計算理工学専攻修士学位論文

実時間ソフトウェアの構成的開発のためのπ計算に対する時間拡張

A Timed Extension of π-Calculus for Compositional Development of Real-Time Software

名古屋大学大学院工学研究科
計算理工学専攻

桑 原　寛 明

2003年2月

指導教官　　阿草 清滋　教授

概要

本論文では，実時間ソフトウェア開発のためのπ計算に対する時間拡張を提案し，リアルタイムオブジェクト指向言語に対し時間拡張したπ計算による記述への変換規則を与える．

時間拡張したπ計算は時間経過とタイムアウトを表現できる．従来のπ計算に対し決められた長さだけ時間待ちするアクションを導入する．時間の経過は構造動作意味によって定義される特別な遷移として表す．時間の経過も含めて振舞いが同等であれば，時間を無視しても同じように振舞うことを示す．

時間拡張したπ計算を用いて実時間ソフトウェアの振舞いを厳密に記述する．形式的記述により動作検証を機械的に行うことができ，実装前に正しく仕様に従って振舞うか確認することが可能になる．ソフトウェアプログラムを変換規則を使って変換することで時間拡張したπ計算による記述が得られる．振舞いの記述とプログラムから変換して得られた記述を比較することで実装の正しさを確認できる．ペースメーカーの例を用いてどのような記述が得られるか示す．

abstract

In this paper, we propose a timed extension of the π-calculus for real-time software development. We present a collection of translation rules that map the syntactic constructions of a real-time object-oriented language into expressions in a time-extended π-calculus.

A time-extended π-calculus which is an extension of the π-calculus enables us to describe the passage of time and timeout. We introduce an action which means waiting for given time. The passage of time is represented as special transition which is defined by the structured operational semantics. We show that when two processes are bisimilar including the passage of time, they behave equivalently even if time is not considered.

We can describe the behaviors of real-time systems strictly using a time-extended π-calculus. These descriptions enable to verify the behaviors of the systems automatically. Translation rules convert software programs that consist of real-time systems to expressions in a time-extended π-calculus. It is possible to check the correctness of implementation by comparing the descriptions of the behavior and the expressions converted from implementation. We take a concrete description of a pacemaker for instance.

目 次

第1章　はじめに

1.1　背景

今日我々の周囲では数多くの実時間システムが用いられている．例えば，ペースメーカー，ブレーキングシステム，携帯電話，カーナビゲーションシステムなど，その応用範囲は多岐に渡る．これら実時間システムは実時間性を持ち，その性質がシステムの動作に大きく影響する．そのため実時間システムの開発では設計段階から時間を明示的にモデル化することや，実時間性により存在する動作の非決定性を十分に表現することが必要である．

実時間システムの中にはペースメーカーのように誤動作の発生が人命にかかわるため非常に高い信頼性が要求されるシステムや，携帯電話のように開発サイクルが短くなり生産性が要求されるシステムがある．そのため信頼性の保証や生産性の向上といった観点から，システムが開発者の意図通りに動作するか実装後のテストだけでなく設計段階から確認することが重要となっている．

しかし，従来の開発手法においては時間を明示せずに設計を行うことが多い．動作についてもいくつかの動作のみを記述し，非決定性に関しては十分に表現しないことが多い．設計段階ではシステムの動作に関して十分に記述がなされないため正しく動作するか確認することが難しくなる．そこで実装後にシステムを実際に動作させてテストを行うが，これにはいくつか問題点がある．

一つ目の問題点として，実装後のテストなので設計に問題が発見された場合もう一度実装し直さなければならず納期に影響する．二つ目に，実時間システ

ムは動作の非決定性により複数の実行トレースを持つため，トレースの数が膨大になる場合すべてのトレースに対しテストすることが困難になる．三つ目に，動作テストによりシステムの動作が仕様に適合してるかは確認できるが，設計通りに実装されたか確認することができないという問題がある．

実時間システムは複数のプロセスから構成される．プロセスは並行に動作し相互にメッセージ通信を行う．このことはシステムを相互にやり取りする複数のオブジェクトの集合であるとしてモデル化するオブジェクト指向と類似している．そこで実時間システムの開発にオブジェクト指向開発を適用する．オブジェクト指向には，ソフトウェアの部品化による再利用性の向上，変更に対する影響範囲の限定による保守性の向上，といった利点がある．オブジェクト指向開発の分析・設計ではシステムの動的な振舞いやシステムを構成する個々のオブジェクトの振舞い，システムの動作に伴って発生するオブジェクト間のメッセージ通信を表現するために UML[1] のステートチャートやシーケンス図を用いる．これらは図的表現であるためシステムの動作を直感的に把握することは容易である．しかしすべての実行トレースの列挙や正しい動作との機械的な比較が難しく動作検証が行いにくい．

1.2　概要

本論文では信頼性や生産性の向上を目的として，実時間システムの開発に形式的手法によるアプローチを導入する．特に実時間システムを構成するソフトウェアの設計と実装に注目し，形式的な検証が可能となるようにそれらに対し形式的記述を与える．形式的手法としてプロセス代数の一つである π 計算 [2, 3] を用いる．実時間システムを記述するために時間の概念のない π 計算を時間に関して拡張する．

π 計算はポート名の送受信や動的生成を表すことが可能であり，動的なプロ

セスを記述できる点で高い表現能力を持っている．ポート名の送受信や動的生成は，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．時間に関して拡張することで時間経過の表現を可能にし，時間制約が存在する動作を記述する．実時間システムの動作をプロセス代数を用いて表すことで

- 実時間性，並行性が記述しやすくなる
- システムの動作を厳密に表現できる
- モデルチェッキング [4] などの形式的手法により動作検証できