcal{T}} Q,\ TI(S)\} \subseteq \precsim_{\mathcal{T}}$ を示せばよい．S は時間独立なプロセスであるため $S \rightsquigarrow^* S$ である．

  - $C'_{ti}[P] \mid S \xrightarrow{\alpha}$ の場合，
    - $C'_{ti}[P] \mid S \xrightarrow{\alpha} P' \mid S$（ただし $C'_{ti}[P] \xrightarrow{\alpha} P'$）の時，帰納法の仮定より $C'_{ti}[P] \precsim_{\mathcal{T}} C'_{ti}[Q]$ であるので P' に対して $C'_{ti}[Q] \rightsquigarrow^* \xrightarrow{\alpha} Q'$ かつ $P' \precsim_{\mathcal{T}} Q'$ なる Q' が存在する．$C'_{ti}[Q] \rightsquigarrow^* Q^*$ とすると，S は時間独立であるため $C'_{ti}[Q] \mid S \rightsquigarrow^* Q^* \mid S$ となる．さらに $Q^* \mid S \xrightarrow{\alpha} Q' \mid S$ とできて，$P' \precsim_{\mathcal{T}} Q'$ ゆえ $(P' \mid S,\ Q' \mid S) \in \mathcal{R}$ である．
    - $C'_{ti}[P] \mid S \xrightarrow{\alpha} C'_{ti}[P] \mid S'$（ただし $S \xrightarrow{\alpha} S'$）の時，$C'_{ti}[Q] \xrightarrow{\alpha} C'_{ti}[Q] \mid S'$ とできて，時間独立の定義から S' も時間独立なプロセスである．よって $(C'_{ti}[P] \mid S',\ C'_{ti}[Q] \mid S') \in \mathcal{R}$ である．
    - $C'_{ti}[P] \mid S \xrightarrow{\tau} P' \mid S'$（ただし $C'_{ti}[P] \xrightarrow{\alpha} P'$，$S \xrightarrow{\overline{\alpha}} S'$）の時，帰納法の仮定より $C'_{ti}[P] \precsim_{\mathcal{T}} C'_{ti}[Q]$ であるので P' に対して $C'_{ti}[Q] \rightsquigarrow^* \xrightarrow{\alpha} Q'$ かつ $P' \precsim_{\mathcal{T}} Q'$ なる Q' が存在する．$C'_{ti}[Q] \rightsquigarrow^* Q^*$ とすると，S は時間独立であるため $C'_{ti}[Q] \mid S \rightsquigarrow^* Q^* \mid S$ となる．ここで，$Q^* \xrightarrow{\alpha} Q'$，$S \xrightarrow{\overline{\alpha}} S'$ ゆえ $Q^* \mid S \xrightarrow{\tau} Q' \mid S'$ とできて，S' も時間独立であるため $(P' \mid S',\ Q' \mid S') \in \mathcal{R}$ である．

– $C'_{ti}[Q] \mid S \rightsquigarrow Q' \mid S$（ただし $C'_{ni}[Q] \rightsquigarrow Q'$）の場合，帰納法の仮定より $C'_{ti}[P] \precsim_{\mathcal{T}} Q'$ あるいは Q' に対し $C'_{ti}[P] \rightsquigarrow P' \precsim_{\mathcal{T}} Q'$ なる P' が存在する．$C'_{ti}[P] \precsim_{\mathcal{T}} Q'$ ならば $(C'_{ti}[P] \mid S,\ Q' \mid S) \in \mathcal{R}$ である．そうでなければ $S \rightsquigarrow S$ ゆえ $C'_{ti}[P] \mid S \rightsquigarrow P' \mid S$ とできて，$P' \precsim_{\mathcal{T}} Q'$ ゆえ $(P' \mid S, Q' \mid S) \in \mathcal{R}$ である．

- 他の場合も同様である． □

遅延時間順関係は時間経過動作を含むプロセスとの並行合成において保存されない．例えば，$P = a.\mathbf{0}$，$Q = t[\underline{1}].a.\mathbf{0}$，$R = t[\underline{2}].b.\mathbf{0}$ とした時，$P \precsim_{\mathcal{T}} Q$ である．しかし，$P \mid R \xrightarrow{a} \mathbf{0} \mid R$ に対し $Q \mid R \rightsquigarrow \xrightarrow{a} \mathbf{0} \mid t[\underline{1}].b.\mathbf{0}$ であるため $P \mid R \not\precsim_{\mathcal{T}} Q \mid R$ となり関係は成り立たない．$P \precsim_{\mathcal{T}} Q$ の時 $Q \mid R$ の方が Q の次の入出力あるいは τ 動作までに多くの時間経過動作を必要とする．R が時間経過動作をプレフィックスに持つならば，R の次の入出力動作までに必要な時間経過は逆に少なくなる．遅延時間順関係はプロセスが遷移しても常に保たれなければならないため，並行合成されるプロセスが時間経過動作を含むことはできない．

3.3.2　弱遅延時間順関係

遅延時間順関係は時間付き強双模倣関係と同様に内部動作についても相互に模倣できる必要がある．外部から観測できる可能な動作と，動作間の時間経過のみからプロセスの振舞いを解析するためには，内部動作を抽象した関係が有用である．そこで，遅延時間順関係において τ 遷移を抽象した弱遅延時間順関係を定義する．

定義 3.13 以下の条件を満たす関係 $\mathcal{R}$ を弱遅延時間順関係と呼ぶ．
$(P, Q) \in \mathcal{R}$ の時，

- $P \xrightarrow{\alpha} P' \Rightarrow \exists Q'.\ Q \rightsquigarrow_{\tau}^{*} \xrightarrow{\widehat{\alpha}}_{\tau} Q' \wedge (P', Q') \in \mathcal{R}$
- $P \rightsquigarrow P' \Rightarrow \exists Q'.\ Q \rightsquigarrow_{\tau} Q' \wedge (P', Q') \in \mathcal{R}$
- $Q \xrightarrow{\alpha} Q' \Rightarrow \exists P'.\ P \xrightarrow{\widehat{\alpha}}_{\tau} P' \wedge (P', Q') \in \mathcal{R}$
- $Q \rightsquigarrow Q' \Rightarrow (P, Q') \in \mathcal{R} \vee (\exists P'.\ P \rightsquigarrow_{\tau} P' \wedge (P', Q') \in \mathcal{R})$ □

$(P, Q) \in \mathcal{R}$ なる弱遅延時間順関係 $\mathcal{R}$ が存在する時 $P \lessapprox_{\mathcal{T}} Q$ と書く．

定義 3.14 $\approx_{\mathcal{T}} = \bigcup\{\mathcal{R} \mid \mathcal{R}$ は時間付き弱双模倣関係 $\}$ □

命題 3.7 $\precsim_{\mathcal{T}}$ は最大の弱遅延時間順関係である. □

命題 3.8 $\precsim_{\mathcal{T}}$ は擬順序関係である. □

定義より明らかに，遅延時間順関係は弱遅延時間順関係でもある.

補題 3.6 $P \precsim_{\mathcal{T}} Q$ ならば $P \precsim_{\mathcal{T}} Q$ である.

証明： $\precsim_{\mathcal{T}}$ および $\precsim_{\mathcal{T}}$ の定義より明らか. □

遅延時間順関係と同様に，弱遅延時間順関係も入力プレフィックスおよび時間独立でないプロセスの並行合成を除く演算子について保存される.

定理 3.4 $P \precsim_{\mathcal{T}} Q$ ならば $C_{ti}[P] \precsim_{\mathcal{T}} C_{ti}[Q]$ である.

証明： 定理 3.3 と同様. □

プロセス P, Q について $P \precsim_{\mathcal{T}} Q$ である時，P と Q は入出力動作については双模倣であり，ある入出力動作から次の入出力動作までの間に経過する時間の長さは P の方が常に短い．このことから，任意の実行系列において任意の二つの入出力動作の間で経過する時間の長さは P のほうが短いということがいえる．実時間システムにおける時間制約の一つとしてデッドラインに関する仕様がある．実時間システムはデッドラインに関する仕様を満たすために，デッドラインの到達よりも早く動作を行う必要がある．システムの入出力動作とデッドラインに関する仕様を $Spec$，システムの実装を $Impl$ とすると，$Impl \precsim_{\mathcal{T}} Spec$ が成り立つことを示すことでシステムがデッドラインの到達よりも早く動作する，すなわちデッドラインに関する仕様を満たしていることが示される.

3.4 例

ストリーミング配信システムの簡単な例を示す．システムは映像を配信するサーバ，配信される映像を受信して再生するプレイヤ，サーバとプレイヤの間の通信路を確保する 2 つのルータからなる．プレイヤはいずれかのルータに接続して配信を受けるが，もし一定時間配信されてこなければもう一方のルータに接続し直す．ルータに接続してから配信の開始まで一定時間必要であり，ここで必要な時

間はルータによって異なるとする．またサーバから送信される映像は圧縮されており，プレイヤで再生する前に解凍しなければならないとする．

サーバ $Server$ は以下のように記述できる．

$$Server \stackrel{def}{=} \,!\,\overline{v}\langle video\rangle.\mathbf{0}$$

サーバは名前 v を通して映像を配信する．名前 $video$ は映像の 1 フレームを表す．実際にはフレームは次々と切り替わっていくが，簡単のためにそれらはすべて同じ名前 $video$ で表す．

2 つのルータ $Router_i$ は

$$\begin{aligned}
Router_1 \stackrel{def}{=}\ & !\,rel_1.attach(l).(\boldsymbol{\nu} u, w, g) \\
& (\overline{l}\langle w\rangle.\overline{w}\langle\underline{2}\rangle.\overline{w}\langle u\rangle.\overline{w}\langle rel_1\rangle.v(video).\overline{u}\langle video\rangle.\overline{g}.\mathbf{0} \\
& \mid !\,(g.v(video).\overline{u}\langle video\rangle.\overline{g}.\mathbf{0} + g.\mathbf{0})) \\
Router_2 \stackrel{def}{=}\ & !\,rel_2.attach(l).(\boldsymbol{\nu} u, w, g) \\
& (\overline{l}\langle w\rangle.\overline{w}\langle\underline{4}\rangle.\overline{w}\langle u\rangle.\overline{w}\langle rel_2\rangle.v(video).\overline{u}\langle video\rangle.\overline{g}.\mathbf{0} \\
& \mid !\,(g.v(video).\overline{u}\langle video\rangle.\overline{g}.\mathbf{0} + g.\mathbf{0}))
\end{aligned}$$

と記述できる．ルータは名前 $attach$ によりプレイヤから接続要求を受け，配信開始までの時間，映像を送るための名前 u，プレイヤとの接続を切断する rel_i をプレイヤに送信する．その後，名前 v によりサーバから受信した映像を名前 u を通してプレイヤに送信する．もしサーバからの受信に失敗したならばプレイヤからの切断を待つ．この例では配信開始までの時間を $Router_1$ で 2 単位時間，$Router_2$ で 4 単位時間とする．

プレイヤ $Player$ の記述を以下に示す．

$$\begin{aligned}
Player \stackrel{def}{=}\ & (\boldsymbol{\nu} l, g, g', h) \\
& (\overline{attach}\langle l\rangle.\overline{h}.\mathbf{0} \mid !\,h.l(w).w(n).w(v).w(r).t[n].\overline{g}\langle v\rangle.\overline{g'}\langle r\rangle.\mathbf{0} \\
& \mid !\,g(v).g'(r).(v(video).t[\underline{1}].\overline{play}\langle video\rangle.\overline{g}\langle v\rangle.\overline{g'}\langle r\rangle.\mathbf{0} \\
& \qquad + t[\underline{1}].\overline{attach}\langle l\rangle.\overline{r}.\overline{h}.\mathbf{0}))
\end{aligned}$$

プレイヤは初めに名前 $attach$ によりいずれかのルータに接続要求を出し，名前 l を通して配信開始までの時間，映像を受けるための名前 v，ルータとの接続を切断する r を受信する．その後，受信した時間だけ待機する．次に名前 v によりルータからの映像の配信を待機する．配信を受けたら 1 単位時間で圧縮された映像を解

凍して名前 $play$ により再生し，次の映像の待機に戻る．もし次の映像が 1 単位時間以内に配信されない場合は，名前 $attach$ により接続していないルータに接続要求を出し，r により接続しているルータと切断する．

システム全体は

$$\begin{aligned} System \stackrel{def}{=}\ & (\boldsymbol{\nu} attach, v, rel_1, rel_2) \\ & (Server \mid \overline{rel_1}.\mathbf{0} \mid Router_1 \mid \overline{rel_2}.\mathbf{0} \mid Router_2 \mid Player) \end{aligned}$$

と記述する．

システムの実行系列の一つを以下に示す．

$$\begin{aligned} System \xrightarrow{\tau}{}^{9} \rightsquigarrow^{2}\ & \cdots \\ \xrightarrow{\tau}{}^{4} \rightsquigarrow\ & (\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, g_4, u, l, h) \\ & (!\,\overline{v}\langle video\rangle.\mathbf{0} \mid !\,(g_1.\cdots + g_1.\mathbf{0}) \mid Router_1 \\ & \mid attach(l).\cdots \mid Router_2 \mid !g_3(v).g_4(r).\cdots \\ & \mid !\,h.\cdots \mid t[\underline{0}].\overline{play}\langle video\rangle.\overline{g_3}\langle u\rangle.\overline{g_4}\langle rel_1\rangle.\mathbf{0}) \\ \xrightarrow{\overline{play}}\xrightarrow{\tau}{}^{2}\ & (\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, g_4, u, l, h) \\ & (!\,\overline{v}\langle video\rangle.\mathbf{0} \mid !\,(g_1.\cdots + g_1.\mathbf{0}) \mid Router_1 \\ & \mid attach(l).\cdots \mid Router_2 \mid !h.\cdots \\ & \mid (u(video).\cdots .\overline{g_3}\langle u\rangle.\overline{g_4}\langle rel_1\rangle.\mathbf{0} + t[1].\cdots) \\ & \mid !\,g_3(v).g_4(r).\cdots) \\ \rightsquigarrow\ & (\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, g_4, u, l, h) \\ & (!\,\overline{v}\langle video\rangle.\mathbf{0} \mid !\,(g_1.\cdots + g_1.\mathbf{0}) \mid Router_1 \\ & \mid attach(l).\cdots \mid Router_2 \mid !\,h.\cdots \\ & \mid \overline{attach}\langle l\rangle.\overline{rel_1}.\overline{h}.\mathbf{0} \mid !\,g_3(v).g_4(r).\cdots) \\ \xrightarrow{\tau}{}^{8} \rightsquigarrow^{4}\ & \cdots \\ \xrightarrow{\tau}{}^{6} \rightsquigarrow\ & (\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, g_4, u, l, h) \\ & (!\,\overline{v}\langle video\rangle.\mathbf{0} \mid \overline{u}\langle video\rangle.\overline{g_2}.\mathbf{0} \mid !(g_2.\cdots + g_2.\mathbf{0}) \\ & \mid !\,(g_1.\cdots + g_1.\mathbf{0}) \mid Router_1 \mid Router_2 \\ & \mid !\,h.\cdots \mid t[\underline{0}].\overline{play}\langle video\rangle.\overline{g_3}\langle u\rangle.\overline{g_4}\langle rel_2\rangle.\mathbf{0} \\ & \mid !\,g_3(v).g_4(r).\cdots) \end{aligned}$$

$$\xrightarrow{\overline{play}} \quad \ldots$$

この実行系列は以下の動作を表す.

1. プレイヤが $Router_1$ に接続し映像を受信して再生
2. 次の映像の待機中にタイムアウトが発生
3. プレイヤは $Router_2$ に接続し次の映像を受信して再生

$\overline{play}$ によって表される映像の再生から次の再生までの遅延が最も短くなるのは，プレイヤが同じルータから続けて受信する場合で 1 単位時間の遅延が発生する．逆に遅延が最も長くなるのは，$Router_1$ に接続している状態で映像配信の待機中にタイムアウトし $Router_2$ に接続する場合で 6 単位時間の遅延が生じる．これらのことは

$$(\boldsymbol{\nu} v)\,(Server \mid !\,v(video).t[\underline{1}].\overline{play}\langle video\rangle.\mathbf{0}) \quad \precsim_{\mathcal{I}} \quad System$$

及び

$$System \quad \precsim_{\mathcal{I}} \quad (\boldsymbol{\nu} v)\,(Server \mid !\,v(video).t[\underline{6}].\overline{play}\langle video\rangle.\mathbf{0})$$

を示すことで確認できる．$Server$ は時間独立であるため，定理 3.4 より

$$!\,v(video).t[\underline{1}].\overline{play}\langle video\rangle.\mathbf{0} \quad \precsim_{\mathcal{I}} \quad \begin{array}{l}(\boldsymbol{\nu} attach, \ldots)\\ (\overline{rel_1}.\mathbf{0} \mid Router_1 \mid \overline{rel_2}.\mathbf{0} \mid Router_2 \mid Player)\end{array}$$

及び

$$\begin{array}{l}(\boldsymbol{\nu} attach, \ldots)\\ (\overline{rel_1}.\mathbf{0} \mid Router_1 \mid \overline{rel_2}.\mathbf{0} \mid Router_2 \mid Player)\end{array} \quad \precsim_{\mathcal{I}} \quad !\,v(video).t[\underline{6}].\overline{play}\langle video\rangle.\mathbf{0}$$

を示せばよい.

3.5　関連研究

Chen による π 計算に対する時間拡張 [Che04] では，選択演算子に対して保存される弱双模倣関係を定義し，弱合同性に対する完全な証明系を構築している．プロセスの振舞い等価性としての双模倣関係は与えられているが，動作のタイミングの違いを表す順序関係については考えられていない.

CCS に対する時間拡張 [ST92, Che92a, Yi91a] では，時間動作について拡張された双模倣関係について議論している．拡張の方法は我々の時間付き双模倣関係と同様に，入出力動作や内部動作だけでなく時間動作についても互いに模倣できる必要がある．CCS ではプロセス間のリンクをメッセージとする通信はできないため，強双模倣関係はすべての演算子に対して保存される．τ 遷移を抽象した弱双模倣関係は選択演算子に対して保存されない．CCS の弱双模倣関係にも同様の問題が存在する [Mil89]．この問題に対応するため，[Che92a, Yi91a] では選択演算子に対しても保存される弱双模倣関係の部分集合と等価である新しい双模倣関係について議論している．[ST92] では弱双模倣関係の条件

$$P \xrightarrow{\alpha} P' \ \Rightarrow \exists Q'.\, Q \xrightarrow{\widehat{\alpha}}_{\tau} Q' \wedge (P', Q') \in \mathcal{R}$$

を

$$P \xrightarrow{\alpha} P' \ \Rightarrow \exists Q'.\, Q \xrightarrow{\alpha}_{\tau} Q' \wedge (P', Q') \in \mathcal{R}$$

とし，τ 遷移を最初の一回だけ抽象しないようにすることで合同性が成り立つことを示している．

プロセスの “速さ” に関する研究として，[AKH92] では CCS を，[BAK96] では π 計算を対象に τ 遷移の回数を数え，回数が少ないほど速いプロセスであると考える Efficiency Preorder が提案され，双模倣の考えに基づく定義がなされている．時間動作を直接扱ってはいないが，τ 遷移の回数による順序付けは，時間経過の回数による順序付けを行う我々の手法と類似している．

時間拡張された CCS の体系においても時間動作に着目してプロセスの “速さ” を表す順序関係が提案されている．これらの関係では，入出力動作については双模倣である点は共通しており，時間差をどのように表現するかという点に特徴が見られる．時間差の表現方法として，ある入出力動作とその次の入出力動作の間の経過時間に着目した関係 [MT91] と，すべての入出力動作について初期状態からその動作を実行するまでに経過した時間に着目した関係 [Sat98] がある．また，これら観点の異なる関係の間の関連について [LV01, LV04b, LV04c, LV04a, LV05] で議論されている．

[MT91] の方法は我々の遅延時間順関係と類似しているが，全体の時間経過量は同じである点が異なる．[Sat98] では入出力動作を保存したまま “遅い” プロセスを “速い” プロセスに変換するための代数的枠組みを与えている点が特徴的である．

3.6　おわりに

本章では，実時間並行計算の動作検証の基礎技法として，時間付き π 計算のプロセス間の等価関係を表す時間付き双模倣関係と，擬順序関係を表す遅延時間順関係を提案した．時間付き双模倣関係はいずれのプロセスも他方のプロセスの動作を模倣できることを表す．時間付き双模倣なプロセスは入出力動作や内部動作だけでなく時間経過についても互いに模倣する．時間経過を模倣するため，入出力動作や内部動作がまったく同じタイミングで発生する．遅延時間順関係は，入出力動作や内部動作は同じであるが，動作間で経過する時間の長さが一方のプロセスの方が常に他方のプロセスよりも短いことを表す．つまり，一方のプロセスが常に速いタイミングで動作を実行する．

プロセス間の相互作用という観点からプロセスの動作を調べる時はプロセスの内部動作は無視したい．そこで，時間付き双模倣関係と遅延時間順関係の双方に対して内部動作を表す τ 遷移を抽象した関係を定義した．また，内部動作を考慮した場合およびしない場合のいずれにおいても，時間付き双模倣関係は入力プレフィックスを除く演算子，遅延時間順関係は入力プレフィックスおよび時間独立でないプロセスの並行合成を除く演算子に対して保存されることを示した．

弱遅延時間順関係は，実時間システムが時間制約の一つであるデッドラインを破らないことを検証するために利用できる．システムの入出力動作とデッドラインに関する仕様を $Spec$，システムの実装を $Impl$ とし，$Impl \precsim_{\mathcal{T}} Spec$ が成り立てば，実装が仕様よりも速いタイミングで動作することが示される．つまり，デッドラインに関する仕様は満たされていることがわかる．

第4章　時間動作の抽象化

4.1　はじめに

我々のπ計算に対する時間拡張はプロセス代数に対して行われている時間拡張[NS92, BM00]に基づいており，時間に関して非常に細かい意味論を提供する．そのため，時間付き強双模倣関係および遅延時間順関係を利用してシステムの時間に関する動作を細かく調べることが可能である．デッドラインなどの時間制約に関係する性質の解析に有用である．

しかし，経過時間の長さがあまり意味を持たない性質を調べるにはあまり適していない．時間をモデル化することで状態をさらに細かく分類するため，時間を考慮しないモデルに比べて状態爆発の問題が深刻になる．示したい性質を保存したまま時間に関して抽象化を行い，モデルの状態数を減らす手法が必要である．

本章では，時間付きπ計算のプロセスを時間に関して抽象化する手法を提案する．提案する抽象化手法は，タイムアウトの発生から次のタイムアウトの発生までの間の時間経過を抽象する．可能動作不変性により，プロセスが実行可能な入出力動作や内部動作はタイムアウトが発生するまで変化しない．時間動作を抽象化することで入出力動作，内部動作およびタイムアウトの発生のみに着目した振舞い解析が行えるようになる．

本章の構成は以下の通りである．4.2節では時間動作の抽象化を行う対象として時間付きπ計算を拡張する．4.3節では拡張された体系においてタイムアウトにも着目した双模倣関係を定義する．4.4節で展開定理を示し，4.5節で時間動作の抽象化手法を提案する．4.6節では抽象化手法の性質として抽象化を行う前後のプロセス間の関係について述べる．最後に関連研究を挙げてまとめを述べる．

4.2　対象言語

時間動作の抽象化を行うため，第2章で述べた時間付きπ計算を拡張する．入出力動作に対し，その動作が実行可能になってから実行されるまでの経過時間を

数えられるようにする．

自然数を表す名前の集合 $\mathcal{I}$ 上の演算 $+, \dot{-}, <, \leq$ を定義する．

定義 4.1 $i_m, i_n \in \mathbb{N}$，$\underline{i_m}, \underline{i_n} \in \mathcal{I}$ に対し

$$\begin{aligned}
\underline{i_m} + \underline{i_n} &\stackrel{def}{=} \underline{i_m + i_n} \\
\underline{i_m} \dot{-} \underline{i_n} &\stackrel{def}{=} \begin{cases} \underline{i_m - i_n} & \text{if } i_m \geq i_n \\ \underline{0} & \text{otherwise} \end{cases} \\
\underline{i_m} < \underline{i_n} &\stackrel{def}{=} i_m < i_n \\
\underline{i_m} \leq \underline{i_n} &\stackrel{def}{=} i_m \leq i_n
\end{aligned}$$

□

拡張はプレフィックスとその意味論に対して行う．

定義 4.2 拡張された時間付き π 計算のプロセス式は以下の構文によって定義される．ここで $d \in \mathcal{I}$ である．

$$\begin{aligned}
\pi &::= x(y)@d \mid \overline{x}\langle z\rangle@d \mid \tau@d \mid t[n] \mid [x=y]\pi \\
P &::= M \mid P \mid P \mid \boldsymbol{\nu} x\, P \mid !\, P \\
M &::= \mathbf{0} \mid \pi.P \mid M + M
\end{aligned}$$

□

並行，選択，制限，複製の各演算子は定義 2.1 と同じである．拡張された時間付き π 計算のプロセス全体の集合を $\mathcal{EP}$ と書く．

定義 4.3 $\mathcal{EP}$ 上の遷移関係 $\{\stackrel{\alpha}{\rightarrow} \mid \alpha \in Act \cup \{\bullet\}\} \cup \{\rightarrow\}$ は図 *4.1* および図 *4.2* の各規則，図 *2.1* の各規則，図 *2.2* の *T-SUM-L*，*SUM-L*，*T-PAR-L*，*PAR-L*，*COMM-L*，*CLOSE-L* の各規則とそれぞれにおいて P と Q を入れ替えた *T-SUM-R*，*SUM-R*，*T-PAR-R*，*PAR-R*，*COMM-R*，*CLOSE-R* の各規則，図 *2.3* の $PASS_{\mathrm{T}}$，$INACT_{\mathrm{T}}$，SUM_{T}，PAR_{T}，RES_{T}，REP_{T} の各規則によって定義される．□

直観的には，$\overline{x}\langle z\rangle@d.P$ は動作 $\overline{x}\langle z\rangle$ が実行できるようになってから経過した時間の長さを d によって数える．例えば，$\overline{x}\langle z\rangle@d.t[d].P$ は $\overline{x}\langle z\rangle$ を実行後に，実行開始から $\overline{x}\langle z\rangle$ が実行されるまでに経過したのと同じ長さの時間待ちを行う．2 単位時間後に $\overline{x}\langle z\rangle$ が実行されたとすると遷移は以下のようになる．

$$\begin{aligned}
\overline{x}\langle z\rangle@d.t[d].P &\rightarrow \overline{x}\langle z\rangle@d.(t[d].P)\{d + \underline{1}/d\} \\
&\rightarrow \overline{x}\langle z\rangle@d.(t[d].P)\{d + \underline{2}/d\}
\end{aligned}$$

$$\text{OUT}: \frac{}{\overline{x}\langle z\rangle@d.P \xrightarrow{\overline{x}\langle z\rangle} P\{\underline{0}/d\}}$$

$$\text{IN}: \frac{}{x(y)@d.P \xrightarrow{x(z)} P\{z/y\}\{\underline{0}/d\}}$$

$$\text{TAU}: \frac{}{\tau@d.P \xrightarrow{\tau} P\{\underline{0}/d\}}$$

$$\text{MATCH}: \frac{\pi.P \xrightarrow{\alpha} P'}{[x=x]\pi.P \xrightarrow{\alpha} P'}$$

図 4.1: 拡張された遷移規則

$$\text{OUT}_\text{T}: \frac{}{\overline{x}\langle z\rangle@d.P \rightarrow \overline{x}\langle z\rangle@d.P\{d+\underline{1}/d\}}$$

$$\text{IN}_\text{T}: \frac{}{x(y)@d.P \rightarrow x(y)@d.P\{d+\underline{1}/d\}}$$

$$\text{N-MAT}_\text{T}: \frac{}{[x=y]\pi.P \rightarrow [x=y]\pi.P}\ x \neq y$$

$$\text{P-MAT}_\text{T}: \frac{\pi.P \rightarrow P'}{[x=x]\pi.P \rightarrow P'}$$

図 4.2: 拡張された時間経過規則

$$
\begin{aligned}
&\overset{\overline{x}\langle z\rangle}{\longrightarrow} && (t[d].P)\{d+\underline{2}/d\}\{\underline{0}/d\}\\
&= && t[\underline{2}].P\{\underline{2}/d\}\\
&\rightharpoonup && t[\underline{1}].P\{\underline{2}/d\}\\
&\rightharpoonup && t[\underline{0}].P\{\underline{2}/d\}\\
&\overset{\bullet}{\longrightarrow} && P\{\underline{2}/d\}
\end{aligned}
$$

名前の束縛と代入は次のように変更する．

定義 4.4 プロセス $x(y)@d.P$ における名前 y, d，$\overline{x}\langle z\rangle@d.P$，$\tau@d.P$ における名前 d のスコープは P に制限される． □

$x(y)@d.P$ や $\overline{x}\langle z\rangle@d.P$, $\tau@d.P$ において $d \notin \mathsf{fn}(P)$ の場合，それぞれ $x(y).P$, $\overline{x}\langle z\rangle.P$, $\tau.P$ とも書く．

定義 4.5 $x(y)@d.P$，$\overline{x}\langle z\rangle@d.P$，$\tau@d.P$ に対する代入の適用を以下のように定義する．

$$
\begin{aligned}
(x(y)@d.P)\sigma &\overset{def}{=} x\sigma(y)@d.P\sigma_{-\{y,d\}}\\
(\overline{x}\langle z\rangle@d.P)\sigma &\overset{def}{=} \overline{x\sigma}\langle z\sigma\rangle@d.P\sigma_{-\{d\}}\\
(\tau@d.P)\sigma &\overset{def}{=} \tau@d.P\sigma_{-\{d\}}
\end{aligned}
$$ □

構造合同関係には比較演算子 $[x = y]$ に関する公理を追加する．

定義 4.6 $\mathcal{EP}$ に含まれるプロセスの構造合同関係は，定義 *2.4* の公理と $\equiv_\alpha$，および以下の公理を満たす最小の合同関係である．

$$[x = x]\pi.P \equiv \pi.P$$ □

以下では，M_i, M_i', N_j, N_j' は比較演算 $[x_1 = y_1][x_2 = y_2]\cdots[x_n = y_n]$ を表す．すべての $i \in \{1 \ldots n\}$ に対して $x_i = y_i$ の時 true，true でなければ false と書く．

拡張された時間付き π 計算のプロセス式に対する型システムは，図 2.5 の T-BASE，T-NAME，T-NAT，T-PAR，T-SUM，T-RES，T-TIME，T-INACT の各規則と，図 4.3 の各規則からなる．型システムによって，型付け可能なプロセスでは $\alpha@d$ の名前 d が時間を数えるための名前であることが保証される．また，型付け可能はプロセスは何らかの動作によって遷移した後も型付け可能なままである．

$$\text{T-TAU}: \frac{\Gamma,\, d:I \vdash P:\diamond}{\Gamma \vdash \tau@d.P:\diamond}$$

$$\text{T-IN}: \frac{\Gamma \vdash x:\sharp T \quad \Gamma,\, y:T,\, d:I \vdash P:\diamond}{\Gamma \vdash x(y)@d.P:\diamond}$$

$$\text{T-OUT}: \frac{\Gamma \vdash x:\sharp T \quad \Gamma \vdash z:T \quad \Gamma,\, d:I \vdash P:\diamond}{\Gamma \vdash \overline{x}\langle z\rangle@d.P:\diamond}$$

$$\text{T-MATCH-L}: \frac{\Gamma \vdash x:L \quad \Gamma \vdash y:L \quad \Gamma \vdash \pi.P:\diamond}{\Gamma \vdash [x=y]\pi.P:\diamond}$$

$$\text{T-MATCH-I}: \frac{\Gamma \vdash x:I \quad \Gamma \vdash y:I \quad \Gamma \vdash \pi.P:\diamond}{\Gamma \vdash [x=y]\pi.P:\diamond}$$

図 4.3: 拡張された型付け規則

4.3 双模倣関係

第 3 章で定義した時間付き双模倣関係は，タイムアウトを抽象する強遷移関係，弱遷移関係に基づいている．そのため，タイムアウトに着目してプロセスの振舞いを解析することができない．時間動作の抽象化をタイムアウトに着目して行うため，タイムアウトを抽象しない双模倣関係を定義する．

定義 4.7 以下を満たす対称な関係 $\mathcal{R}$ を時間付き詳細双模倣関係と呼ぶ．
$(P,Q) \in \mathcal{R}$ の時，

- $P \stackrel{\alpha}{\rightarrow\!\!\!\!\!\!\rightarrow} P' \;\Rightarrow\; \exists Q'.\; Q \stackrel{\alpha}{\rightarrow\!\!\!\!\!\!\rightarrow} Q' \wedge (P',Q') \in \mathcal{R}$
- $P \rightharpoonup P'' \;\Rightarrow\; \exists Q''.\; Q \rightharpoonup Q'' \wedge (P'',Q'') \in \mathcal{R}$ □

$(P,Q) \in \mathcal{R}$ なる時間付き詳細双模倣関係 $\mathcal{R}$ が存在する時 $P \sim_{\mathcal{T}}^{f} Q$ と書く．

定義 4.8 $\sim_{\mathcal{T}}^{f} \;=\; \bigcup\{\mathcal{R} \mid \mathcal{R}$ は時間付き詳細双模倣関係 $\}$ □

命題 4.1 $\sim_{\mathcal{T}}^{f}$ は最大の時間付き詳細双模倣関係である． □

命題 4.2 $\sim_{\mathcal{T}}^{f}$ は等価関係である． □

時間付き詳細双模倣関係は，時間付き強双模倣関係と同様に入力プレフィックスに対して保存されない．入力プレフィックスを含むすべての演算子に対して保存される時間付き詳細双模倣関係の部分集合を求める．

定義 4.9 任意の代入 σ に対して $P\sigma \sim_{\mathcal{T}}^{f} Q\sigma$ であるような P と Q の関係を時間付き完全詳細双模倣関係と呼び，$P \sim_{\mathcal{T}}^{fc} Q$ と書く．□

定義より明らかに $\sim_{\mathcal{T}}^{fc} \subseteq \sim_{\mathcal{T}}^{f} \subseteq \sim_{\mathcal{T}}$ である．

補題 4.1 $z \in \mathsf{fn}(P) \cup \mathsf{fn}(Q) \cup \{y\}$ なるすべての z に対し $P\{z/y\} \sim_{\mathcal{T}}^{f} Q\{z/y\}$ ならば $x(y).P \sim_{\mathcal{T}}^{f} x(y).Q$ である．□

定義 4.10 コンテキスト $C[\cdot]$ を以下のように定義する．

$$C[\cdot] \;::=\; [\cdot] \mid \pi.C[\cdot] \mid \pi.C[\cdot] + R \mid C[\cdot] \mid S \mid \boldsymbol{\nu} x\, C[\cdot] \mid !\, C[\cdot]$$

ここで R はガード付きプロセス，S は任意のプロセスとする．□

定理 4.1 $P \sim_{\mathcal{T}}^{fc} Q$ の時，任意のコンテキスト $C[\cdot]$ に対し $C[P] \sim_{\mathcal{T}}^{fc} C[Q]$ である．

証明： コンテキストの構造に関する帰納法による．$C[\cdot] = [\cdot]$ の時，仮定より明らかに $C[P] \sim_{\mathcal{T}}^{fc} C[Q]$ である．以下では $C[\cdot] = x(y).C'[\cdot]$ の場合を示す．帰納法の仮定より $C'[P] \sim_{\mathcal{T}}^{fc} C'[Q]$，つまり任意の代入 σ に対し $C'[P]\sigma \sim_{\mathcal{T}}^{f} C'[Q]\sigma$．$z \in \mathsf{fn}(C'[P]\sigma) \cup \mathsf{fn}(C'[Q]\sigma) \cup \{y\}$ に対し $C'[P]\sigma\{z/y\} \sim_{\mathcal{T}}^{f} C'[Q]\sigma\{z/y\}$ とできて，補題 4.1 より $x(y).C'[P]\sigma \sim_{\mathcal{T}}^{f} x(y).C'[Q]\sigma$，つまり $C[P]\sigma \sim_{\mathcal{T}}^{f} C[Q]\sigma$ ゆえ $C[P] \sim_{\mathcal{T}}^{c} C[Q]$．□

4.4　展開定理

時間付き詳細双模倣関係のもとで成り立つ展開定理は，複数のサブプロセスの並行合成によって構成されるプロセスに対して，振舞い等価な単一のプロセスが存在することを表す．例えば，通信可能な 2 つのプロセスが並行に動作する場合，それぞれのプロセスが個々に動作するか，あるいは通信するかいずれかの動作を行う．

$$\overline{x}.\mathbf{0} \mid x.\mathbf{0} \;\sim_{\mathcal{T}}^{f}\; \overline{x}.x.\mathbf{0} + x.\overline{x}.\mathbf{0} + \tau.\mathbf{0}$$

並行プロセスに対して，次に実行可能な動作と実行後の状態をすべて列挙することができる．

定義 4.2 の構文の定義と並行合成演算子に関する構造合同関係から，並行合成されるプロセスは一般的に $\sum_{i\in I} M_i\alpha_i@d_i.P_i + \sum_{j\in J} N_j t[n_j].Q_j$ と表すことができる．ここで，$I=\{1,\ldots,n\}$ とする時 $\sum_{i\in I} P_i$ は $P_1+\cdots+P_n$ の略記である．

定理 4.2 $P=\sum_i M_i\alpha_i@d_i.P_i+\sum_i M_i't[n_i].P_i'$, $Q=\sum_j N_j\beta_j@e_j.Q_j+\sum_j N_j't[m_j].Q_j'$ とする．この時,

$$\begin{aligned} P\,|\,Q \;\sim_T^{fc}\;& \textstyle\sum_i M_i\alpha_i@d_i.(P_i\,|\,\sum_j N_j\beta_j@e_j.Q_j\{e_j{+}d_i/e_j\}+\sum_j N_j't[m_j\dot{-}d_i].Q_j') \\ &+\textstyle\sum_j N_j\beta_j@e_j.(\sum_i M_i\alpha_i@d_i.P_i\{d_i{+}e_j/d_i\}+\sum_i M_i't[n_i\dot{-}e_j].P_i'\,|\,Q_j) \\ &+\textstyle\sum_i\sum_j M_iN_j[x=y]\tau.R_{ij} \\ &+\textstyle\sum_i M_i't[n_i].(P_i'\,|\,\sum_j N_j\beta_j@e_j.Q_j\{e_j{+}n_i/e_j\}+\sum_j N_j't[m_j\dot{-}n_i].Q_j') \\ &+\textstyle\sum_j N_j't[m_j].(\sum_i M_i\alpha_i@d_i.P_i\{d_i{+}m_j/d_i\}+\sum_i M_i't[n_i\dot{-}m_j].P_i'\,|\,Q_j') \end{aligned}$$

ここで R_{ij} は α_i,β_j の組み合わせによる以下のいずれかである．

α_i	β_j	R_{ij}
$\overline{x}\langle v\rangle$	$y(u)$	$P_i\{\underline{0}/d_i\}\,\|\,Q_j\{v/u\}\{\underline{0}/e_j\}$
$\overline{x}(v)$	$y(v)$	$(\boldsymbol{\nu}v)\,(P_i\{\underline{0}/d_i\}\,\|\,Q_j\{\underline{0}/e_j\})$
$x(u)$	$\overline{y}\langle v\rangle$	$P_i\{v/u\}\{\underline{0}/d_i\}\,\|\,Q_j\{\underline{0}/e_j\}$
$x(v)$	$\overline{y}(v)$	$(\boldsymbol{\nu}v)\,(P_i\{\underline{0}/d_i\}\,\|\,Q_j\{\underline{0}/e_j\})$

証明： はじめに記法をいくつか導入する．

$$\begin{aligned} P(l) &= \textstyle\sum_i M_i\alpha_i@d_i.P_i\{d_i+l/d_i\}+\sum_i M_i't[n_i\dot{-}l].P_i' \\ Q(l) &= \textstyle\sum_j N_j\beta_j@e_j.Q_j\{e_j+l/e_j\}+\sum_j N_j't[m_j\dot{-}l].Q_j' \end{aligned}$$

とする．$l=\underline{i_l}$ の時，$P(l)$，$Q(l)$ はそれぞれ P，Q から i_l 単位時間経過した後のプロセスを表す．また,

$$\begin{aligned} S_{P_1}(l) &= \textstyle\sum_i M_i\alpha_i@d_i.(P_i\{d_i{+}l/d_i\} \\ &\qquad\qquad \textstyle\,|\,\sum_j N_j\beta_j@e_j.Q_j\{e_j{+}d_i{+}l/e_j\}+\sum_j N_j't[m_j\dot{-}d_i\dot{-}l].Q_j') \\ S_{Q_1}(l) &= \textstyle\sum_j N_j\beta_j@e_j.(\sum_i M_i\alpha_i@d_i.P_i\{d_i{+}e_j{+}l/d_i\}+\sum_i M_i't[n_i\dot{-}e_j\dot{-}l].P_i' \\ &\qquad\qquad \textstyle\,|\,Q_j\{e_j{+}l/e_j\}) \\ S_R(l) &= \textstyle\sum_i\sum_j M_iN_j[x=y]\tau.R_{ij} \end{aligned}$$

$$S_{P_2}(l) = \sum_i M_i' t[n_i \dot{-} l].(P_i' \mid \sum_j N_j \beta_j @ e_j.Q_j\{e_j{+}n_i/e_j\} + \sum_j N_j' t[m_j \dot{-} n_i].Q_j')$$
$$S_{Q_2}(l) = \sum_j N_j' t[m_j \dot{-} l].(\sum_i M_i \alpha_i @ d_i.P_i\{d_i{+}m_j/d_i\} + \sum_i M_i' t[n_i \dot{-} m_j].P_i' \mid Q_j')$$

とする．これらの記法を用いると展開定理は

$$P(\underline{0}) \mid Q(\underline{0}) \sim_{\mathcal{T}}^{fc} S_{P_1}(\underline{0}) + S_{Q_1}(\underline{0}) + S_R(\underline{0}) + S_{P_2}(\underline{0}) + S_{Q_2}(\underline{0})$$

と表される．

以下では，

$$\mathcal{R} = \{(P(l) \mid Q(l), S_{P_1}(l) + S_{Q_1}(l) + S_R(l) + S_{P_2}(l) + S_{Q_2}(l)) \mid \underline{0} \le l \le l_\tau\} \cup Id$$

とした時に $\mathcal{R} \subseteq \sim_{\mathcal{T}}^{fc}$ であることを示す．ここで，

$$l_\tau = \begin{cases} \underline{0} & \text{if } \exists i.(M_i = \mathsf{true} \wedge \alpha_i = \tau) \vee \exists j.(N_j = \mathsf{true} \wedge \beta_j = \tau) \\ & \quad \vee \exists i,j.(M_i = \mathsf{true} \wedge N_j = \mathsf{true} \wedge x = y) \\ \min(\{n_i\} \cup \{m_j\}) & \text{otherwise} \end{cases}$$

である．ただし $\min(\emptyset) = \infty$ とする．

- $P(l) \mid Q(l) \stackrel{\alpha}{\rightarrow} P' \mid Q(l)$ (ただし $P(l) \stackrel{\alpha}{\rightarrow} P'$) の場合，
  - $\alpha \neq \bullet$ ならば，$M_i = \mathsf{true}$ かつ $\alpha_i = \alpha$ かつ $P' = P_i\{l/d_i\}$ なる i が存在し，$M_i' = \mathsf{true}$ かつ $n_i \dot{-} l = \underline{0}$ なる i は存在しない．この時，$S_{P_1}(l) \stackrel{\alpha}{\rightarrow} P_i\{l/d_i\} \mid Q(l) = P' \mid Q(l)$ であるので，$(P' \mid Q(l), P' \mid Q(l)) \in Id \subseteq \mathcal{R}$ である．
  - $\alpha = \bullet$ ならば，$M_i' = \mathsf{true}$，$n_i = l$，$P' = P_i'$ なる i が存在する．この時，$S_{P_2}(l) \stackrel{\bullet}{\rightarrow} P_i' \mid Q(n_i)$ であるので，$(P_i' \mid Q(n_i), P_i' \mid Q(n_i)) \in Id \subseteq \mathcal{R}$ である．
- $P(l) \mid Q(l) \stackrel{\alpha}{\rightarrow} P(l) \mid Q'$ (ただし $Q(l) \stackrel{\alpha}{\rightarrow} Q'$) の場合，同様．
- $P(l) \mid Q(l) \stackrel{\tau}{\rightarrow} R$ の場合，α_i と β_j のチャネル名が同じなるような i および j が存在する．この時 $l = \underline{0}$ である．以下では $x = y$ とする．
  - $(\alpha_i, \beta_j) = (\overline{x}\langle v \rangle, y(u))$ ならば，$R = P_i\{\underline{0}/d_i\} \mid Q_j\{v/u\}\{\underline{0}/e_j\}$ である．また，$S_R(l) \stackrel{\tau}{\rightarrow} P_i\{\underline{0}/d_i\} \mid Q_j\{v/u\}\{\underline{0}/e_j\} = R$ であるため $(R, R) \in Id \subseteq \mathcal{R}$ である．

– $(\alpha_i, \beta_j) = (\overline{x}(v), y(v))$，$(x(u), \overline{x}\langle v\rangle)$，$(x(v), \overline{x}\langle v\rangle)$ の場合も同様である．

- $P(l) \,|\, Q(l) \rightharpoonup P(l+1) \,|\, Q(l+1)$ の場合，最大進行性より $l+1 \leq l_\tau$ である．この時 $l_\tau = \min(\{n_i\} \cup \{m_j\})$ であるので，任意の i, j に対して $\underline{1} \leq n_i \dot{-} l$，$\underline{1} \leq m_j \dot{-} l$ であり，$S_{P_2}(l) \rightharpoonup S_{P_2}(l+1)$ および $S_{Q_2}(l) \rightharpoonup S_{Q_2}(l+1)$ とできる．また，明らかに $S_{P_1}(l) \rightharpoonup S_{P_1}(l+1)$，$S_{Q_1}(l) \rightharpoonup S_{Q_1}(l+1)$ であり，$P(l)$ と $Q(l)$ が通信できないことからチャネル名が同じ α_i, β_j が存在しないため $S_R(l) \rightharpoonup S_R(l+1)$ とできる．以上より $(P(l+1) \,|\, Q(l+1), S_{P_1}(l+1) + S_{Q_1}(l+1) + S_R(l+1) + S_{P_2}(l+1) + S_{Q_2}(l+1)) \in \mathcal{R}$ である．

- 他の場合も同様に示される．

ここまでで $\mathcal{R} \subseteq \sim_{\mathcal{T}}^{f}$ であることが示された．また，$\mathcal{R} \subseteq \sim_{\mathcal{T}}^{f}$ であることと，$x \neq y$ ならば $[x = y]\pi.P \sim_{\mathcal{T}}^{f} \mathbf{0}$ であることから $\mathcal{R} \subseteq \sim_{\mathcal{T}}^{fc}$ は容易に導かれる． □

複製演算子に対する展開定理を以下に示す．

定理 4.3 $P = \sum_i M_i \alpha_i @ d_i.P_i + \sum_i M_i' t[n_i].P_i'$ とする．この時，

$$
\begin{aligned}
!\,P \;\sim_{\mathcal{T}}^{fc}\; & \textstyle\sum_i M_i \alpha_i @ d_i.(P_i \,|\, !\,(\sum_j M_j \alpha_j @ d_j.P_j\{d_j + d_i/d_j\} + \sum_j M_j' t[n_j \dot{-} d_i].P_j')) \\
& + \textstyle\sum_i \sum_j M_i M_j [x = y]\tau.(R_{ij} \,|\, !\,P) \\
& + \textstyle\sum_i M_i' t[n_i].(P_i' \,|\, !\,(\sum_j M_j \alpha_j @ d_j.P_j\{d_j + n_i/d_j\} + \sum_j M_j' t[n_j \dot{-} n_i].P_j'))
\end{aligned}
$$

ここで R_{ij} は α_i, α_j の組み合わせによる以下のいずれかである．

α_i	α_j	R_{ij}
$\overline{x}\langle v\rangle$	$y(u)$	$P_i\{\underline{0}/d_i\} \,|\, P_j\{v/u\}\{\underline{0}/e_j\}$
$\overline{x}(v)$	$y(v)$	$(\boldsymbol{\nu} v)\,(P_i\{\underline{0}/d_i\} \,|\, P_j\{\underline{0}/e_j\})$
$x(u)$	$\overline{y}\langle v\rangle$	$P_i\{v/u\}\{\underline{0}/d_i\} \,|\, P_j\{\underline{0}/e_j\}$
$x(v)$	$\overline{y}(v)$	$(\boldsymbol{\nu} v)\,(P_i\{\underline{0}/d_i\} \,|\, P_j\{\underline{0}/e_j\})$

証明： 定理 4.2 と同様． □

並行合成の展開の例を以下に示す．

$$t[\underline{1}].x.\mathbf{0} \mid t[\underline{1}].y.\mathbf{0}$$

を展開定理に従って展開すると

$$t[\underline{1}].(x.\mathbf{0} \,|\, t[\underline{0}].y.\mathbf{0}) + t[\underline{1}].(t[\underline{0}].x.\mathbf{0} \,|\, y.\mathbf{0})$$

が得られる．例えば，$\xrightarrow{\bullet}\xrightarrow{\bullet}\xrightarrow{x}$ という遷移に対して

$$
\begin{array}{lll}
t[\underline{1}].x.\mathbf{0} \,|\, t[\underline{1}].y.\mathbf{0} & \xrightarrow{\bullet}\xrightarrow{\bullet}\xrightarrow{x} & \mathbf{0} \,|\, y.\mathbf{0} \\
t[\underline{1}].(x.\mathbf{0} \,|\, t[\underline{0}].y.\mathbf{0}) + t[\underline{1}].(t[\underline{0}].x.\mathbf{0} \,|\, y.\mathbf{0}) & \xrightarrow{\bullet}\xrightarrow{\bullet}\xrightarrow{x} & \mathbf{0} \,|\, y.\mathbf{0}
\end{array}
$$

となり，遷移先のプロセスは同じである．他の遷移についても双方のプロセスは対応付けられる．

4.5　時間経過動作の抽象化

4.5.1　並行合成の展開

時間経過動作の抽象化は，展開定理に基づいて元のプロセス式の並行合成演算子と複製演算子を展開して得られるプロセス式に対して行う．展開定理から展開前後のプロセスは時間付き詳細双模倣である．

定義 4.11 時間付き π 計算のプロセス式に対し並行合成演算子および複製演算子の展開を行う関数 $[\![\cdot]\!]_e$ を以下のように定義する．

$$
\begin{array}{rcl}
[\![\mathbf{0}]\!]_e & \stackrel{def}{=} & \mathbf{0} \\
[\![\pi.P]\!]_e & \stackrel{def}{=} & \pi.P \\
[\![P_1{+}P_2]\!]_e & \stackrel{def}{=} & [\![P_1]\!]_e + [\![P_2]\!]_e \\
[\![\boldsymbol{\nu} x\, P]\!]_e & \stackrel{def}{=} & \boldsymbol{\nu} x\, [\![P]\!]_e \\
[\![P_a \,|\, P_b]\!]_e & \stackrel{def}{=} & \sum_i M_i\alpha_i@d_i.(P_i \,|\, \sum_j N_j\beta_j@e_j.Q_j\{e_j{+}d_i/e_j\} + \sum_j N'_j t[m_j \dot{-} d_i].Q'_j) \\
& & \quad + \sum_j N_j\beta_j@e_j.(\sum_i M_i\alpha_i@d_i.P_i\{d_i{+}e_j/d_i\} + \sum_i M'_i t[n_i \dot{-} e_j].P'_i \,|\, Q_j) \\
& & \quad + \sum_i \sum_j M_i N_j [x=y]\tau.R_{ij} \\
& & \quad + \sum_i M'_i t[n_i].(P'_i \,|\, \sum_j N_j\beta_j@e_j.Q_j\{e_j{+}n_i/e_j\} + \sum_j N'_j t[m_j \dot{-} n_i].Q'_j) \\
& & \quad + \sum_j N'_j t[m_j].(\sum_i M_i\alpha_i@d_i.P_i\{d_i{+}m_j/d_i\} + \sum_i M'_i t[n_i \dot{-} m_j].P'_i \,|\, Q'_j) \\
[\![!\, P_a]\!]_e & \stackrel{def}{=} & \sum_i M_i\alpha_i@d_i.(P_i \\
& & \qquad\qquad |\, !\, (\sum_j M_j\alpha_j@d_j.P_j\{d_j + d_i/d_j\} + \sum_j M'_j t[n_j \dot{-} d_i].P'_j)) \\
& & \quad + \sum_i \sum_j M_i M_j [x=y]\tau.(R'_{ij} \,|\, !\, P_a) \\
& & \quad + \sum_i M'_i t[n_i].(P'_i \\
& & \qquad\qquad |\, !\, (\sum_j M_j\alpha_j@d_j.P_j\{d_j + n_i/d_j\} + \sum_j M'_j t[n_j \dot{-} n_i].P'_j))
\end{array}
$$

ただし，$[\![P_a]\!]_e = \sum_i M_i\alpha_i@d_i.P_i + \sum_i M_i't[n_i].P_i'$ および $[\![P_b]\!]_e = \sum_j N_j\beta_j@e_j.Q_j + \sum_j N_j't[m_j].Q_j'$ とする．また R_{ij} は定理 *4.2* における R_{ij}，R'_{ij} は定理 *4.3* における R_{ij} である． □

補題 4.2 任意のプロセス式 $P \in \mathcal{EP}$ に対し $P \sim_{\mathcal{T}}^{fc} [\![P]\!]_e$ である．

証明： P の構造に関する帰納法と定理 4.2，4.3 による． □

4.5.2 時間経過動作の抽象

タイムアウトが発生してから次のタイムアウトが発生するまでの時間経過を抽象する．例えば，$a.P + t[\underline{5}].Q$ は 4 単位時間以内に動作 a が発生すれば P へ，動作 a が発生せず 5 単位時間経過すればタイムアウト動作を行って Q へ遷移する．この時，実行開始から 4 単位時間経過するまでは動作 a を実行するか時間経過するかのいずれかの可能性しか存在しない．そこで，$a.P + t.Q$ のように時間経過動作によるタイムアウトを通常の動作 t として抽象化することを考える．Q へ遷移するまでに経過する時間の長さはわからなくなるが，動作 a によって P へ遷移するか，あるいは時間経過によって Q へ遷移することは表現されている．

展開定理によりプロセス式は一般的に $(\boldsymbol{\nu}\tilde{z})\,(\sum_i M_i\alpha_i@d_i.P_i + \sum_j N_j t[n_j].Q_j)$ のように逐次プロセスの形で表現可能である．そこで，以下では逐次プロセスを対象として時間経過動作の抽象化を定義する．任意のプロセス式に対して関数 $[\![\cdot]\!]_e$ を適用することで，振舞い等価な単一のプロセスによる表現を得られる．

定義 4.12 並行プロセスではない単一のプロセスを表すプロセス式 $P \in \mathcal{EP}$ に対し時間経過動作の抽象化 $[\![P]\!]_t$ を以下のように定義する．

$$
\begin{aligned}
&[\![\mathbf{0}]\!]_t \stackrel{def}{=} \mathbf{0}\\
&[\![\boldsymbol{\nu} x\, P]\!]_t \stackrel{def}{=} \boldsymbol{\nu} x\, [\![P]\!]_t\\
&[\![\textstyle\sum_{i\in I} M_i\alpha_i@d_i.P_i + \sum_{j\in J} N_j t[n_j].Q_j]\!]_t\\
&\quad\stackrel{def}{=}
\begin{cases}
\sum_{j\in J'} N_j t.Q_j & \text{if } \exists j.(N_j = \mathsf{true} \wedge n_j = \underline{0})\\
\sum_{i\in I} M_i\alpha_i.P_i\{\underline{0}/d_i\} & \text{if } \forall j.(N_j = \mathsf{false} \vee \underline{0} < n_j) \wedge\\
& \quad \exists i.(M_i = \mathsf{true} \wedge \alpha_i = \tau)\\
\sum_{i\in I} M_i\alpha_i.P_i\{\underline{0}/d_i\} + \sum_{j\in J''} N_j t.Q_j & \text{otherwise}
\end{cases}
\end{aligned}
$$

ここで $J' = \{j' \mid N_{j'} = \mathsf{true} \wedge n_{j'} = \underline{0}\}$，$J'' = \{j' \mid n_{j'} = \min(\{n_j \mid N_j = \mathsf{true}\})\}$ である．また，t はタイムアウトを表す名前であり，P および $[\![P]\!]_t$ には出現しないとする． □

時間経過動作の抽象化はプロセス式のプレフィックスのみに対して適用され，サブプロセス式に対しては再帰的に適用されない．なぜなら，時間経過動作の添字が入力プレフィックスによって束縛される場合，どのように抽象化するか決定できない場合が存在するためである．再帰的に時間経過動作の抽象化を適用するためには，例えば，プロセス式 $x(n).(t[n].Q + t[\underline{5}].R)$ に対し $t[n].Q + t[\underline{5}].R$ も抽象化する必要がある．しかし，入力動作 $x(m)$ によって n に代入される名前 m に応じて抽象化は以下の 3 通りの可能性が存在し，一意に決めることができない．

$$\begin{array}{ll} t.R & \text{if } m < \underline{5} \\ t.Q + t.R & \text{if } m = \underline{5} \\ t.Q & \text{if } \underline{5} < m \end{array}$$

よって，$t[n].Q + t[\underline{5}].R$ を精密に抽象化するためには入力動作が実行されて n に代入される名前が確定するまで待つ必要がある．そのため時間経過動作の抽象化は再帰的にサブプロセスに適用しない．入出力動作，τ 動作，時間経過動作を抽象化した t 動作が実行されるごとに抽象化を行う．

補題 4.3 プロセス式 P を $\mathcal{EP}$ に含まれ，かつ単一のプロセスを表すプロセス式とする．

1. $P \stackrel{\alpha}{\rightarrowtail} P'$ ならば $[\![P]\!]_t \stackrel{\alpha}{\rightarrowtail} P'$（ただし $\alpha \neq \bullet$）
2. $[\![P]\!]_t \stackrel{\alpha}{\rightarrowtail} P'$ ならば $P \stackrel{\alpha}{\rightarrowtail} P'$（ただし $\alpha \neq t$）
3. $P \rightarrow^* \stackrel{\bullet}{\rightarrowtail} P'$ ならば $[\![P]\!]_t \stackrel{t}{\rightarrowtail} P'$
4. $[\![P]\!]_t \stackrel{t}{\rightarrowtail} P'$ ならば $P \rightarrow^* \stackrel{\bullet}{\rightarrowtail} P'$
5. $[\![P]\!]_t \stackrel{\alpha}{\rightarrowtail} P' \iff [\![P]\!]_t \stackrel{\alpha}{\rightarrow} P'$

証明：

1. - $P = \mathbf{0}$ の場合，$P \stackrel{\alpha}{\not\rightarrowtail}$.
   - $P = \sum_i M_i \alpha_i @ d_i.P_i + \sum_j N_j t[n_j].Q_j$ の場合，$P \stackrel{\alpha}{\rightarrowtail} P'$ とすると，$M_i = \mathsf{true}$ かつ $\alpha = \alpha_i$ かつ $P' = P_i\{\underline{0}/d_i\}$ なる i が存在し，$N_j = \mathsf{true}$ かつ $n_j = \underline{0}$ なる j は存在しない．この時 $[\![P]\!]_t$ は $\sum_i M_i \alpha_i.P_i\{\underline{0}/d_i\}$ または $\sum_i M_i \alpha_i.P_i\{\underline{0}/d_i\} + \sum_j N_j t.Q_j$ のいずれかであり，いずれの場合も $[\![P]\!]_t \stackrel{\alpha}{\rightarrowtail} P_i\{\underline{0}/d_i\} = P'$ とできる．

- $P = \boldsymbol{\nu} x\, P'$ の場合，$P' \xrightharpoonup{\alpha} P''$，$P \xrightharpoonup{\alpha} \boldsymbol{\nu} x\, P''$ とする．$[\![P]\!]_t = \boldsymbol{\nu} x\, [\![P']\!]_t$ であり，帰納法の仮定より $[\![P']\!]_t \xrightharpoonup{\alpha} P''$．よって $[\![P]\!]_t \xrightharpoonup{\alpha} \boldsymbol{\nu} x\, P''$ とできる．

2. - $[\![P]\!]_t = \mathbf{0}$ の場合，$[\![P]\!]_t \not\xrightharpoonup{\alpha}$.
   - $[\![P]\!]_t = \sum_{j \in J'} N_j t.Q_j$ の場合，$\alpha \neq t$ に反する．
   - $[\![P]\!]_t = \sum_i M_i \alpha_i.P_i\{\underline{0}/d_i\}$ の場合，$[\![P]\!]_t \xrightharpoonup{\alpha} P'$ とすると，$M_i =$ true かつ $\alpha = \alpha_i$ かつ $P' = P_i\{\underline{0}/d_i\}$ なる i が存在する．$P = \sum_i M_i \alpha_i @ d_i.P_i + \sum_j N_j t[n_j].Q_j$ であり，$M_i =$ true かつ $\alpha = \alpha_i = \tau$，および $N_j =$ true かつ $n_j = \underline{0}$ なる j は存在しないため，$P \xrightharpoonup{\alpha} P_i\{\underline{0}/d_i\}$ とできる．
   - $[\![P]\!]_t = \boldsymbol{\nu} x\, P'$ の場合，$P' = [\![P_0]\!]_t$ なる P_0 が存在して $P = \boldsymbol{\nu} x\, P_0$ である．$P' \xrightharpoonup{\alpha} P''$，$[\![P]\!]_t \xrightharpoonup{\alpha} \boldsymbol{\nu} x\, P''$ とする．この時，$[\![P_0]\!]_t \xrightharpoonup{\alpha} P''$ であり帰納法の仮定より $P_0 \xrightharpoonup{\alpha} P''$ である．よって $P \xrightharpoonup{\alpha} \boldsymbol{\nu} x\, P''$ とできる．

3. 1. と同様.

4. 2. と同様.

5. $[\![P]\!]_t$ は時間経過動作を含まないためタイムアウト遷移は存在しない． □

並行合成の展開と逐次プロセスに対する時間経過動作の抽象化を組み合わせて，任意のプロセスに対する時間経過動作の抽象化を定義する．

定義 4.13 プロセス式 $P \in \mathcal{EP}$ に対し時間経過動作の抽象化 $[\![P]\!]_a$ を

$$[\![P]\!]_a \quad \stackrel{def}{=} \quad [\![[\![P]\!]_e]\!]_t$$

と定義する． □

補題 4.4 プロセス式 $P \in \mathcal{EP}$ に対して以下が成り立つ．

1. $P \xrightharpoonup{\alpha} P'$ ならば $[\![P]\!]_a \xrightharpoonup{\alpha} \sim^f_T P'$（ただし $\alpha \neq \bullet$）

2. $[\![P]\!]_a \xrightharpoonup{\alpha} P'$ ならば $P \xrightharpoonup{\alpha} \sim^f_T P'$（ただし $\alpha \neq t$）

3. $P \rightharpoondown^* \xrightharpoonup{\bullet} P'$ ならば $[\![P]\!]_a \xrightharpoonup{t} \sim^f_T P'$

4. $[\![P]\!]_a \xrightharpoonup{t} P'$ ならば $P \rightharpoondown^* \xrightharpoonup{\bullet} \sim^f_T P'$

5. $[\![P]\!]_a \xrightharpoonup{\alpha} P' \iff [\![P]\!]_a \xrightarrow{\alpha} P'$

証明：

1. 補題 4.2 より $P \sim_{\mathcal{T}}^{f} [\![P]\!]_e$ ゆえ P' に対して $[\![P]\!]_e \stackrel{\alpha}{\rightarrow} P_e$ かつ $P' \sim_{\mathcal{T}}^{f} P_e$ なる P_e が存在する．ここで，補題 4.3(1) より $[\![[\![P]\!]_e]\!]_t \stackrel{\alpha}{\rightarrow} P_e$ であるので，$[\![P]\!]_a \stackrel{\alpha}{\rightarrow} \sim_{\mathcal{T}}^{f} P'$.

2. 補題 4.3(2) より $[\![P]\!]_e \stackrel{\alpha}{\rightarrow} P'$ である．補題 4.2 より $P \sim_{\mathcal{T}}^{f} [\![P]\!]_e$ ゆえ P' に対して $P \stackrel{\alpha}{\rightarrow} P''$ かつ $P' \sim_{\mathcal{T}}^{f} P''$ なる P'' が存在する．よって $P \stackrel{\alpha}{\rightarrow} \sim_{\mathcal{T}}^{f} P'$.

3. 1. と同様.

4. 2. と同様.

5. $[\![\cdot]\!]_a = [\![[\![\cdot]\!]_e]\!]_t$ であり $[\![P]\!]_t$ は時間経過動作を含まないためタイムアウト遷移は存在しない. □

4.6　性質

時間経過動作を抽象する前後のプロセス間の関係について述べる．初めに時間経過動作を抽象化するプロセスの間の関係を定義する.

定義 4.14 以下を満たす関係 $\mathcal{R}$ を時間抽象双模倣関係と呼ぶ.
$(P, Q) \in \mathcal{R}$ の時,

- $[\![P]\!]_a \xrightarrow{\alpha} P' \;\Rightarrow\; \exists Q'.\, [\![Q]\!]_a \xrightarrow{\alpha} Q' \wedge (P', Q') \in \mathcal{R}$
- $[\![Q]\!]_a \xrightarrow{\alpha} Q' \;\Rightarrow\; \exists P'.\, [\![P]\!]_a \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathcal{R}$ □

$(P, Q) \in \mathcal{R}$ なる時間抽象双模倣関係 $\mathcal{R}$ が存在する時 $P \sim_{\mathcal{T}}^{ta} Q$ と書く.

定義 4.15 $\sim_{\mathcal{T}}^{ta} \;=\; \bigcup\{\mathcal{R} \mid \mathcal{R} \text{ は時間抽象双模倣関係 }\}$ □

命題 4.3 $\sim_{\mathcal{T}}^{ta}$ は最大の時間抽象双模倣関係である. □

時間待ちの長さが動的に決まる場合があるため，また複製演算子が存在するため，初めにプロセス式全体について時間経過動作を抽象することは不可能であり，入出力動作，内部動作，タイムアウト動作が実行されるたびに抽象化を行う.

$P \sim_{\mathcal{T}}^{ta} Q$ ならば，P と Q は時間経過動作を抽象化すると振舞いが等価であるとみなすことができる．例えば,

$$P \;=\; t[\underline{1}].x.\mathbf{0} \mid t[\underline{1}].y.\mathbf{0} \tag{4.1}$$

$$Q \;=\; t[\underline{1}].t[\underline{0}].(x.y.\mathbf{0} + y.x.\mathbf{0}) \tag{4.2}$$

とすると $P \sim_{\mathcal{T}}^{ta} Q$ である．なぜなら，

$$\begin{aligned}
[\![P]\!]_a &= t.(x.\mathbf{0} \,|\, t[\underline{0}].y.\mathbf{0}) + t.(t[\underline{0}].x.\mathbf{0} \,|\, y.\mathbf{0}) \\
[\![x.\mathbf{0} \,|\, t[\underline{0}].y.\mathbf{0}]\!]_a &= t.(x.\mathbf{0} \,|\, y.\mathbf{0}) \\
[\![t[\underline{0}].x.\mathbf{0} \,|\, y.\mathbf{0}]\!]_a &= t.(x.\mathbf{0} \,|\, y.\mathbf{0}) \\
[\![Q]\!]_a &= t.t[\underline{0}].(x.y.\mathbf{0} + y.x.\mathbf{0}) \\
[\![t[\underline{0}].(x.y.\mathbf{0} + y.x.\mathbf{0})]\!]_a &= t.(x.y.\mathbf{0} + y.x.\mathbf{0})
\end{aligned}$$

であり，関係 $\mathcal{R}$ を

$$\begin{aligned}
\mathcal{R} \;=\; \{&(t[\underline{1}].x.\mathbf{0} \,|\, t[\underline{1}].y.\mathbf{0},\ t[\underline{1}].t[\underline{0}].(x.y.\mathbf{0} + y.x.\mathbf{0})), \\
&(x.\mathbf{0} \,|\, t[\underline{0}].y.\mathbf{0},\ t[\underline{0}].(x.y.\mathbf{0} + y.x.\mathbf{0})), \\
&(t[\underline{0}].x.\mathbf{0} \,|\, y.\mathbf{0},\ t[\underline{0}].(x.y.\mathbf{0} + y.x.\mathbf{0})), \\
&(x.\mathbf{0} \,|\, y.\mathbf{0},\ x.y.\mathbf{0} + y.x.\mathbf{0}), \\
&(\mathbf{0} \,|\, y.\mathbf{0},\ y.\mathbf{0}), \\
&(x.\mathbf{0} \,|\, \mathbf{0},\ x.\mathbf{0}), \\
&(\mathbf{0} \,|\, \mathbf{0},\ \mathbf{0})\}
\end{aligned}$$

とすると，$\mathcal{R} \subseteq \sim_{\mathcal{T}}^{ta}$ となるからである．

補題 4.5 $\sim_{\mathcal{T}}^{f} \sim_{\mathcal{T}}^{ta} \sim_{\mathcal{T}}^{f} \subseteq \sim_{\mathcal{T}}^{ta}$ □

証明： $P \sim_{\mathcal{T}}^{f} Q \sim_{\mathcal{T}}^{ta} R \sim_{\mathcal{T}}^{f} S$ とする．

- $[\![P]\!]_a \overset{\alpha}{\rightarrow} P'$（ただし $\alpha \neq t$）の場合，補題 4.4(2) より $P \overset{\alpha}{\rightarrow} P'' \sim_{\mathcal{T}}^{f} P'$ なる P'' が存在する．$P \sim_{\mathcal{T}}^{f} Q$ ゆえ P'' に対し $Q \overset{\alpha}{\rightarrow} Q'' \sim_{\mathcal{T}}^{f} P''$ なる Q'' が存在する．明らかに $\alpha \neq \bullet$ であり補題 4.4(1) より $[\![Q]\!]_a \overset{\alpha}{\rightarrow} Q' \sim_{\mathcal{T}}^{f} Q''$ なる Q' が存在する．$Q \sim_{\mathcal{T}}^{ta} R$ ゆえ Q' に対し $[\![R]\!]_a \overset{\alpha}{\rightarrow} R' \sim_{\mathcal{T}}^{ta} Q'$ なる R' が存在する．補題 4.4(2) より $R \overset{\alpha}{\rightarrow} R'' \sim_{\mathcal{T}}^{f} R'$ なる R'' が存在する．$R \sim_{\mathcal{T}}^{f} S$ ゆえ R'' に対し $S \overset{\alpha}{\rightarrow} S'' \sim_{\mathcal{T}}^{f} R''$ なる S'' が存在する．補題 4.4(1) より $[\![S]\!]_a \overset{\alpha}{\rightarrow} S' \sim_{\mathcal{T}}^{f} S''$ なる S' が存在する．以上より，P' に対し $[\![S]\!]_a \overset{\alpha}{\rightarrow} S'$ かつ $P' \sim_{\mathcal{T}}^{f} P'' \sim_{\mathcal{T}}^{f} Q'' \sim_{\mathcal{T}}^{f} Q' \sim_{\mathcal{T}}^{ta} R' \sim_{\mathcal{T}}^{f} R'' \sim_{\mathcal{T}}^{f} S'' \sim_{\mathcal{T}} S'$ なる S' が存在し，$\sim_{\mathcal{T}}^{f}$ の推移性から $P' \sim_{\mathcal{T}}^{f} \sim_{\mathcal{T}}^{ta} \sim_{\mathcal{T}}^{f} S'$ である．
- 他の場合も同様である． □

次に，入出力動作，内部動作，タイムアウト動作に着目し，時間経過動作には着目しない双模倣関係を定義する．

定義 4.16 以下を満たす対称な関係 $\mathcal{R}$ をタイムアウト双模倣関係と呼ぶ．
$(P,Q) \in \mathcal{R}$ の時，

- $P \stackrel{\alpha}{\rightharpoonup} P' \;\Rightarrow\; \exists Q'.\, Q \stackrel{\alpha}{\rightharpoonup} Q' \wedge (P',Q') \in \mathcal{R} \quad (\text{ただし } \alpha \neq \bullet)$
- $P \rightharpoonup^* \stackrel{\bullet}{\rightharpoonup} P' \;\Rightarrow\; \exists Q'.\, Q \rightharpoonup^* \stackrel{\bullet}{\rightharpoonup} Q' \wedge (P',Q') \in \mathcal{R}$ □

$(P,Q) \in \mathcal{R}$ なるタイムアウト双模倣関係 $\mathcal{R}$ が存在する時 $P \sim_{\mathcal{T}}^{to} Q$ と書く．

定義 4.17 $\sim_{\mathcal{T}}^{to} \;=\; \bigcup\{\mathcal{R} \mid \mathcal{R} \text{ はタイムアウト双模倣関係}\}$ □

命題 4.4 $\sim_{\mathcal{T}}^{to}$ は最大のタイムアウト双模倣関係である． □

例えば，式 (4.1) の P と式 (4.2) の Q に対して $P \sim_{\mathcal{T}}^{to} Q$ である．このことは以下に示す関係 $\mathcal{R}$ が $\mathcal{R} \subseteq \sim_{\mathcal{T}}^{to}$ であることからわかる．

$$\begin{aligned}\mathcal{R} \;=\; \{&(t[\underline{1}].x.\mathbf{0} \mid t[\underline{1}].y.\mathbf{0},\; t[\underline{1}].t[\underline{0}].(x.y.\mathbf{0} + y.x.\mathbf{0})),\\ &(x.\mathbf{0} \mid t[\underline{0}].y.\mathbf{0},\; t[\underline{0}].(x.y.\mathbf{0} + y.x.\mathbf{0})),\\ &(t[\underline{0}].x.\mathbf{0} \mid y.\mathbf{0},\; t[\underline{0}].(x.y.\mathbf{0} + y.x.\mathbf{0})),\\ &(x.\mathbf{0} \mid y.\mathbf{0},\; x.y.\mathbf{0} + y.x.\mathbf{0}),\\ &(\mathbf{0} \mid y.\mathbf{0},\; y.\mathbf{0}),\\ &(x.\mathbf{0} \mid \mathbf{0},\; x.\mathbf{0}),\\ &(\mathbf{0} \mid \mathbf{0},\; \mathbf{0})\}\end{aligned}$$

補題 4.6 $\sim_{\mathcal{T}}^{f} \sim_{\mathcal{T}}^{to} \sim_{\mathcal{T}}^{f} \subseteq \sim_{\mathcal{T}}^{to}$

証明： $P \sim_{\mathcal{T}}^{f} Q \sim_{\mathcal{T}}^{to} R \sim_{\mathcal{T}}^{f} S$ とする．

- $P \rightharpoonup^* \stackrel{\bullet}{\rightharpoonup} P'$ の場合，$P \rightharpoonup^n \stackrel{\bullet}{\rightharpoonup} P'$ とすると $P \sim_{\mathcal{T}}^{f} Q$ ゆえ $Q \rightharpoonup^n \stackrel{\bullet}{\rightharpoonup} Q' \sim_{\mathcal{T}}^{f} P'$ なる Q' が存在する．$Q \sim_{\mathcal{T}}^{to} R$ なので Q' に対し $R \rightharpoonup^m \stackrel{\bullet}{\rightharpoonup} R' \sim_{\mathcal{T}}^{to} Q'$ なる m と R' が存在する．$R \sim_{\mathcal{T}}^{f} S$ なので R' に対し $S \rightharpoonup^m \stackrel{\bullet}{\rightharpoonup} S' \sim_{\mathcal{T}}^{f} R'$ なる S' が存在する．以上より，P' に対し $S \rightharpoonup^* \stackrel{\bullet}{\rightharpoonup} S'$ かつ $P' \sim_{\mathcal{T}}^{f} \sim_{\mathcal{T}}^{to} \sim_{\mathcal{T}}^{f} S'$ なる S' が存在する．
- 他の場合も同様である． □

この時，時間抽象双模倣なプロセスはタイムアウト双模倣であり，逆も成り立つ．時間経過動作に着目する必要がない性質についてプロセスの振舞いを解析する場合，構文的に時間経過動作を抽象してから 1 ステップずつプロセスの動作を調べればよい．

定理 4.4 $\sim_{\mathcal{I}}^{ta} = \sim_{\mathcal{I}}^{to}$

証明： 初めに $\sim_{\mathcal{I}}^{ta} \subseteq \sim_{\mathcal{I}}^{to}$ を示す．$P \sim_{\mathcal{I}}^{ta} Q$ とする．

- $P \stackrel{\alpha}{\rightarrow} P'$（ただし $\alpha \neq \bullet$）の時，補題 4.4(1) よりある P'' が存在して $[\![P]\!]_a \stackrel{\alpha}{\Rightarrow} P'' \sim_{\mathcal{I}}^{f} P'$ である．$P \sim_{\mathcal{I}}^{ta} Q$ ゆえ P'' に対し $[\![Q]\!]_a \stackrel{\alpha}{\Rightarrow} Q''$ かつ $P'' \sim_{\mathcal{I}}^{ta} Q''$ なる Q'' が存在する．補題 4.4(2) よりある Q' が存在して $Q \stackrel{\alpha}{\Rightarrow} Q' \sim_{\mathcal{I}}^{f} Q''$ である．この時 $P' \sim_{\mathcal{I}}^{f} P'' \sim_{\mathcal{I}}^{ta} Q'' \sim_{\mathcal{I}}^{f} Q'$ であり，補題 4.5 より $P' \sim_{\mathcal{I}}^{ta} Q'$ である．

- $P \rightarrow^* \stackrel{\bullet}{\Rightarrow} P'$ の時，補題 4.4(3) よりある P'' が存在して $[\![P]\!]_a \stackrel{t}{\Rightarrow} P'' \sim_{\mathcal{I}}^{f} P'$ である．$P \sim_{\mathcal{I}}^{ta} Q$ ゆえ P'' に対して $[\![Q]\!]_a \stackrel{t}{\Rightarrow} Q''$ かつ $P'' \sim_{\mathcal{I}}^{ta} Q''$ なる Q'' が存在する．補題 4.4(4) よりある Q' が存在して $Q \rightarrow^* \stackrel{\bullet}{\Rightarrow} Q' \sim_{\mathcal{I}}^{f} Q''$ である．この時 $P' \sim_{\mathcal{I}}^{f} P'' \sim_{\mathcal{I}}^{ta} Q'' \sim_{\mathcal{I}}^{f} Q'$ であり，補題 4.5 より $P' \sim_{\mathcal{I}}^{ta} Q'$ である．

次に $\sim_{\mathcal{I}}^{to} \subseteq \sim_{\mathcal{I}}^{ta}$ を示す．$P \sim_{\mathcal{I}}^{to} Q$ とする．

- $[\![P]\!]_a \stackrel{\alpha}{\Rightarrow} P'$（ただし $\alpha \neq t$）の時，補題 4.4(2) よりある P'' が存在して $P \stackrel{\alpha}{\Rightarrow} P'' \sim_{\mathcal{I}}^{f} P'$ である．ここで $\alpha \neq t$ ゆえ $\alpha \neq \bullet$ でもある．$P \sim_{\mathcal{I}}^{to} Q$ ゆえ P'' に対して $Q \stackrel{\alpha}{\Rightarrow} Q''$ かつ $P'' \sim_{\mathcal{I}}^{to} Q''$ なる Q'' が存在する．補題 4.4(1) よりある Q' が存在して $[\![Q]\!] \stackrel{\alpha}{\Rightarrow} Q' \sim_{\mathcal{I}}^{f} Q''$ である．この時 $P' \sim_{\mathcal{I}}^{f} P'' \sim_{\mathcal{I}}^{to} Q'' \sim_{\mathcal{I}}^{f} Q'$ であり，補題 4.6 より $P' \sim_{\mathcal{I}}^{to} Q'$ である．

- $[\![P]\!]_a \stackrel{t}{\Rightarrow} P'$ の時，補題 4.4(4) よりある P'' が存在して $P \rightarrow^* \stackrel{\bullet}{\Rightarrow} P'' \sim_{\mathcal{I}}^{f} P'$ である．$P \sim_{\mathcal{I}}^{to} Q$ ゆえ P'' に対して $Q \rightarrow^* \stackrel{\bullet}{\Rightarrow} Q''$ かつ $P'' \sim_{\mathcal{I}}^{to} Q''$ なる Q'' が存在する．補題 4.4(3) よりある Q' が存在して $[\![Q]\!]_a \stackrel{t}{\Rightarrow} Q' \sim_{\mathcal{I}}^{f} Q''$ である．この時 $P' \sim_{\mathcal{I}}^{f} P'' \sim_{\mathcal{I}}^{to} Q'' \sim_{\mathcal{I}}^{f} Q'$ であり，補題 4.6 より $P' \sim_{\mathcal{I}}^{to} Q'$ である． □

タイムアウト双模倣関係は入出力動作，内部動作および最も直近のタイムアウト動作のみに着目した時に双模倣関係が成り立つことを表している．定理 4.4 は，時間経過動作を抽象化した時に双模倣であることを表す時間抽象双模倣関係にある 2 つのプロセスは，タイムアウト双模倣関係でもあることを示している．すなわち，直近のタイムアウトを除く時間に関する動作を無視してもよい場合は，時間経過動作を抽象化して調べればよい．

例えば，プロセス

$$\begin{aligned} S &= (\boldsymbol{\nu} send, ack) \\ &\quad (\overline{send}.(ack.P + t[\underline{5}].\overline{send}.P) \mid send.\overline{ack}.Q + send.t[\underline{10}].\overline{ack}.Q) \\ T &= (\boldsymbol{\nu} send, ack)\,(\tau.\tau.(P \mid Q) + \tau.t[\underline{7}].\mathbf{0}) \end{aligned}$$

について，$S \sim_{\mathcal{T}}^{to} T$ を示すことを考える．直接的に示すのであれば，S から τ で遷移したプロセス

$$\begin{cases} (\boldsymbol{\nu} send, ack)\,(ack.P + t[\underline{5}].\overline{send}.P \mid \overline{ack}.Q) \\ (\boldsymbol{\nu} send, ack)\,(ack.P + t[\underline{5}].\overline{send}.P \mid t[\underline{10}].\overline{ack}.Q) \end{cases}$$

と，T から τ で遷移したプロセス

$$\begin{cases} (\boldsymbol{\nu} send, ack)\,(\tau.(P \mid Q)) \\ (\boldsymbol{\nu} send, ack)\,(t[\underline{7}].\mathbf{0}) \end{cases}$$

が対応付けられるか確認する．この時，いずれも後者のプロセスについては 1 単位時間ずつ時間経過を発生させて，いつどのサブプロセスでタイムアウトが発生するか確認しなければならない．

一方，$S \sim_{\mathcal{T}}^{ta} T$ はより簡潔に示すことができる．

$$\begin{aligned} [\![S]\!]_a &= \sim_{\mathcal{T}} \quad (\boldsymbol{\nu} send, ack) \\ &\qquad\quad (\tau.(ack.P + t[\underline{5}].\overline{send}.P \mid \overline{ack}.Q) \\ &\qquad\quad +\tau.(ack.P + t[\underline{5}].\overline{send}.P \mid t[\underline{10}].\overline{ack}.Q)) \\ [\![T]\!]_a &= T \end{aligned}$$

ゆえ，最初の τ による遷移先は $S \sim_{\mathcal{T}}^{to} T$ を直接示す場合と同じである．その次は，

$$\begin{aligned} &[\![(\boldsymbol{\nu} send, ack)\,(ack.P + t[\underline{5}].\overline{send}.P \mid \overline{ack}.Q)]\!]_a \\ &\quad = \sim_{\mathcal{T}} \quad (\boldsymbol{\nu} send, ack)\,(\tau.(P \mid Q)) \\ &[\![(\boldsymbol{\nu} send, ack)\,(ack.P + t[\underline{5}].\overline{send}.P \mid t[\underline{10}].\overline{ack}.Q)]\!]_a \\ &\quad = \sim_{\mathcal{T}} \quad (\boldsymbol{\nu} send, ack)\,(t.(\overline{send}.P \mid t[\underline{5}].\overline{ack}.Q)) \end{aligned}$$

および

$$[\![(\boldsymbol{\nu} send, ack)\,(\tau.(P \mid Q))]\!]_a = (\boldsymbol{\nu} send, ack)\,(\tau.(P \mid Q))$$

$$[\![(\boldsymbol{\nu} send, ack)\,(t[\underline{7}].\mathbf{0})]\!]_a \;=\; (\boldsymbol{\nu} send, ack)\,(t.\mathbf{0})$$

となり，次に実行可能な動作からプロセスの対応付けをすぐに行うことができる．

この例では非常に簡単なプロセスを用いたため，時間経過動作を抽象化しなくとも直感的に双模倣関係が存在するか確認できるかもしれない．しかし，プロセスの規模が大きくなると双模倣性の判定は難しくなり，時間経過動作の抽象化によって状態数を減らすことは有用である．また，時間経過動作を単に捨象したり τ 動作などに置換するだけでは最も早く発生するタイムアウトが選択されることが表現されず，提案手法により精度の良い抽象化が可能である．

4.7 関連研究

Larsen らは CCS の時間拡張である Timed CCS[Yi91a, Yi91b] において，時間動作を完全に抽象化した双模倣関係である Time Abstracted Bisimulation を提案している [LY93, LY97]．時間に関する性質をほとんど表現しない関係であるが，プロセス P と Q について任意のプロセスの並行合成に対して time abstracted bisimilar であれば P と Q は時間動作に関して双模倣であることが示されている．オートマトンを時間制約を表現できるよう拡張した時間オートマトンに対しては，時間制約に基づいて時間変数の領域を分割することで時間を抽象化し状態空間を削減する手法が提案されている [AD90, AD94]．

我々の体系では，タイムアウトの発生は特別なラベルによる遷移 $\overset{\bullet}{\rightarrow}$ によって表現され，時間制約は時間経過動作 $t[n]$ によって表現される．タイムアウトを表す遷移があるため，時間領域を単位時間ではなくタイムアウトの発生を基準にして分割することができる．また，タイムアウトの発生を捉えられるため，Larsen らの Time Abstracted Bisimulation よりも詳細に振舞いを解析することができる．

4.8 おわりに

本章では，時間付き π 計算のプロセスの時間動作をタイムアウト動作を残しながら抽象化する手法を提案した．時間付き π 計算は時間に関して細かい意味論を提供しているため，時間動作を詳細に調べることが可能である一方，状態を細かく分割することになるため状態爆発の問題が深刻になる．モデルの状態数を削減するための抽象化手法が必要である．

提案した時間動作の抽象化手法は

1. 展開定理に基づく並行プロセスから逐次プロセスへの変換

2. 時間経過動作の抽象

からなる．並行プロセスから逐次プロセスに変換することで時間経過動作の抽象が容易になる．展開定理により並行プロセスから逐次プロセスへの変換は振舞いを保存することが保証されている．時間経過動作の抽象では，タイムアウトの発生から次のタイムアウトの発生までを一つの状態にまとめる．タイムアウトから次のタイムアウトまでの間に実行可能な入出力動作と内部動作は，可能動作不変性により不変であるため一つの状態にまとめることができる．

時間動作の抽象化が行われたプロセスが双模倣であるならば，元になったプロセスは入出力動作，内部動作，タイムアウト動作のみに着目すれば双模倣であることを示した．つまり，時間動作に着目する必要のない振舞いを解析する時は，時間動作を抽象化すればよい．複製演算子の存在と，時間付き π 計算の特徴である時間待ちの長さの動的決定から，抽象化は入出力動作，内部動作，タイムアウト動作ごとに行う必要がある．抽象化は構文的に行うことが可能であり，抽象化を行わずに時間経過動作を直接扱うよりも振舞い解析が容易になると考えられる．

時間動作が抽象されたプロセス式は従来の π 計算の枠組みの中で表現可能である．また，振舞い等価性を判定する際に時間経過動作を無視することができる．抽象化の手続きを融合することで従来の π 計算における双模倣関係に基づいて振舞い解析を行うことが可能になる．従来の π 計算に対しては，プロセスの双模倣性の判定に関する研究が多く行われている [San93, Lin94, MP95, San98, Lin00, PS01, FJ01]．また，Mobility Workbench[VM94, Vic94][1] ，PiET[2]，ABC[3]といった等価性判定のためのツールも作成されている．従来の π 計算の枠組みの中で振舞いを解析できればこれらの成果を利用することができ有益である．

[1] http://www.it.uu.se/research/group/mobility/mwb/
[2] http://piet.sourceforge.net/
[3] http://lampwww.epfl.ch/~sbriais/

第5章　リアルタイムオブジェクト指向言語の形式的記述

5.1　はじめに

実時間システムが正常に動作することを振舞い等価性判定やモデル検査技法を用いて確認するためには，システムの時間を含む動作を定式化する必要がある．実時間システムの多くは組込みシステムであるが，近年，その開発にはオブジェクト指向開発技法が用いられている．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し，オブジェクト指向言語を用いて実装を行う．システムの動作はUML[Dou99, OMG05a, OMG05b]などを用いて描かれた設計図，通信プロセスモデルによる振舞い記述，実装時に作成されたソフトウェアなどによって表現される．

システムの実際の動作を最も詳細かつ直接的に表現しているのはシステムを構成するソフトウェアプログラムである．ソフトウェアプログラムの形式的記述を得ることでプログラムの実行に沿ってシステムの動作を解析したり検証することができる．

そこで本章では，実時間システムの動作の定式化としてソフトウェアプログラムから第2章で定義した時間付きπ計算による記述への変換規則を与える．具体的には，時間に関する動作を直接的に記述するための構文を持つリアルタイムオブジェクト指向言語の意味論を時間付きπ計算を用いて定義する．時間付きπ計算は，π計算に由来する通信や並行性に関する高い表現能力と時間拡張に由来する実時間性を表現する能力を持ち，振舞いを検証する技法が存在する．そのためリアルタイムオブジェクト指向言語の意味論を記述するメタ言語に適している．

本章の構成は以下の通りである．5.2節では対象とするリアルタイムオブジェクト指向言語OOL_{RT}について述べる．5.3節ではOOL_{RT}の意味論を時間付きπ計算によって定義する．OOL_{RT}の各構文要素に対して時間付きπ計算による記述への変換規則を与える．5.4節ではOOL_{RT}のプログラムとその形式的記述の例を示

す．最後に関連研究を挙げてまとめを述べる．

5.2 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[SW01] からデータ型の取り扱いを省き，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} ではクラスの外からメンバ変数へ直接アクセスすることは不可能でありカプセル化を実現している．

OOL_{RT} の構文を図 5.1 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, S_i は文，X, X_i は変数名，$Numeral$ は整数リテラルを表す．プログラムは必要なクラスの定義 $Cdec_i$ と実行を開始する時に評価される式 E からなる．クラスはメンバ変数の宣言 $Vdecs$ とメソッドの定義 $Mdecs$ からなる．メソッドは任意の数の引数を持つことができる．

OOL_{RT} の文には代入，連接，分岐，繰り返しの他に，単純な時間待ちと分岐を伴う時間待ちが存在する．単純な時間待ちは

$$\mathsf{wait}\ E$$

のように記述する．式 E を評価して得られた値が表す長さだけ時間待ちを行う．分岐を伴う時間待ちは

$$\mathsf{within}\ E\ \mathsf{do}\ S_1\ \mathsf{timeout}\ S_2\ \mathsf{done}$$

のように記述する．初めに文 S_1 の実行を開始すると同時に時間待ちを開始する．式 E を評価して得られた値が表す時間長が経過する前に S_1 の実行が完了すれば次の文を実行する．もし S_1 の実行が時間内に完了しなければ文 S_2 を実行する．

OOL_{RT} の式には変数，真偽値リテラル，整数リテラル，インスタンス生成，メソッド呼び出しがある．カプセル化のためにクラスのメンバ変数へのアクセスする機能は提供されない．

5.3 形式的記述

OOL_{RT} の構文要素それぞれに対し時間付き π 計算による形式的記述を与える．構文要素 A に対応する形式的記述を $[\![A]\!]$ と表す．ここで与える形式的記述全体によって OOL_{RT} の意味論が定義される．

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

図 5.1: OOL_{RT} の構文

$$
\begin{aligned}
&[\![Pdec]\!] \\
&\quad \equiv \ (\boldsymbol{\nu}\widetilde{k})\,([\![Cdec_1]\!](\widetilde{k}) \mid \cdots \mid [\![Cdec_n]\!](\widetilde{k}) \mid \mathrm{Boolean}(k_{Bool}) \mid (\boldsymbol{\nu}l)[\![E]\!](l, \mathsf{nil}, \widetilde{k}))
\end{aligned}
$$

図 5.2: プログラムに対する形式的記述

時間付き π 計算では一度の通信で送受信できる名前は高々一つであるが，形式的記述を簡潔にするために複数個の名前の送受信を表現する記法を導入する．

定義 5.1 任意の n 個の名前の受信に対応した入力プレフィックス，および送信に対応した出力プレフィックスを以下のように定義する．

$$
\begin{aligned}
x(y_1, \ldots, y_n).P &\stackrel{def}{=} x(w).w(y_1).\cdots.w(y_n).P \\
\overline{x}\langle z_1, \ldots, z_n\rangle.P &\stackrel{def}{=} \boldsymbol{\nu}w\,\overline{x}\langle w\rangle.\overline{w}\langle z_1\rangle.\cdots.\overline{w}\langle z_n\rangle.P
\end{aligned}
$$

w は新しい名前である． □

以下では，名前のリストを $\widetilde{x}$ のように表し，i 番目の名前を x_i と書く．$\widetilde{x}\cdot z$ はリスト $\widetilde{x}$ に名前 z を追加することを表し，$\widetilde{x}\cdot\widetilde{y}$ は二つのリスト $\widetilde{x}$ と $\widetilde{y}$ の連結を表す．

5.3.1 プログラム

プログラム $Pdec$ に対する時間付き π 計算による形式的記述を図 5.2 に示す．各クラスの定義 $Cdec_i$，プログラムに標準で組み込まれるブール値を表すクラス Boolean，プログラムの実行開始時に評価される式 E それぞれの形式的記述が並行合成されたプロセスとして表される．$\widetilde{k}$ はプログラム中で定義されるクラスを表す名前のリストを示す．名前 k_{Bool} は Boolean クラスを表し $k_{Bool} \in \widetilde{k}$ である．

5.3.2 クラス定義

クラス定義 $Cdec$ に対する時間付き π 計算による形式的記述を図 5.3 に示す．名前 k がここで定義されているクラスを表し，名前 a がこのクラスのインスタンスの一つを表す．$k \in \widetilde{k}$ である．このクラスのインスタンスが生成される時は以下のように動作する．

$$[\![Cdec]\!](\widetilde{k})$$

$$
\begin{aligned}
&[\![Cdec]\!](\widetilde{k}) \\
&\quad \equiv \ !\,(\boldsymbol{\nu} a)\,(k(l).\bar{l}\langle a\rangle.(\boldsymbol{\nu}\widetilde{x},\widetilde{m})\,([\![Vdecs]\!](\widetilde{x},\widetilde{c}\cdot a) \mid [\![Mdecs]\!](\widetilde{m},a,\widetilde{x},\widetilde{k})))
\end{aligned}
$$

図 5.3: クラス定義に対する形式的記述

$$
\begin{aligned}
&[\![Vdecs]\!](\widetilde{x},\widetilde{c}) \\
&\quad \equiv \ [\![\mathsf{var}\ X_1]\!](x_1,c_1) \mid \cdots \mid [\![\mathsf{var}\ X_k]\!](x_k,c_k) \\
&[\![\mathsf{var}\ X_i]\!](x,c) \\
&\quad \equiv \ (\boldsymbol{\nu} l)\,(\bar{l}\langle c\rangle.\mathbf{0} \mid !\,l(c).x(r,u).(\bar{r}\langle c\rangle.\bar{l}\langle c\rangle.\mathbf{0} + u(c').\bar{l}\langle c'\rangle.\mathbf{0}))
\end{aligned}
$$

図 5.4: 変数宣言に対する形式的記述

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

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．この時に渡される名前 l を通して新しいインスタンスを表す名前 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．

5.3.3 変数宣言

変数宣言 $Vdecs$ に対する時間付き π 計算による形式的記述を図 5.4 に示す．$\widetilde{x}$ が各変数を表す名前のリスト，$\widetilde{c}$ が各変数の初期値を表す名前のリストである．それぞれの変数にアクセスするには，アクセスしたい変数を表す名前 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．

5.3.4 メソッド定義

メソッド定義 $Mdecs$ 及び $Mdec$ に対する時間付き π 計算による形式的記述を図 5.5 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おり，そのインスタンスは名前 a によって表される．名前 a にはクラス定義 $[\![Cdec]\!]$ においてインスタンスを生成するたびに新たに生成される名前 a が代入される．メソッド呼び出し時の動作例を以下に示す．

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

名前 a を通してメソッド呼び出しの要求を受けると，a が表すインスタンスに属するメソッドを表す名前をすべて返す．呼び出し側で返された名前の中から呼び出したいメソッドを表す名前 m を選択し，その名前を通して引数などを送信することでメソッド呼び出しが行われる．$\widetilde{y}$ は実引数を表す名前のリストであり，それぞれの初期値として実引数の値を表す名前 $\widetilde{v}$ が設定される．初期値の設定が完了するとメソッド本体を構成する文 S の実行が開始される．$\widetilde{r}$ 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ための名前のリストである．m を通して渡される名前 t は return 文により値を返すために，名前 l は制御を次の文に移すために利用される．

5.3.5 文

文要素 S に対する時間付き π 計算による形式的記述 $[\![S]\!](l, r, \widetilde{x}, \widetilde{k})$ を図 5.6 に示す．l は文の実行完了を表す名前，r は return 文による値の返却を表す名前，$\widetilde{x}$ はメンバ変数を表す名前のリスト，$\widetilde{k}$ はプログラム中で定義されているクラスを表す名前のリストである．

$[\![\mathsf{wait}\ E]\!]$ 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．

$$
\begin{aligned}
&[\![Mdecs]\!](\widetilde{m}, a, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \ !\,a(n).\overline{n}\langle\widetilde{m}\rangle.([\![Mdec_1]\!](m_1, \widetilde{x}, \widetilde{k}) \mid \cdots \mid [\![Mdec_m]\!](m_n, \widetilde{x}, \widetilde{k})) \\
&[\![Mdec]\!](m, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \ (\boldsymbol{\nu} g, \widetilde{y}, \widetilde{r}, \widetilde{u}) \\
&\qquad (m(\widetilde{v}, t, l).\overline{y_1}\langle r_1, u_1\rangle.\overline{u_1}\langle v_1\rangle.\cdots.\overline{y_l}\langle r_l, u_l\rangle.\overline{u_l}\langle c_l\rangle.\overline{g}\langle t, l\rangle.\mathbf{0} \\
&\qquad \mid [\![\mathsf{var}\ X_1]\!](y_1, \mathsf{nil}) \mid \cdots \mid [\![\mathsf{var}\ X_l]\!](y_l, \mathsf{nil}) \mid g(t, l).[\![S]\!](l, t, \widetilde{x} \cdot \widetilde{y}, \widetilde{k}))
\end{aligned}
$$

図 5.5: メソッド定義に対する形式的記述

$[\![\mathsf{within}\ E\ \mathsf{do}\ S_1\ \mathsf{timeout}\ S_2\ \mathsf{done}]\!]$ 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．l'' の受信側では 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 の実行が開始される．

5.3.6 式

式要素 E に対する時間付き π 計算による形式的記述 $[\![E]\!](l, \widetilde{x}, \widetilde{k})$ を図 5.7 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 を表す名前，$[\![X]\!]$ の x_X は変数 X を表す名前である．

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表す名前 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．

5.3.7 Boolean クラス

Boolean クラスに対する時間付き π 計算による形式的記述を図 5.8 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

$$
\begin{aligned}
&[\![X := E]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad (\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.\bar{l}.\mathbf{0})) \\
&[\![E]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad (\boldsymbol{\nu} l')([\![E]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\bar{l}.\mathbf{0}) \\
&[\![S_1; S_2]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad (\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}) \\
&\quad \equiv \quad (\boldsymbol{\nu} l')([\![E]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(n).t[n].\bar{l}.\mathbf{0}) \\
&[\![\mathsf{return}\ E]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad (\boldsymbol{\nu} l')([\![E]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\overline{r}\langle v\rangle.\bar{l}.\mathbf{0}) \\
&[\![\mathsf{if}\ E\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{endif}]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad (\boldsymbol{\nu} l', l_1, l_2) \\
&\qquad ([\![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}.\mathbf{0} \mid f.\overline{l_2}.\mathbf{0})) \\
&\qquad \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}) \\
&\quad \equiv \quad (\boldsymbol{\nu} g, l', l_1, l_2) \\
&\qquad (\overline{g} \mid !\, g.([\![E]\!](l', \widetilde{x}, \widetilde{k}) \\
&\qquad\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}.\mathbf{0} \mid f.\overline{l_2}.\mathbf{0})) \\
&\qquad\qquad \mid l_1.[\![S]\!](g, r, \widetilde{x}, \widetilde{k}) \mid l_2.\bar{l}.\mathbf{0})) \\
&[\![\mathsf{within}\ E\ \mathsf{do}\ S_1\ \mathsf{timeout}\ S_2\ \mathsf{done}]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad (\boldsymbol{\nu} l', l'', r') \\
&\qquad ([\![E]\!](l'', \widetilde{x}, \widetilde{k}) \\
&\qquad \mid l''(n).([\![S_1]\!](l', r', \widetilde{x}, \widetilde{k}) \\
&\qquad\qquad \mid (r'(v).\overline{r}\langle v\rangle.\mathbf{0} + l'.\bar{l}.\mathbf{0} + t[n].[\![S_2]\!](l, r, \widetilde{x}, \widetilde{k}))))
\end{aligned}
$$

図 5.6: 文要素に対する形式的記述

$$
\begin{aligned}
&[\![\mathsf{true}]\!](l, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad (\boldsymbol{\nu} l', t, f)\, (\overline{k_{Bool}}\langle l', t, f\rangle.\bar{t}.l'(b).\bar{l}\langle b\rangle.\mathbf{0}) \\
&[\![\mathsf{false}]\!](l, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad (\boldsymbol{\nu} l', t, f)\, (\overline{k_{Bool}}\langle l', t, f\rangle.\overline{f}.l'(b).\bar{l}\langle b\rangle.\mathbf{0}) \\
&[\![\mathsf{nil}]\!](l, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad \bar{l}.\mathbf{0} \\
&[\![\mathit{Numeral}]\!](l, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad \bar{l}\langle \underline{\mathit{Numeral}}\rangle.\mathbf{0} \\
&[\![\mathsf{new}(A)]\!](l, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad (\boldsymbol{\nu} l')\, (\overline{k_A}\langle l'\rangle.l'(c).\bar{l}\langle c\rangle.\mathbf{0}) \\
&[\![X]\!](l, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad (\boldsymbol{\nu} r, u)\, (\overline{x_X}\langle r, u\rangle.r(v).\bar{l}\langle v\rangle.\mathbf{0}) \\
&[\![E!m(E_1, \ldots, E_n)]\!](l, \widetilde{x}, \widetilde{k}) \\
&\quad \equiv \quad (\boldsymbol{\nu} h, h_1, \ldots, h_n) \\
&\qquad ([\![E]\!](h, \widetilde{x}, \widetilde{k}) \mid [\![E_1]\!](h_1, \widetilde{x}, \widetilde{k}) \mid \cdots \mid [\![E_n]\!](h_n, \widetilde{x}, \widetilde{k}) \\
&\qquad \mid h(v).h_1(v_1).\cdots.h_n(v_n).(\boldsymbol{\nu} r, l', n)\, (\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_M}\langle v_1, \ldots, v_n, r, l'\rangle.\mathbf{0} \\
&\qquad\qquad \mid (r(v).\bar{l}\langle v\rangle.\mathbf{0} + l'.\bar{l}.\mathbf{0})))
\end{aligned}
$$

図 5.7: 式要素に対する形式的記述

$$
\begin{aligned}
&\mathrm{Boolean}(k)\\
&\quad\equiv\ (\boldsymbol{\nu} b_t, b_f)\,(\mathrm{BoolClass}(k, b_t, b_f) \mid \mathrm{Bool}_t(b_t, k) \mid \mathrm{Bool}_f(b_f, k))\\
&\mathrm{BoolClass}(k, b_t, b_f)\\
&\quad\equiv\ !\,k(l, t, f).(t.\overline{l}\langle b_t\rangle.\mathbf{0} \mid f.\overline{l}\langle b_f\rangle.\mathbf{0})\\
&\mathrm{Bool}_v(b, k)\\
&\quad\equiv\ (\boldsymbol{\nu}\widetilde{n})\,(!\,\mathrm{BoolBody}(b, \widetilde{n}) \mid !\,\mathrm{BoolVal}_v(n_1)\\
&\qquad\qquad \mid !\,\mathrm{BoolNot}(n_2, n_1, k) \mid !\,\mathrm{BoolAnd}(n_3, n_1, k))\\
&\mathrm{BoolBody}(b, \widetilde{n})\\
&\quad\equiv\ b(\widetilde{n'}).(n_1'(t, f).\overline{n_1}\langle t, f\rangle.\mathbf{0} \mid n_2'(l).\overline{n_2}\langle l\rangle.\mathbf{0} \mid n_3'(b', l).\overline{n_3}\langle b', l\rangle.\mathbf{0})\\
&\mathrm{BoolVal}_t(n_1)\\
&\quad\equiv\ n_1(t, f).\overline{t}.\mathbf{0}\\
&\mathrm{BoolVal}_f(n_1)\\
&\quad\equiv\ n_1(t, f).\overline{f}.\mathbf{0}\\
&\mathrm{BoolNot}(n_2, n_1, k)\\
&\quad\equiv\ (\boldsymbol{\nu} t, f)\,(n_2(l).\overline{n_1}\langle t, f\rangle.(t.\overline{k}\langle l, t, f\rangle.\overline{f}.\mathbf{0} \mid f.\overline{k}\langle l, t, f\rangle.\overline{t}.\mathbf{0}))\\
&\mathrm{BoolAnd}(n_3, n_1, k)\\
&\quad\equiv\ (\boldsymbol{\nu} t, f)\,(n_3(b', l).\overline{n_1}\langle t, f\rangle.\\
&\qquad (f.\overline{k}\langle l, t, f\rangle.\overline{f}.\mathbf{0}\\
&\qquad \mid t.(\boldsymbol{\nu} t', f')\,(b'(\widetilde{n'}).\overline{n_1'}\langle t', f'\rangle.(t'.\overline{k}\langle l, t, f\rangle.\overline{t}.\mathbf{0} \mid f'.\overline{k}\langle l, t, f\rangle.\overline{f}.\mathbf{0}))))
\end{aligned}
$$

図 5.8: Boolean クラスに対する形式的記述

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．Boolean クラスには Not 演算を行う機能と And 演算を行う機能が備わっている．b_t, b_f に対して名前を 3 つ送信することが可能であり，一つ目の名前によって値が真か偽かを確認できる．二つ目の名前は Not 演算，三つ目の名前は And 演算を行うためのものである．

5.4 例

5.4.1 タイマ

簡単なタイマのプログラムの例を示す．このタイマは一定時間ごとに決められた特定のタスクを行う．タスクの完了から次のタスクの開始までの時間が一定になるように動作する．この動作は時間付き π 計算では以下のように記述できる．ここでは一定時間は 10 単位時間とし，具体的なタスクの内容は省略している．

$$\begin{aligned} Spec &= (\boldsymbol{\nu} g) \\ &\quad (\overline{g}.\mathbf{0} \mid !\, g.t[\underline{10}].\boldsymbol{\nu} r\, \overline{fire}\langle r\rangle.r.\overline{g}.\mathbf{0}) \end{aligned}$$

$\overline{fire}(r)$ によってタスクが開始し，終了すると r によって通知されてくる．$Spec$ の遷移を以下に示す．

$$\begin{aligned} &Spec \\ &\xrightarrow{\tau} \equiv \\ &(\boldsymbol{\nu} g)\,(t[\underline{10}].\boldsymbol{\nu} r\, \overline{fire}\langle r\rangle.r.\overline{g}.\mathbf{0} \mid !\, g.t[\underline{10}].\boldsymbol{\nu} r\, \overline{fire}\langle r\rangle.r.\overline{g}.\mathbf{0}) \\ &\rightsquigarrow^{10} \xrightarrow{\overline{fire}(r)} \equiv \\ &(\boldsymbol{\nu} g, r)\,(r.\overline{g}.\mathbf{0} \mid !\, g.t[\underline{10}].\boldsymbol{\nu} r\, \overline{fire}\langle r\rangle.r.\overline{g}.\mathbf{0}) \\ &\xrightarrow{r} \equiv \\ &Spec \end{aligned}$$

遷移からわかるように，実行を開始するとまず 10 単位時間待機し，その後 $\overline{fire}(r)$ によってタスクを開始する．r によってタスクの終了が通知されると初期状態に戻るので，タスクの終了から開始までの間隔を 10 単位時間として動作していることがわかる．

OOL_{RT} によるプログラムを図 5.9 に示す．プログラムが実行されると初めに $Timer$ クラスのインスタンスを生成して $start$ メソッドを呼び出す．$start$ メソッドは 10 単位時間待機してから $fire$ メソッドを呼び出す．$start$ メソッドの本体は無限ループになっているため，$fire$ メソッドの実行が終了すると再度 10 単位時間待機して $fire$ メソッドを呼び出し，これを繰り返す．メインとなる処理内容は $fire$ メソッドに実装するが，具体的な処理内容については省略する．

図 5.9 のプログラムを時間付き π 計算による記述に変換した結果を付録 A に示す．プロセス $Timer$ が時間付き π 計算による記述であり，以下に示すように名前

```
 1:  program Timer 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 5.9: タイマのプログラム

k_{Timer} による通信を行って $Timer$ クラスのインスタンスを生成するところから実行が開始する．

$$
\begin{aligned}
&Timer\\
&\xrightarrow{\tau}\\
&(\boldsymbol{\nu} k_{Timer}, k_{Bool}, l')\\
&([\![Timer]\!](\widetilde{k}) \mid Boolean(k_{Bool})\\
&\mid (\boldsymbol{\nu} a)\,(\overline{l'}\langle a\rangle.(\boldsymbol{\nu} x_{this}, m_{start}, m_{fire})\\
&\qquad ([\![\mathsf{var}\ X_{this}]\!](x_{this},) \mid !\,a(n).\overline{n}\langle\widetilde{m}\rangle.([\![Mdec_{start}]\!](m_{start}, \widetilde{x}, \widetilde{k})\\
&\qquad\qquad \mid [\![Mdec_{fire}]\!](m_{fire}, \widetilde{x}, \widetilde{k}))))\\
&\mid (\boldsymbol{\nu} l, h)\,(l'(c).\overline{h}\langle c\rangle.\mathbf{0}\\
&\qquad \mid h(v).(\boldsymbol{\nu} r, l', n)\,(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{start}}\langle r, l'\rangle.\mathbf{0} \mid r(v).\bar{l}\langle v\rangle.\mathbf{0} \mid l'.\bar{l}.\mathbf{0})))
\end{aligned}
$$

10 単位時間後に $fire$ メソッドが呼び出されることを時間付き π 計算による記述を用いて確認する．プログラム上では wait 10; this!$fire()$ の部分であり，この部分に対応する形式的記述 $[\![$wait 10; this!$fire()]\!]$ の遷移の経過を図 5.10 に示す．$\rightsquigarrow^{10}$

$$
\begin{aligned}
&(\boldsymbol{\nu} l')\,((\boldsymbol{\nu} l'')\,(\overline{l''}\langle\underline{10}\rangle.\mathbf{0} \mid l''(n).t[n].\overline{l'}.\mathbf{0}) \mid l'.[\![\mathsf{this!}fire()]\!](l, r, \widetilde{x}, \widetilde{k}))\\
&\xrightarrow{\tau} \equiv\\
&(\boldsymbol{\nu} l')\,(t[\underline{10}].\overline{l'}.\mathbf{0} \mid l'.[\![\mathsf{this!}fire()]\!](l, r, \widetilde{x}, \widetilde{k}))\\
&\rightsquigarrow^{10}\xrightarrow{\tau}\\
&(\boldsymbol{\nu} l'')\,((\boldsymbol{\nu} h)\,((\boldsymbol{\nu} r, u)\,(\overline{x_{this}}\langle r, u\rangle.r(v).\overline{h}\langle v\rangle.\mathbf{0})\\
&\qquad\qquad\qquad\qquad \mid h(v).(\boldsymbol{\nu} r, l''', n)\,(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{fire}}\langle r, l'''\rangle.\mathbf{0}\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mid r(v).\overline{l''}\langle v\rangle.\mathbf{0}\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mid l'''.\overline{l''}.\mathbf{0}))\\
&\qquad \mid l''(v).\bar{l}.\mathbf{0})\\
&\xrightarrow{\overline{x_{this}}\langle r,u\rangle}\xrightarrow{r(v)}\xrightarrow{\tau} \equiv\\
&(\boldsymbol{\nu} l'')\,((\boldsymbol{\nu} r, l''', n)\,(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{fire}}\langle r, l'''\rangle.\mathbf{0} \mid r(v).\overline{l''}\langle v\rangle.\mathbf{0} \mid l'''.\overline{l''}.\mathbf{0}) \mid l''(v).\bar{l}.\mathbf{0})\\
&\xrightarrow{\overline{v}\langle n\rangle}\xrightarrow{n(\widetilde{m})}\\
&(\boldsymbol{\nu} l'')\,((\boldsymbol{\nu} r, l''')\,(\overline{m_{fire}}\langle r, l'''\rangle.\mathbf{0} \mid r(v).\overline{l''}\langle v\rangle.\mathbf{0} \mid l'''.\overline{l''}.\mathbf{0}) \mid l''(v).\bar{l}.\mathbf{0})\\
&\xrightarrow{\overline{m_{fire}}\langle r,l'''\rangle} \equiv\\
&(\boldsymbol{\nu} l'')\,(r(v).\overline{l''}\langle v\rangle.\mathbf{0} \mid l'''.\overline{l''}.\mathbf{0} \mid l''(v).\bar{l}.\mathbf{0})
\end{aligned}
$$

図 5.10: $[\![\mathsf{wait}\ 10; \mathsf{this!}fire()]\!]$ の遷移

は10回連続して $\rightsquigarrow$ によって遷移することを表す．$[\![\mathsf{wait}\ 10]\!]$ は時間待ちの長さ10を評価するプロセスと時間待ちを行うプロセスの並行合成であり，最初の τ 遷移によって評価した結果を時間待ちを行うプロセスに送信して時間待ちが開始される．時間待ちが終了したら $fire$ メソッドの呼び出しを行う．初めに this を評価する．メンバ変数から this が指すオブジェクトを表す名前を取り出す．次に this が指すオブジェクトに存在するメソッドを表す名前のリストを取得し，その中から $fire$ メソッドを表す名前 $\overline{m_{fire}}$ に送信を行う．これらから10単位時間経過した後に $fire$ メソッドが呼び出されることがわかる．この動作は $Spec$ の動作を模倣している．

$Timer$ と $Spec$ が時間待ち以外の動作について双模倣であることは，π 計算におけるプロセスの振舞等価性の判定ツールである Mobility Workbench[VM94, Vic94] を利用することによって確認した．その際，Mobility Workbench は時間に関する動作を扱うことはできないため，時間経過動作 $t[n]$ を時間待ちの発生を表す特別なアクションに置換した．また，簡単のためにタイマのタスクを $\mathbf{0}$ として省略した．この時，双模倣であることから時間待ちの発生するタイミングが同じであることがわかる．時間待ちの長さはプロセスの定義からいずれも明らかに10単位時間で同じである．

5.4.2 ペースメーカ

簡略化したペースメーカの例を示す．ペースメーカは心臓の拍動を監視しており，直前の拍動を検知してから一定時間以内に次の拍動が検知されなければ電気的刺激を発生させる．電気的刺激を発生させたらその後一定時間は不応状態に入り拍動の検知を停止する．本節ではここで述べた動作のみを対象として仕様と OOL_{RT} によるプログラムを示し，時間付き π 計算による形式的記述を与える．

ペースメーカの動作仕様は以下のように記述できる．

$$\begin{aligned} PM &= (\boldsymbol{\nu} m_s, m_g)\,(M \mid S \mid G) \\ M &= (\boldsymbol{\nu} g)\,(\overline{g}.\mathbf{0} \mid !\,g.(m_s.\overline{g}.\mathbf{0} + t[\underline{1000}].\overline{m_g}.t[\underline{250}].\overline{g}.\mathbf{0})) \\ S &= !\,pulse.\overline{m_s}.\mathbf{0} \\ G &= !\,m_g.\overline{fire}.\mathbf{0} \end{aligned}$$

ここで $pulse$ は拍動の検知を，$fire$ は電気的刺激の発生を表している．例えば，1000単位時間の間拍動が検知できず電気的刺激を発生させる動作は以下に示す遷

移によって表される．

$$
\begin{aligned}
&PM \\
&\xrightarrow{\tau} \equiv \\
&(\boldsymbol{\nu} m_s, m_g) \\
&(m_s.\overline{g}.\mathbf{0} + t[\underline{1000}].\overline{m_g}.t[\underline{250}].\overline{g}.\mathbf{0} \mid !\, g.(m_s.\overline{g}.\mathbf{0} + t[\underline{1000}].\overline{m_g}.t[\underline{250}].\overline{g}.\mathbf{0}) \mid S \mid G) \\
&\rightsquigarrow^{1000} \xrightarrow{\tau} \equiv \\
&(\boldsymbol{\nu} m_s, m_g) \\
&(t[\underline{250}].\overline{g}.\mathbf{0} \mid !\, g.(m_s.\overline{g}.\mathbf{0} + t[\underline{1000}].\overline{m_g}.t[\underline{250}].\overline{g}.\mathbf{0}) \mid S \mid \overline{fire}.\mathbf{0} \mid G) \\
&\xrightarrow{\overline{fire}} \rightsquigarrow^{250} \xrightarrow{\tau} \equiv \\
&PM
\end{aligned}
$$

図 5.11 にペースメーカを実装したプログラムの一例を示す．このプログラムでは，ペースメーカはメインの $PaceMaker$ クラス，時間計測を行い電気的刺激を発生させるか判断する $Manager$ クラス，センサを表す $Sensor$ クラス，電気的刺激を発生させる $Generator$ クラスから構成される．実行が開始されると各クラスのインスタンスの生成と関連付けが行われ，その後，$Sensor$ クラスで拍動の監視が，$Manager$ クラスで時間の計測が開始される．拍動が検知されると時間の計測がリセットされる．1000 単位時間内に拍動が検知されなければ電気的刺激を発生させて 250 単位時間待機する．その後，拍動の監視と時間の計測を再開する．

センサや電気的刺激の発生器は実際には何らかのハードウェアを用いて実装されるが，ここでは具体的なハードウェアについては考慮せず，ハードウェアを抽象化したクラスのみを考える．センサが拍動を検知すると $Sensor$ クラスの $pulse$ メソッドが呼び出され，$Generator$ クラスの $fire$ メソッドが呼び出されると電気的刺激が発生するように実装されているとする．

図 5.11 のプログラムに対する時間付き π 計算による記述を付録 B に示す．プロセス $PaceMaker$ がプログラム全体の時間付き π 計算に記述であり，名前 k_P による通信を行って $PaceMaker$ クラスのインスタンスを生成するところから実行が開始する．

拍動を待機している時の動作例を図 5.12 に示す．ここでは動作に関連するプロセスのみ示している．初めは 739 単位時間経過した時に拍動があり動作 m_{pulse} が発生したことを示している．その後拍動がなく 1000 単位時間経過したためタイムアウトし動作 m_{fire} により電気的刺激を発生させたことを示す．この動作は一定時

```
 1: program PaceMaker is
 2:
 3: class PaceMaker is
 4:     var m, s, g
 5:
 6:     method start() is
 7:         m := new(Manager)
 8:         s := new(Sensor)
 9:         g := new(Generator)
10:         s!setManager(m)
11:         m!setGenerator(g)
12:         m!setInterval(1000)
13:         m!run()
14:
15: class Manager is
16:     var gen, b, n
17:
18:     method setGenerator(g) is
19:         gen := g
20:
21:     method setInterval(i) is
22:         n := i
23:
24:     method pulse() is
25:         b := false
26:
27:     method run() is
28:         while true is
29:             within n do
30:                 while b  nil done;
31:                 b := true
32:             timeout
33:                 gen!fire();
34:                 wait 250
35:             done
36:         done
37:
38: class Sensor is
39:     var manager
40:
41:     method setManager(m) is
42:         manager := m
43:
44:     method pulse is
45:         manager!pulse()
46:
47: class Generator  is
48:     method fire() is
49:         ...
50:
51: with
52:  new(PaceMaker)!start()
```

図 5.11: ペースメーカのプログラム

$$
\begin{array}{l}
[\![S_1]\!](l', r', \widetilde{x}, \widetilde{k}) \\
\mid (r'(v).\overline{r}\langle v\rangle.\mathbf{0} + l'.\overline{l}.\mathbf{0} + t[\underline{1000}].[\![gen!fire(); \mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k})) \\
\mid (\boldsymbol{\nu} g)\,(m_{pulse}(r, l).\overline{g}\langle r, l\rangle.\mathbf{0} \mid g(r, l).[\![manager!pulse()]\!](l, r, \widetilde{x}, \widetilde{k})) \\
\mid \cdots\cdots \\
\rightsquigarrow^{739} \xrightarrow{m_{pulse}(r,l)} \xrightarrow{\tau}{}^{*} \\
[\![S_1]\!](l', r', \widetilde{x}, \widetilde{k}) \\
\mid (r'(v).\overline{r}\langle v\rangle.\mathbf{0} + l'.\overline{l}.\mathbf{0} + t[\underline{1000}].[\![gen!fire(); \mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k})) \\
\mid (\boldsymbol{\nu} g)\,(m_{pulse}(r, l).\overline{g}\langle r, l\rangle.\mathbf{0} \mid g(r, l).[\![manager!pulse()]\!](l, r, \widetilde{x}, \widetilde{k})) \\
\mid \cdots\cdots \\
\rightsquigarrow^{1000} \\
[\![gen!fire(); \mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k}) \\
\mid (\boldsymbol{\nu} g)\,(m_{pulse}(r, l).\overline{g}\langle r, l\rangle.\mathbf{0} \mid g(r, l).[\![manager!pulse()]\!](l, r, \widetilde{x}, \widetilde{k})) \\
\mid \cdots\cdots \\
\xrightarrow{\tau}{}^{*} \xrightarrow{m_{fire}(r,l)} \rightsquigarrow^{250} \xrightarrow{\tau}{}^{*} \\
[\![S_1]\!](l', r', \widetilde{x}, \widetilde{k}) \\
\mid (r'(v).\overline{r}\langle v\rangle.\mathbf{0} + l'.\overline{l}.\mathbf{0} + t[\underline{1000}].[\![gen!fire(); \mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k})) \\
\mid (\boldsymbol{\nu} g)\,(m_{pulse}(r, l).\overline{g}\langle r, l\rangle.\mathbf{0} \mid g(r, l).[\![manager!pulse()]\!](l, r, \widetilde{x}, \widetilde{k})) \\
\mid \cdots\cdots
\end{array}
$$

図 5.12: ペースメーカプログラムの形式的記述の部分プロセスの遷移

間内に次の拍動が検知されない時に電気的刺激を発生させるというペースメーカに要求される動作である．

5.5 関連研究

本章で対象としたリアルタイムオブジェクト指向言語 OOL_{RT} の基礎としたオブジェクト指向言語 OOL は π 計算によって意味論が定義されている [SW01, Wal95, Wal91]．OOL_{RT} の意味論は OOL の意味論に基づいており，時間の概念の導入によって拡張された動作意味に基づいて wait 文および timeout 文の意味を新たに与えている．

Milner は [Mil80, Mil89] において Pascal 風の手続き型言語の意味論を CCS を用いて定義している．また，Papathomas は継承機構を持つ並行オブジェクト指向言語に対し CCS への変換規則を与えることで意味を定義する方法を示している [Pap92]．

実時間性を考慮したものとしては，佐藤らによって時間拡張された CCS である RtCCS[ST92] を用いてリアルタイムオブジェクト指向言語の意味論を定義する手法が提案されている [ST94]．[ST94] で対象としているプログラミング言語は，OOL_{RT} の timeout 文の代わりにメソッド呼出し式にタイムアウト機能が付加されている点を除いて OOL_{RT} とほぼ同様の言語である．CCS に基づく体系を用いているため，オブジェクトを区別するために ID と ID を管理するプロセスを導入するなど変換規則を工夫することでオブジェクト生成やメソッド呼出しの意味を定義している．これに対し OOL_{RT} の意味論は π 計算に基づくため，オブジェクト生成やメソッド呼出しの意味は名前生成や名前通信によって自然に定義できる．

5.6 おわりに

本章では，リアルタイムオブジェクト指向言語 OOL_{RT} の構文要素に対して時間付き π 計算への変換規則を与えることで，OOL_{RT} の意味論を定義した．この変換規則により任意の OOL_{RT} プログラムに対して時間付き π 計算による形式的記述を得ることができ，時間付き π 計算の理論的な枠組みを通してプログラムの動作の厳密な解析や検証が可能になる．

本手法により，オブジェクト指向開発手法に基づいて開発される実時間システムの動作をソフトウェアプログラムを通して定式化できる．オブジェクトの生成

やオブジェクト間のメッセージ通信といった並行的な動作に加えて，タイムアウトのタイミングや時間待ちの長さなどの時間に依存する動作が定式化される．実時間性については「5 秒後にタイムアウト」のような量的性質を表現する．

プログラムと変換の例として簡単化したタイマとペースメーカの例を挙げた．仕様記述として与えられた時間付き π 計算のプロセス式と，実装のプログラムを変換して得られたプロセス式の遷移を比較することで，それぞれの間に弱い模倣性が存在することを示した．

本章で与えた変換規則は OOL_{RT} の意味を非常に細かく記述する．そのため，タイマの 10 行程度のプログラムであっても変換して得られる時間付き π 計算による記述は大きくなり，形式的記述に基づく動作解析や検証は現実的には非常に小規模なプログラムに制限される．示したい性質を保存したまま形式モデルを小さくする抽象化技法と，モジュール性に基づいてシステムを分割して部分ごとに解析あるいは検証を行う手法を確立する必要がある．

第6章　結論

6.1　本論文のまとめ

動作に時間制約が存在する並行システムである実時間システムは，その実時間性と並行性により，正しく動作することを動作テストによって確認することが難しい．実時間システムの動作の正しさを示すためには，形式手法を用いてシステムをモデル化し，システムの振舞いを詳細に網羅的に解析したり検証するための基礎技法が必要である．そこで本論文では，並行システムのモデル化に適した通信プロセスモデルの一種であるπ計算に対し時間の概念を導入して拡張した時間付きπ計算を提案した．時間付きπ計算による振舞い解析の手法として，振舞いの等価関係と擬順序関係を定義し，それらの関係が合同的性質を持つための十分条件を示した．また，振舞いの解析を行うにあたりモデルの状態数を削減するために，タイムアウトに着目しながら時間に関する振舞いを抽象化する手法を示した．時間付きπ計算による実時間システムのモデル化手法として，リアルタイムオブジェクト指向言語から時間付きπ計算による形式的記述への変換規則を与えた．

本論文ではπ計算に対して時間の経過を表すプリミティブを新たに導入し，タイムアウトの発生を表現するための意味論を定義した．時間に依存する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，我々の拡張により時間に関する振舞いをπ計算を用いて直接的に記述することができる．本論文における時間拡張の方法はプロセス代数に対して行われている時間拡張の方法に基づいており，時間に関して非常に細かい意味論を提供する．時間付きπ計算の大きな特徴は次の2点である．一つは，時間待ちの長さをメッセージとした通信が可能なことである．これにより時間待ちの長さを実行中に動的に決定することができる．もう一つは，意味論においてタイムアウトの発生を一つの遷移として明示する点である．実時間システムをモデル化できる従来の体系では，タイムアウトは暗黙的に発生する形式化が行われている．タイムアウトの発生が明示できることにより詳細な振舞い解析を行うことが可能になる．

最も基本的な振舞い解析は振舞いの等価性判定であり，時間付きπ計算におけ

る振舞いの等価関係として第3章において時間付き双模倣関係を定義した．時間付き双模倣関係は相互に動作を模倣できることを表しており，模倣可能な動作には時間経過も含まれる．また，時間に関する振舞いのみが異なることを表す擬順序関係として遅延時間順関係を定義した．遅延時間順関係は入出力動作や内部動作は同じであるが，それらの動作間で経過する時間の長さが一方が他方より常に短い，つまり速いことを表す．遅延時間順関係を利用してデッドラインなどの時間制約を満たしながら動作するか確認することができる．プロセス間の相互作用という観点から振舞い解析を行う際にはプロセスの内部動作を無視できるとよい．そこで，時間付き双模倣関係と遅延時間順関係の双方に対して内部動作を表すτ遷移を抽象した関係を定義した．また，内部動作を考慮した場合およびしない場合のいずれにおいても，時間付き双模倣関係は入力プレフィックスを除く演算子，遅延時間順関係は入力プレフィックスおよび時間独立でないプロセスの並行合成を除く演算子に対して合同であることを示した．例えば，サーバクライアントシステムでは，3.4節のストリーミング配信システムで示されたようにサーバは通常クライアントの要求に対して応答するプロセスであり，その意味で時間経過による変化はしない．このようなシステムの解析においては，並行合成の合同性が時間独立なプロセスに制限されても大きな問題とはならない．合同性に基づいてシステムをコンポーネントに分割し，コンポーネントごとに性質を調べることができる．

時間付きπ計算は時間に関する細かい意味論を提供するため，詳細な振舞い解析を可能にする反面，状態を細かく分割することから状態爆発の問題が容易に発生する．そこで第4章では，時間経過とタイムアウトの発生に着目して状態を3通りに区別することで，時間に関する情報を単に捨象するのではなく，タイムアウトの発生を捉えながら時間経過を抽象化する手法を提案した．提案する抽象化手法では，初めに展開定理に基づいて並行プロセスを逐次プロセスに変換した後，時間の経過を表すプリミティブを通常の動作へと抽象する．抽象化が行われたプロセスが双模倣であるならば，元になったプロセスは入出力動作，内部動作，直近のタイムアウト動作のみに着目すれば双模倣であることを示した．つまり，時間に着目する必要のない振舞いを解析する時に抽象化できる．抽象されたプロセスは従来のπ計算の枠組みで表現することが可能であり，従来のπ計算における種々の成果を活用することができる．

実時間システムのモデル化手法として，第5章でリアルタイムオブジェクト指向言語OOL_{RT}の意味論を時間付きπ計算によって定義した．意味論はOOL_{RT}の各

構文要素に対する時間付きπ計算による記述への変換規則として与えられる．この変換規則により実時間システムの動作をシステムを構成するソフトウェアを通してモデル化できる．また，時間付きπ計算の理論的な枠組みを用いてシステムの動作の詳細な解析あるいは検証が可能になる．簡単化したタイマとペースメーカを取り上げて，プログラムと変換の例を示した．

実時間システム開発において，本論文で提案した手法は

- 設計におけるシステムの動作記述
- 実装と設計の振舞い等価性の判定
- デッドライン仕様などの時間制約を満たすか否かの判定

などに用いることができる．本手法により実時間システムの動作の正しさを保証し，システムの信頼性の向上に寄与することが期待できる．

6.2 今後の課題

本論文では，時間付きπ計算の基礎理論として，時間付き双模倣関係と遅延時間順関係の性質および時間に関する振舞いの抽象化について述べた．また，応用として実時間システムのモデル化のためのリアルタイムオブジェクト指向言語に対する形式的記述について述べた．以下に今後の課題を挙げる．

時間制約の包含関係

遅延時間順関係は，時間制約の上限あるいは下限のどちらか一方についての関係である．しかし，例えば「入力は5秒後から15秒後の間のみ受け付ける」のように，実時間システムの動作に対する時間制約は上限と下限が同時に与えられる場合がある．この時,「7秒後から12秒後の間入力を受け付ける」動作は時間制約を満たしており，時間制約に着目すると包含関係が成り立っている．このような時間制約の包含関係による振舞い解析は有用であり，その手法の開発は今後の課題である．双模倣の考え方に基づく手法，あるいは実行系列の集合に着目してテスト等価性 [NH84, CZ91, CH93, BN95, HR95, JAK96, NC96] に基づく手法が考えられる．

実行開始からの経過時間に基づく順序関係

遅延時間順関係は，ある入出力動作，内部動作が発生してからとその次に入出力動作，内部動作が発生するまでの経過時間の違いによってプロセスの関係付けを行う．そのため，ある実行系列において任意の 2 つの入出力動作，内部動作の間の経過時間も必然的に一方のプロセスが他方のプロセスよりも短くなる．しかし，実時間システムでは特定の入出力動作の間の経過時間や実行開始からの経過時間について大小関係を示せば十分な場合も多い．時間経過によって状態が変化するプロセスの並行合成に対して保存されないため，時間制約を持つコンポーネントの組み合わせに適用できない問題もある．[Sat98, LV04b, LV05] で提案されているような初期状態からの経過時間によってプロセスの順序付ける関係について検討することは今後の課題である．

論理体系

時間付き π 計算のプロセスの性質を検査するための論理体系も今後の課題として挙げられる．並行合成を含むプロセス代数における論理体系 [HM85] や，π 計算における論理体系 [MPW93, Dam96, Dam03, Tak05] が提案されている．これらの体系を時間に関する振舞いを扱えるよう拡張し，例えばデッドラインに関する仕様を満たすことを証明できる体系を構築することが考えられる．

ツールによる支援

第 5 章の例のように，時間付き π 計算によるモデルは容易に大きくなるため，計算機による支援を行うためのツールの開発も今後の課題である．Mobility Workbench [VM94, Vic94] のようなプロセス実行のシミュレータ，等価性判定器，モデル検査器で時間付き π 計算が扱えるツールが必要である．また，時間オートマトンによるモデルからソースコードを生成する TIMES[AFM+03] のように，時間付き π 計算によるモデルからソースコードを生成するツールも有用である．

これらの課題を解決していくことで，時間付き π 計算に基づく理論的な基礎を固めるとともに，実際のソフトウェア開発における適用範囲が広くなると考えられる．

謝辞

本研究を進めるにあたり，多岐に渡って熱心な御指導御鞭撻を頂いた名古屋大学大学院情報科学研究科 阿草清滋教授に心より感謝致します．本論文の執筆にあたり，論文の構成および内容に関して有益な御意見を頂いた名古屋大学大学院情報科学研究科 坂部俊樹教授ならびに同 結縁祥治助教授に深く感謝致します．結縁祥治助教授には本論文の執筆以外にも研究の様々な局面において数々の適切な御助言を頂いたことに深く感謝致します．

また，熱心に議論し有意義な意見を頂いた愛知県立大学情報科学部 山本晋一郎助教授，名古屋大学大学院情報科学研究科 濱口毅助手をはじめ，名古屋大学阿草研究室の同期，先輩，後輩，関係者諸氏に感謝致します．

最後に，これまで温かく見守って下さった両親に感謝致します．

参考文献

[AD90] Rajeev Alur and David Dill. Automata For Modeling Real-Time Systems. In *Automata, Languages and Programming*, Vol. 443 of *LNCS*, pp. 322–335. Springer, 1990.

[AD94] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, Vol. 126, pp. 183–235, 1994.

[AFM+03] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems. In *FORMAT'03*, Vol. 2791 of *LNCS*, pp. 60–72. Springer, 2003.

[AKH92] S. Arun-Kumar and Matthew Hennessy. An Efficiency Preorder for Processes. *Acta Informatica*, Vol. 29, No. 8, pp. 737–760, 1992.

[BAK96] Amitabha Bagchi and S. Arun-Kumar. A Strong Efficiency Preorder for Mobile Processes. Technical report, IIT Delhi, 1996.

[Ber02] Martin Berger. *Towards Abstractions For Distributed Systems*. PhD thesis, Department of Computing, Imperial College, 2002.

[Ber04] Martin Berger. Basic Theory of Reduction Congruence for the Timed Asynchronous π-Calculus. In *CONCUR'2004*, Vol. 3170 of *LNCS*, pp. 115–130. Springer, 2004.

[BH00] Martin Berger and Kohei Honda. The Two-Phase Commitment Protocol in an Extended π-Calculus. In *Preliminary Proceedings of EXPRESS '00*, 2000.

[BM00] J.C.M. Baeten and C.A. Middelburg. Process Algebra with Timing: Real Time and Discrete Time. In *Handbook of Process Algebra*, chapter 10, pp. –. North-Holland, 2000.

[BN95] Michele Boreale and Rocco De Nicola. Testing Equivalence for Mobile Processes. *Information and Computation*, Vol. 120, pp. 279–303, 1995.

[Bou92] Gérard Boudol. Asynchrony and the π-calculus. Technical report, INRIA Sophia-Antipolis, 1992.

[But97] Giorgio C. Buttazzo. *HARD REAL-TIME COMPUTING SYSTEM: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.

[CH93] Rance Cleaveland and Matthew Hennessy. Testing Equivalence as a Bisimulation Equivalence. *Formal Aspects of Computing*, Vol. 5, pp. 1–20, 1993.

[Che92a] Liang Chen. An Interleaving Model for Real-Time Systems. In *LFCS'92*, Vol. 620 of *LNCS*, pp. 81–92. Springer, 1992.

[Che92b] Liang Chen. *Timed Processes: Models, Axioms and Decidability*. PhD thesis, School of Informatics, University of Edinburgh, 1992.

[Che04] Jing Chen. A Proof System for Weak Congruence in Timed π Calculus. Technical report, Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, France, 2004.

[CZ91] Rance Cleaveland and Amy E. Zwarico. A Theory fo Testing for Real-Time. In *LICS'91*, pp. 110–119. IEEE Computer Society Press, 1991.

[Dam96] Mads Dam. Model Checking Mobile Processes. *Information and Computation*, Vol. 129, No. 1, pp. 35–51, 1996.

[Dam03] Mads Dam. Proof System for π-Calculus Logics. In *Logic for Concurrency and Synchronisation*, Trends in Logic, Studia Logics Library, pp. 145–212. Kluwer, 2003.

[DLP96] Pierpaolo Degano, Jean-Vincent Loddo, and Corrado Priami. Mobile Processes with Local Clocks. In *Workshop on Analysis and Verification of Multiple-Agent Languages*, Vol. 1192 of *LNCS*, pp. 296–319. Springer, 1996.

[Dou99] Bruce P. Douglass. *Real-time UML – Second Edition*. Addison Wesley, 1999.

[FJ01] Ulrik Frendrup and Jesper Nyholm Jensen. Checking for Open Bisimilarity in the π-Calculus. Technical report, BRICS, Department of Computer Science, University of Aarbus, 2001.

[Gom00] Hassan Gomaa. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison-Wesley, 2000.

[Hen92] Matthew Hennessy. Timed Process Algebras: A Tutorial. In *Proceedings of International Summer School on Process Design Calculi*, Martoberdorf, 1992.

[HM85] Matthew Hennessy and Robin Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of the ACM*, Vol. 32, No. 1, pp. 137–161, 1985.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[HR95] Matthew Hennessy and Tim Regan. A Process Algebra for Timed Systems. *Information and Computation*, Vol. 117, No. 2, pp. 221–239, 1995.

[HT91] Kohei Honda and Mario Tokoro. An Object Calculus for Asynchronous Communication. In *Proc. The Fifth European Conference on Object-Oriented Programming*, LNCS. Springer, 1991.

[JAK96] Kamal Jain and S. Arun-Kumar. Testing Processes for Efficiency. In *FSTTCS 16*, Vol. 1180 of *LNCS*, pp. 100–110. Springer, 1996.

[Lin94] Huimin Lin. Symbolic Bisimulations and Proof Systems for the Pi-Calculus. Technical report, University of Sussex, 1994.

[Lin00] Huimin Lin. Computing Bisimulations for Finite-Control π-Calculus. *Journal of Computer Science and Technology*, Vol. 15, No. 1, pp. 1–9, 2000.

[LV01] Gerald Lüttgen and Walter Vogler. A Faster-than relation for Asynchronous Processes. In *CONCUR'01*, Vol. 2154 of *LNCS*, pp. 262–276. Springer, 2001.

[LV04a] Gerald Lüttgen and Walter Vogler. Bisimulation on Speed: A Unified Approach. Technical report, University of Augsburg, 2004.

[LV04b] Gerald Lüttgen and Walter Vogler. Bisimulation on Speed: Worst-case Efficiency. *Information and Computation*, Vol. 191, No. 2, pp. 105–144, 2004.

[LV04c] Gerald Lüttgen and Walter Vogler. Bisimulations on Speed: Lower Time Bounds. In *FoSSaCS*, Vol. 2987 of *LNCS*, pp. 333–347. Springer, 2004.

[LV05] Gerald Lüttgen and Walter Vogler. Bisimulations on Speed: Lower Time Bounds. *Theoretical Informatics and Applications*, Vol. 39, pp. 587–618, 2005.

[LY93] Kim G. Larsen and Wang Yi. Time Abstracted Bisimulation: Implicit Specifications and Decidability. In *MFPS93*, Vol. 802 of *LNCS*, pp. 160–176. Springer, 1993.

[LY97] Kim G. Larsen and Wang Yi. Time-Abstracted Bisimulation: Implicit Specifications and Decidability. *Information and Computation*, Vol. 134, No. 2, pp. 75–101, 1997.

[LŽ02] Jeremy Y. Lee and John Žic. On Modeling Real-time Mobile Processes. In *Proceedings of the twenty-fifth Australasian conference on Computer science*, pp. 139–147. Australian Computer Society, Inc., 2002.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, Vol. 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil92] Robin Milner. Functions as Processes. *Mathematical Structure in Computer Science*, Vol. 2, No. 2, pp. 119–141, 1992.

[Mil99] Robin Milner. *Communication and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[MP95] Ugo Montanari and Marco Pistore. Checking Bisimilarity for Finitary π-calculus. In *CONCUR'95*, Vol. 962 of *LNCS*, pp. 42–56. Springer, 1995.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, Part I/II. *Information and Computation*, Vol. 100, pp. 1–77, 1992.

[MPW93] Robin Milner, Joachim Parrow, and David Walker. Modal Logics for Mobile Processes. *Theoretical Computer Science*, Vol. 114, No. 1, pp. 149–171, 1993.

[MT90] Faron Moller and Chris Tofts. A Temporal Calculus of Communicating Systems. In *CONCUR'90*, Vol. 458 of *LNCS*, pp. 401–415. Springer, 1990.

[MT91] Faron Moller and Chris Tofts. Relating Processes with Respect to Speed. In *CONCUR'91*, Vol. 527 of *LNCS*, pp. 424–438. Springer, 1991.

[NC96] V. Natarajan and Rance Cleaveland. An Algebraic Theory of Process Efficiency. In *LICS '96*, pp. 63–72. IEEE Computer Society Press, 1996.

[NH84] R. De Nicola and M.C.B. Hennessy. Testing Equivalence for Processes. *Theoretical Computer Science*, Vol. 34, pp. 83–133, 1984.

[NS92] Xavier Nicollin and Joseph Sifakis. An Overview and Synthesis on Timed Process Algebras. In *Real-Time: Theory in Practice*, Vol. 600 of *LNCS*, pp. 526–548. Springer, 1992.

[OMG05a] Object Management Group. *Unified Modeling Language: Infrastructure Version 2.0*, July 2005.

[OMG05b] Object Management Group. *Unified Modeling Language: Superstructure Version 2.0*, July 2005.

[Pap92] Michael Papathomas. A Unifying Framework for Process Calculus Semantics of Concurrent Object-Oriented Laguages. In *Object-Oriented Concurrent Computing*, Vol. 612 of *LNCS*, pp. 53–79. Springer, 1992.

[PS01] Marco Pistore and Davide Sangiorgi. A Partition Refinement Algorithm for the π-Calculus. *Information and Computation*, Vol. 164, No. 2, pp. 264–321, 2001.

[San93] Davide Sangiorgi. A Theory of Bisimulation for the π-Calculus. In *CONCUR'93*, Vol. 715 of *LNCS*, pp. 127–142. Springer, 1993.

[San98] Davide Sangiorgi. On the Bisimulation Proof Method. *Mathematical Structures in Computer Science*, Vol. 8, No. 5, pp. 447–479, 1998.

[Sat98] Ichiro Satoh. An Algebraic Framework for Optimizing Parallel Programs. In *Proceedings of International Symposium on Software Engineering for Parallel and Distributed Systems*, pp. 28–38. IEEE Computer Society, 1998.

[ST92] Ichiro Satoh and Mario Tokoro. A Fomalism for Real-Time Concurrent Object-Oriented Computing. In *OOPSLA'92*, Vol. 27 of *10*, pp. 315–326. ACM Press, 1992.

[ST94] Ichiro Satoh and Mario Tokoro. Semantics for Real-Time Object-Oriented Programming Language. In *Proceedings of IEEE Conference on Computer Languages*, pp. 159–170. IEEE Computer Society, 1994.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[Tak05] 竹内泉. パイ計算による仕様を検証する論理体系. 情報処理学会論文誌：プログラミング, Vol. 46, No. SIG 11, pp. 57–65, 2005.

[UY97] Irek Ulidowski and Shoji Yuen. Extending Process Languages with Time. In *AMAST97*, Vol. 1394 of *LNCS*, pp. 524–538. Springer, 1997.

[UY04] Irek Ulidowski and Shoji Yuen. Process languages with discrete relative time based On the Ordered SOS format and rooted eager bisimulation. *The Journal of Logic and Algebraic Programming*, Vol. 60–61, pp. 401–460, 2004.

[Vic94] Björn Victor. *A Verification Tool for the Polyadic π-Calculus*. PhD thesis, Department of Computer Science, Uppsala University, Sweden, May 1994. Available as report DoCS 94/50.

[VM94] Björn Victor and Faron Moller. The Mobility Workbench — A Tool for the π-Calculus. In *CAV'94: Computer Aided Verification*, Vol. 818 of *LNCS*, pp. 428–440. Springer, 1994.

[Wal91] David Walker. π-Calculus Semantics of Object-Oriented Programming Languages. In *TACS'91*, Vol. 526 of *LNCS*, pp. 532–547. Springer, 1991.

[Wal95] David Walker. Objects in the π-Calculus. *Information and Computation*, Vol. 116, No. 2, pp. 253–271, 1995.

[Wel04] Andy Wellings. *Concurrent and Real-Time Programming in Java*. Wiley, 2004.

[Yi91a] Wang Yi. *A Calculus of Real Time Systems*. PhD thesis, Chalmers University of Technology, 1991.

[Yi91b] Wang Yi. CCS + Time = an Interleaving Model for Real Time Systems. In *Proceedings of the 18th International Colloquium on Automata, Language and Programming*, Vol. 510 of *LNCS*, pp. 217–228. Springer, 1991.

発表文献

論文誌

[KYA04a] 桑原 寛明，結縁 祥治，阿草 清滋．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．情報処理学会論文誌，Vol.45, No.6，pp.1498–1507，2004.

[KYA06] 桑原 寛明，結縁 祥治，阿草 清滋．π 計算に対する時間拡張と合同的性質．電子情報通信学会論文誌 D，Vol.J89-D，No.4，pp.632–641，2006.

国際会議

[KYA05] Hiroaki Kuwabara, Shoji Yuen and Kiyoshi Agusa. Congruence Properties for a Timed Extension of the π-Calculus. In Supplemental Volume of the DSN2005, pp.207–214, 2005.

研究会/シンポジウム

[KYA03a] 桑原 寛明，結縁 祥治，阿草 清滋．π 計算に対する時間拡張と双模倣関係．LA シンポジウム 2003 夏，2003.

[KYA03b] 桑原 寛明，結縁 祥治，阿草 清滋．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．情報処理学会オブジェクト指向シンポジウム 2003，pp.69–76，2003.

[KYA04b] 桑原 寛明，結縁 祥治，阿草 清滋．π 計算に対する時間拡張と代数的意味論．ソフトウェア工学の基礎 XI (FOSE2004)，pp.97–108，2004.

[KYA07] 桑原 寛明，結縁 祥治，阿草 清滋．時間付き通信プロセスモデルにおける時間動作の抽象化．LA シンポジウム 2006 冬，2007.

付 録A　タイマプログラムの時間付きπ計算による記述

図 5.9 のプログラムの時間付き π 計算による記述を以下に示す．$Timer$ がプログラム全体を表す時間付き π 計算のプロセス式である．実行は名前 k_{Timer} による通信から開始する．

$$
\begin{aligned}
&Timer\\
&\quad = \quad (\boldsymbol{\nu} k_{Timer}, k_{Bool})\\
&\qquad (\llbracket Timer \rrbracket(\widetilde{k}) \mid \mathrm{Boolean}(k_{Bool}) \mid (\boldsymbol{\nu} l)\, \llbracket \mathsf{new}(Timer)!start() \rrbracket(l, \mathsf{nil}, \widetilde{k}))\\
&\llbracket Timer \rrbracket(\widetilde{k})\\
&\quad = \quad !(\boldsymbol{\nu} a)\, (k_{Timer}(l).\overline{l}\langle a \rangle.(\boldsymbol{\nu} x_{this}, m_{start}, m_{fire})\\
&\qquad\qquad\qquad\qquad (\llbracket \mathsf{var}\ X_{this} \rrbracket(x_{this}, a)\\
&\qquad\qquad\qquad\qquad \mid (!\, a(n).\overline{n}\langle \widetilde{m} \rangle.(\llbracket Mdec_{start} \rrbracket(m_{start}, \widetilde{x}, \widetilde{k})\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mid \llbracket Mdec_{fire} \rrbracket(m_{fire}, \widetilde{x}, \widetilde{k})))))\\
&\llbracket \mathsf{var}\ X_{this} \rrbracket(x, c)\\
&\quad = \quad (\boldsymbol{\nu} l)\, (\overline{l}\langle c \rangle.\mathbf{0} \mid !\, l(c).x(r, u).(\overline{r}\langle c \rangle.\overline{l}\langle c \rangle.\mathbf{0} \mid u(c').\overline{l}\langle c' \rangle.\mathbf{0}))\\
&\llbracket Mdec_{start} \rrbracket(m, \widetilde{x}, \widetilde{k})\\
&\quad = \quad (\boldsymbol{\nu} g)\, (m(r, l).\overline{g}\langle r, l \rangle \mid g(r, l).\llbracket Body_{start} \rrbracket(l, r, \widetilde{x}, \widetilde{k}))\\
&\llbracket Body_{start} \rrbracket(l, r, \widetilde{x}, \widetilde{k})\\
&\quad = \quad (\boldsymbol{\nu} g, l', l_1, l_2)\\
&\qquad (\overline{g} \mid !g.(\llbracket \mathsf{true} \rrbracket(l', \widetilde{x}, \widetilde{k})\\
&\qquad\qquad \mid (\boldsymbol{\nu} t, f)\, (l'(v).\overline{v}\langle \widetilde{n} \rangle.\overline{n_1}\langle t, f \rangle.(t.\overline{l_1}.\mathbf{0} \mid f.\overline{l_2}.\mathbf{0}))\\
&\qquad\qquad \mid l_1.\llbracket \mathsf{wait}\ 10;\ this!fire() \rrbracket(g, r, \widetilde{x}, \widetilde{k})\\
&\qquad\qquad \mid l_2.\overline{l}.\mathbf{0}))
\end{aligned}
$$

$[\![\mathsf{wait}\ 10;\ this!fire()]\!](l, r, \widetilde{x}, \widetilde{k})$

$= (\boldsymbol{\nu} l')\,([\![\mathsf{wait}\ 10]\!](l', r, \widetilde{x}, \widetilde{k}) \mid l'.[\![this!fire()]\!](l, r, \widetilde{x}, \widetilde{k}))$

$[\![\mathsf{wait}\ 10]\!](l, r, \widetilde{x}, \widetilde{k})$

$= (\boldsymbol{\nu} l')\,([\![10]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(n).t[n].\bar{l}.\mathbf{0})$

$[\![10]\!](l, \widetilde{x}, \widetilde{k})$

$= \bar{l}\langle \underline{10} \rangle.\mathbf{0}$

$[\![this!fire()]\!](l, r, \widetilde{x}, \widetilde{k})$

$= (\boldsymbol{\nu} l')\,([\![this!fire()]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\bar{l}.\mathbf{0})$

$[\![this!fire()]\!](l, \widetilde{x}, \widetilde{k})$

$= (\boldsymbol{\nu} h)\,([\![this]\!](h, \widetilde{x}, \widetilde{k})$

$\qquad \mid h(v).(\boldsymbol{\nu} r, l', n)\,(\overline{v}\langle n \rangle.n(\widetilde{m}).\overline{m_{fire}}\langle r, l' \rangle.\mathbf{0} \mid r(v).\bar{l}\langle v \rangle.\mathbf{0} \mid l'.\bar{l}.\mathbf{0}))$

$[\![this]\!](l, \widetilde{x}, \widetilde{k})$

$= (\boldsymbol{\nu} r, u)\,(\overline{x_{this}}\langle r, u \rangle.r(v).\bar{l}\langle v \rangle.\mathbf{0})$

$[\![Mdec_{fire}]\!](m, \widetilde{x}, \widetilde{k})$

$= (\boldsymbol{\nu} g)\,(m(r, l).\overline{g}\langle r, l \rangle.\mathbf{0} \mid g(r, l).[\![Body^{i}_{fire}]\!](l, r, \widetilde{x}, \widetilde{k}))$

$[\![\mathsf{new}(Timer)!start()]\!](l, \widetilde{x}, \widetilde{k})$

$= (\boldsymbol{\nu} h)\,([\![\mathsf{new}(Timer)]\!](h, \widetilde{x}, \widetilde{k})$

$\qquad \mid h(v).(\boldsymbol{\nu} r, l', n)\,(\overline{v}\langle n \rangle.n(\widetilde{m}).\overline{m_{start}}\langle r, l' \rangle.\mathbf{0} \mid r(v).\bar{l}\langle v \rangle.\mathbf{0} \mid l'.\bar{l}.\mathbf{0}))$

$[\![\mathsf{new}(Timer)]\!](l, \widetilde{x}, \widetilde{k})$

$= (\boldsymbol{\nu} l')\,(\overline{k_{Timer}}\langle l' \rangle.l'(c).\bar{l}\langle c \rangle.\mathbf{0})$

付 録B　ペースメーカプログラムの時間付きπ計算による記述

図5.11のプログラムの時間付きπ計算による記述を以下に示す．$PaceMaker$がプログラム全体を表す時間付きπ計算のプロセス式である．実行は名前k_Pによる通信から開始する．

$$
\begin{aligned}
&PaceMaker \\
&\quad = \quad (\boldsymbol{\nu} k_P, k_M, k_S, k_G, k_{Bool}) \\
&\qquad (\llbracket PaceMaker \rrbracket(\widetilde{k}) \mid \llbracket Manager \rrbracket(\widetilde{k}) \mid \llbracket Sensor \rrbracket(\widetilde{k}) \mid \llbracket Generator \rrbracket(\widetilde{k}) \\
&\qquad \mid \mathrm{Boolean}(k_{Bool}) \mid (\boldsymbol{\nu} l)\llbracket \mathsf{new}(PaceMaker)!start() \rrbracket(l, \mathsf{nil}, \widetilde{k}))
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket PaceMaker \rrbracket(\widetilde{k}) \\
&\quad = \quad !\,(\boldsymbol{\nu} a)\,(k_P(l).\overline{l}\langle a \rangle.(\boldsymbol{\nu} x_m, x_s, x_g, x_{this}, m_{start}) \\
&\qquad (\llbracket \mathsf{var}\ X \rrbracket(x_m, \mathsf{nil}) \mid \llbracket \mathsf{var}\ X \rrbracket(x_s, \mathsf{nil}) \mid \llbracket \mathsf{var}\ X \rrbracket(x_g, \mathsf{nil}) \mid \llbracket \mathsf{var}\ X \rrbracket(x_{this}, \mathsf{nil}) \\
&\qquad \mid !\,a(n).\overline{n}\langle m_{start} \rangle.\llbracket Mdec_{start} \rrbracket(m_{start}, \widetilde{x}, \widetilde{k})))
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket Mdec_{start} \rrbracket(m_{start}, \widetilde{x}, \widetilde{k}) \\
&\quad = \quad (\boldsymbol{\nu} g)\,(m_{start}(\mathsf{nil}, r, l).\overline{g}\langle r, l \rangle.\mathbf{0} \mid g(r, l).\llbracket Body_{start} \rrbracket(l, r, \widetilde{x}, \widetilde{k}))
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket Body_{start} \rrbracket(l, r, \widetilde{x}, \widetilde{k}) \\
&\quad = \quad (\boldsymbol{\nu} \widetilde{l})\,(\llbracket m := \mathsf{new}(Manager) \rrbracket(l_1, r, \widetilde{x}, \widetilde{k}) \\
&\qquad \mid l_1.\llbracket s := \mathsf{new}(Sensor) \rrbracket(l_2, r, \widetilde{x}, \widetilde{k}) \\
&\qquad \mid l_2.\llbracket g := \mathsf{new}(Generator) \rrbracket(l_3, r, \widetilde{x}, \widetilde{k}) \\
&\qquad \mid l_3.\llbracket s!setManager(m) \rrbracket(l_4, r, \widetilde{x}, \widetilde{k}) \\
&\qquad \mid l_4.\llbracket m!setGenerator(g) \rrbracket(l_5, r, \widetilde{x}, \widetilde{k}) \\
&\qquad \mid l_5.\llbracket m!setInterval(1000) \rrbracket(l_6, r, \widetilde{x}, \widetilde{k})
\end{aligned}
$$

$$\quad \mid l_6.[\![m!run()]\!](l, r, \widetilde{x}, \widetilde{k}))$$

$$[\![m := \mathsf{new}(Manager)]\!](l, r, \widetilde{x}, \widetilde{k})$$

$$= (\boldsymbol{\nu} l')\,([\![\mathsf{new}(Manager)]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)\,(l'(v).\overline{x_m}\langle r, u\rangle.\overline{u}\langle v\rangle.\bar{l}.\mathbf{0}))$$

$$[\![\mathsf{new}(Manager)]\!](l, \widetilde{x}, \widetilde{k})$$

$$= (\boldsymbol{\nu} l')\,(\overline{k_M}\langle l'\rangle.l'(c).\bar{l}\langle c\rangle.\mathbf{0})$$

$$[\![s := \mathsf{new}(Sensor)]\!](l, r, \widetilde{x}, \widetilde{k})$$

$$= (\boldsymbol{\nu} l')\,([\![\mathsf{new}(Sensor)]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)\,(l'(v).\overline{x_s}\langle r, u\rangle.\overline{u}\langle v\rangle.\bar{l}.\mathbf{0}))$$

$$[\![\mathsf{new}(Sensor)]\!](l, \widetilde{x}, \widetilde{k})$$

$$= (\boldsymbol{\nu} l')\,(\overline{k_S}\langle l'\rangle.l'(c).\bar{l}\langle c\rangle.\mathbf{0})$$

$$[\![g := \mathsf{new}(Generator)]\!](l, r, \widetilde{x}, \widetilde{k})$$

$$= (\boldsymbol{\nu} l')\,([\![\mathsf{new}(Generator)]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)\,(l'(v).\overline{x_g}\langle r, u\rangle.\overline{u}\langle v\rangle.\bar{l}.\mathbf{0}))$$

$$[\![\mathsf{new}(Generator)]\!](l, \widetilde{x}, \widetilde{k})$$

$$= (\boldsymbol{\nu} l')\,(\overline{k_G}\langle l'\rangle.l'(c).\bar{l}\langle c\rangle.\mathbf{0})$$

$$[\![s!setManager(m)]\!](l, r, \widetilde{x}, \widetilde{k})$$

$$= (\boldsymbol{\nu} l')\,([\![s!setManager(m)]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\bar{l}.\mathbf{0})$$

$$[\![s!setManager(m)]\!](l, \widetilde{x}, \widetilde{k})$$

$$= (\boldsymbol{\nu} h, h_1, g_1)\,([\![s]\!](h, \widetilde{x}, \widetilde{k}) \mid g_1.[\![m]\!](h_1, \widetilde{x}, \widetilde{k})$$

$$\quad \mid h(v).\overline{g_1}.h_1(v_1).(\boldsymbol{\nu} r, l', n)\,(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{setM}}\langle v_1, r, l'\rangle.\mathbf{0} \mid r(v).\bar{l}\langle v\rangle.\mathbf{0} \mid l'.\bar{l}.\mathbf{0}))$$

$$[\![m!setGenerator(g)]\!](l, r, \widetilde{x}, \widetilde{k})$$

$$= (\boldsymbol{\nu} l')\,([\![m!setGenerator(g)]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\bar{l}.\mathbf{0})$$

$$[\![m!setGenerator(g)]\!](l, \widetilde{x}, \widetilde{k})$$

$$= (\boldsymbol{\nu} h, h_1, g_1)\,([\![m]\!](h, \widetilde{x}, \widetilde{k}) \mid g_1.[\![g]\!](h_1, \widetilde{x}, \widetilde{k})$$

$$\quad \mid h(v).\overline{g_1}.h_1(v_1).(\boldsymbol{\nu} r, l', n)\,(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{setG}}\langle v_1, r, l'\rangle.\mathbf{0} \mid r(v).\bar{l}\langle v\rangle.\mathbf{0} \mid l'.\bar{l}.\mathbf{0}))$$

$$[\![m!setInterval(1000)]\!](l, r, \widetilde{x}, \widetilde{k})$$

$$= (\boldsymbol{\nu} l')([\![m!setInterval(1000)]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\bar{l}.\mathbf{0})$$

$$[\![m!setInterval(1000)]\!](l, \widetilde{x}, \widetilde{k})$$

$$= (\boldsymbol{\nu} h, h_1, g_1)\,([\![m]\!](h, \widetilde{x}, \widetilde{k}) \mid g_1.[\![1000]\!](h_1, \widetilde{x}, \widetilde{k})$$

$$\quad \mid h(v).\overline{g_1}.h_1(v_1).(\boldsymbol{\nu} r, l', n)\,(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{setI}}\langle v_1, r, l'\rangle.\mathbf{0} \mid r(v).\bar{l}\langle v\rangle.\mathbf{0} \mid l'.\bar{l}.\mathbf{0}))$$

$$[\![m]\!](l, \widetilde{x}, \widetilde{k})$$

$$= \quad (\boldsymbol{\nu} r, u)\,(\overline{x_m}\langle r, u\rangle.r(v).\bar{l}\langle v\rangle.\mathbf{0})$$

$[\![s]\!](l, \widetilde{x}, \widetilde{k})$

$$= \quad (\boldsymbol{\nu} r, u)\,(\overline{x_s}\langle r, u\rangle.r(v).\bar{l}\langle v\rangle.\mathbf{0})$$

$[\![g]\!](l, \widetilde{x}, \widetilde{k})$

$$= \quad (\boldsymbol{\nu} r, u)\,(\overline{x_g}\langle r, u\rangle.r(v).\bar{l}\langle v\rangle.\mathbf{0})$$

$[\![1000]\!](l, \widetilde{x}, \widetilde{k})$

$$= \quad \bar{l}\langle \underline{1000}\rangle.\mathbf{0}$$

$[\![Manager]\!](\widetilde{k})$

$$\begin{aligned} = \quad & !\,(\boldsymbol{\nu} a)\,(k_M(l).\bar{l}\langle a\rangle.(\boldsymbol{\nu} x_{gen}, x_b, x_n, x_{this}, m_{setG}, m_{setI}, m_{pulse}, m_{run}) \\ & ([\![\mathsf{var}\ X]\!](x_{gen}, \mathsf{nil}) \mid [\![\mathsf{var}\ X]\!](x_b, \mathsf{nil}) \mid [\![\mathsf{var}\ X]\!](x_n, \mathsf{nil}) \mid [\![\mathsf{var}\ X]\!](x_{this}, \mathsf{nil}) \\ & \mid !\,a(n).\overline{n}\langle \widetilde{m}\rangle.([\![Mdec_{setG}]\!](m_{setG}, \widetilde{x}, \widetilde{k}) \mid [\![Mdec_{setI}]\!](m_{setI}, \widetilde{x}, \widetilde{k}) \\ & \mid [\![Mdec_{pulse}]\!](m_{pulse}, \widetilde{x}, \widetilde{k}) \mid [\![Mdec_{run}]\!](m_{run}, \widetilde{x}, \widetilde{k})))) \end{aligned}$$

$[\![Mdec_{setG}]\!](m_{setG}, \widetilde{x}, \widetilde{k})$

$$\begin{aligned} = \quad & (\boldsymbol{\nu} g, y_g, r_g, u_g)\,(m_{setG}(v, r, l).\overline{y_g}\langle r_g, u_g\rangle.\overline{u_g}\langle v\rangle.\overline{g}\langle r, l\rangle.\mathbf{0} \mid [\![\mathsf{var}\ X]\!](y_g, \mathsf{nil}) \\ & \mid g(r, l).[\![gen := g]\!](l, r, \widetilde{x} \cdot \widetilde{y}, \widetilde{k})) \end{aligned}$$

$[\![gen := g]\!](l, r, \widetilde{x} \cdot \widetilde{y}, \widetilde{k})$

$$= \quad (\boldsymbol{\nu} l')\,([\![g]\!](l', \widetilde{x} \cdot \widetilde{y}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)\,(l'(v).\overline{x_{gen}}\langle r, u\rangle.\overline{u}\langle v\rangle.\bar{l}.\mathbf{0}))$$

$[\![g]\!](l, \widetilde{x} \cdot \widetilde{y}, \widetilde{k})$

$$= \quad (\boldsymbol{\nu} r, u)\,(\overline{y_g}\langle r, u\rangle.r(v).\bar{l}\langle v\rangle.\mathbf{0})$$

$[\![Mdec_{setI}]\!](m_{setI}, \widetilde{x}, \widetilde{k})$

$$\begin{aligned} = \quad & (\boldsymbol{\nu} g, y_i, r_i, u_i)\,(m_{setI}(v, r, l).\overline{y_i}\langle r_i, u_i\rangle.\overline{u_i}\langle v\rangle.\overline{g}\langle r, l\rangle.\mathbf{0} \mid [\![\mathsf{var}\ X]\!](y_i, \mathsf{nil}) \\ & \mid g(r, l).[\![n := i]\!](l, r, \widetilde{x} \cdot \widetilde{y}, \widetilde{k})) \end{aligned}$$

$[\![n := i]\!](l, r, \widetilde{x} \cdot \widetilde{y}, \widetilde{k})$

$$= \quad (\boldsymbol{\nu} l')\,([\![i]\!](l', \widetilde{x} \cdot \widetilde{y}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)\,(l'(v).\overline{x_n}\langle r, u\rangle.\overline{u}\langle v\rangle.\bar{l}.\mathbf{0}))$$

$[\![i]\!](l, \widetilde{x} \cdot \widetilde{y}, \widetilde{k})$

$$= \quad (\boldsymbol{\nu} r, u)\,(\overline{y_i}\langle r, u\rangle.r(v).\bar{l}\langle v\rangle.\mathbf{0})$$

$[\![Mdec_{pulse}]\!](m_{pulse}, \widetilde{x}, \widetilde{k})$

$$= \quad (\boldsymbol{\nu} g)\,(m_{pulse}(v, r, l).\overline{g}\langle r, l\rangle.\mathbf{0} \mid g(r, l).[\![b := \mathsf{false}]\!](l, r, \widetilde{x}, \widetilde{k}))$$

$$
\begin{aligned}
&[\![b := \mathsf{false}]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad = \ (\boldsymbol{\nu} l')\,([\![\mathsf{false}]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)\,(l'(v).\overline{x_b}\langle r, u\rangle.\overline{u}\langle v\rangle.\bar{l}.\mathbf{0})) \\
&[\![Mdec_{run}]\!](m_{run}, \widetilde{x}, \widetilde{k}) \\
&\quad = \ (\boldsymbol{\nu} g)\,(m_{run}(\mathsf{nil}, r, l).\overline{g}\langle r, l\rangle.\mathbf{0} \mid g(r, l).[\![Body_{run}]\!](l, r, \widetilde{x}, \widetilde{k})) \\
&[\![Body_{run}]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad = \ (\boldsymbol{\nu} g, l', l_1, l_2)\,(\overline{g}.\mathbf{0} \\
&\qquad \mid !\, g.([\![\mathsf{true}]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} t, f)\,(l'(v).\overline{v}\langle \widetilde{n}\rangle.\overline{n_1}\langle t, f\rangle.(t.\overline{l_1}.\mathbf{0} \mid f.\overline{l_2}.\mathbf{0})) \\
&\qquad \mid l_1.[\![within]\!](g, r, \widetilde{x}, \widetilde{k}) \mid l_2.\bar{l}.\mathbf{0})) \\
&[\![within]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad = \ (\boldsymbol{\nu} l', l'', r')\,([\![n]\!](l'', \widetilde{x}, \widetilde{k}) \mid l''(n).([\![S_1]\!](l', r', \widetilde{x}, \widetilde{k}) \\
&\qquad \mid (r'(v).\overline{r}\langle v\rangle.\mathbf{0} + l'.\bar{l}.\mathbf{0} + t[n].[\![gen!fire(); \mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k})))) \\
&[\![n]\!](l, \widetilde{x}, \widetilde{k}) \\
&\quad = \ (\boldsymbol{\nu} r, u)\,(\overline{x_n}\langle r, u\rangle.r(v).\bar{l}\langle v\rangle.\mathbf{0}) \\
&[\![S_1]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad = \ (\boldsymbol{\nu} l')\,([\![\mathsf{while}\ b\ \mathsf{do\ nil\ done}]\!](l', r, \widetilde{x}, \widetilde{k}) \mid l'.[\![b := \mathsf{true}]\!](l, r, \widetilde{x}, \widetilde{k})) \\
&[\![\mathsf{while}\ b\ \mathsf{do\ nil\ done}]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad = \ (\boldsymbol{\nu} g, l', l_1, l_2)\,(\overline{g}.\mathbf{0} \\
&\qquad \mid !\, g.([\![b]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} t, f)\,(l'(v).\overline{v}\langle \widetilde{n}\rangle.\overline{n_1}\langle t, f\rangle.(t.\overline{l_1}.\mathbf{0} \mid f.\overline{l_2}.\mathbf{0})) \\
&\qquad \mid l_1.[\![\mathsf{nil}]\!](g, r, \widetilde{x}, \widetilde{k}) \mid l_2.\bar{l}.\mathbf{0})) \\
&[\![b]\!](l, \widetilde{x}, \widetilde{k}) \\
&\quad = \ (\boldsymbol{\nu} r, u)\,(\overline{x_b}\langle r, u\rangle.r(v).\bar{l}\langle v\rangle.\mathbf{0}) \\
&[\![b := \mathsf{true}]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad = \ (\boldsymbol{\nu} l')\,([\![\mathsf{true}]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)\,(l'(v).\overline{x_b}\langle r, u\rangle.\overline{u}\langle v\rangle.\bar{l}.\mathbf{0})) \\
&[\![gen!fire(); \mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad = \ (\boldsymbol{\nu} l')\,([\![gen!fire()]\!](l', r, \widetilde{x}, \widetilde{k}) \mid l'.[\![\mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k})) \\
&[\![gen!fire()]\!](l, r, \widetilde{x}, \widetilde{k}) \\
&\quad = \ (\boldsymbol{\nu} l')\,([\![gen!fire()]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\bar{l}.\mathbf{0}) \\
&[\![gen!fire()]\!](l, \widetilde{x}, \widetilde{k}) \\
&\quad = \ (\boldsymbol{\nu} h)\,([\![gen]\!](h, \widetilde{x}, \widetilde{k})
\end{aligned}
$$

$$\mid h(v).(\boldsymbol{\nu} r, l', n)\,(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{fire}}\langle r, l'\rangle.\mathbf{0} \mid r(v).\bar{l}\langle v\rangle.\mathbf{0} \mid l'.\bar{l}.\mathbf{0}))$$

$[\![\mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k})$

$$= (\boldsymbol{\nu} l')\,([\![250]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(n).t[n].\bar{l}.\mathbf{0})$$

$[\![250]\!](l, \widetilde{x}, \widetilde{k})$

$$= \bar{l}\langle\underline{250}\rangle.\mathbf{0}$$

$[\![Sensor]\!](\widetilde{k})$

$$= !\,(\boldsymbol{\nu} a)\,(k_S(l).\bar{l}\langle a\rangle.(\boldsymbol{\nu} x_{man}, m_{setM}, m_{pulse})\,([\![\mathsf{var}\ X]\!](x_{man}, \mathsf{nil})$$
$$\mid !\,a(n).\overline{n}\langle\widetilde{m}\rangle.([\![Mdec_{setM}]\!](m_{setM}, \widetilde{x}, \widetilde{k}) \mid [\![Mdec_{pulse}]\!](m_{pulse}, \widetilde{x}, \widetilde{k}))))$$

$[\![Mdec_{setM}]\!](m_{setM}, \widetilde{x}, \widetilde{k})$

$$= (\boldsymbol{\nu} g, y_m, r_m, u_m)\,(m_{setM}(v, r, l).\overline{y_m}\langle r_m, u_m\rangle.\overline{u_m}\langle v\rangle.\overline{g}\langle r, l\rangle.\mathbf{0} \mid [\![\mathsf{var}\ X]\!](y_m, \mathsf{nil})$$
$$\mid g(r, l).[\![manager := m]\!](l, r, \widetilde{x}\cdot\widetilde{y}, \widetilde{k}))$$

$[\![manager := m]\!](l, r, \widetilde{x}\cdot\widetilde{y}, \widetilde{k})$

$$= (\boldsymbol{\nu} l')\,([\![m]\!](l', \widetilde{x}\cdot\widetilde{y}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)\,(l'(v).\overline{x_{man}}\langle r, u\rangle.\overline{u}\langle v\rangle.\bar{l}.\mathbf{0}))$$

$[\![m]\!](l, \widetilde{x}\cdot\widetilde{y}, \widetilde{k})$

$$= (\boldsymbol{\nu} r, u)\,(\overline{y_m}\langle r, u\rangle.r(v).\bar{l}\langle v\rangle.\mathbf{0})$$

$[\![Mdec_{pulse}]\!](m_{pulse}, \widetilde{x}, \widetilde{k})$

$$= (\boldsymbol{\nu} g)\,(m_{pulse}(r, l).\overline{g}\langle r, l\rangle.\mathbf{0} \mid g(r, l).[\![manager!pulse()]\!](l, r, \widetilde{x}, \widetilde{k}))$$

$[\![manager!pulse()]\!](l, r, \widetilde{x}, \widetilde{k})$

$$= (\boldsymbol{\nu} l')\,([\![manager!pulse()]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\bar{l}.\mathbf{0})$$

$[\![manager!pulse()]\!](l, \widetilde{x}, \widetilde{k})$

$$= (\boldsymbol{\nu} h)\,([\![manager]\!](h, \widetilde{x}, \widetilde{k}) \mid h(v).(\boldsymbol{\nu} r, l', n)\,(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{pulse}}\langle r, l'\rangle.\mathbf{0}$$
$$\mid r(v).\bar{l}\langle v\rangle.\mathbf{0} \mid l'.\bar{l}.\mathbf{0}))$$

$[\![manager]\!](l, \widetilde{x}, \widetilde{k})$

$$= (\boldsymbol{\nu} r, u)\,(\overline{x_{man}}\langle r, u\rangle.r(v).\bar{l}\langle v\rangle).\mathbf{0}$$

$[\![Generator]\!](\widetilde{k})$

$$= !\,(\boldsymbol{\nu} a)\,(k_G(l).\bar{l}\langle a\rangle.(\boldsymbol{\nu} m_{fire})\,(!\,a(n).\overline{n}\langle m_{fire}\rangle.[\![Mdec_{fire}]\!](m_{fire}, \widetilde{x}, \widetilde{k})))$$

$[\![Mdec_{fire}]\!](m_{fire}, \widetilde{x}, \widetilde{k})$

$$= (\boldsymbol{\nu} g)\,(m_{fire}(\mathsf{nil}, r, l).\overline{g}\langle r, l\rangle.\mathbf{0} \mid g(r, l).[\![Body_{fire}]\!](l, r, \widetilde{x}, \widetilde{k}))$$

$[\![\mathsf{new}(PaceMaker)!start()]\!](l, \mathsf{nil}, \widetilde{k})$

$$= (\boldsymbol{\nu} h)\,([\![\mathsf{new}(PaceMaker)]\!](h, \mathsf{nil}, \widetilde{k}) \mid h(v).(\boldsymbol{\nu} r, l', n)\,(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{start}}\langle r, l'\rangle.\mathbf{0} \mid r(v).\bar{l}\langle v\rangle.\mathbf{0} \mid l'.\bar{l}.\mathbf{0}))$$

$[\![\mathsf{new}(PaceMaker)]\!](l, \widetilde{x}, \widetilde{k})$

$$= (\boldsymbol{\nu} l')\,(\overline{k_P}\langle l'\rangle.l'(c).\bar{l}\langle c\rangle.\mathbf{0})$$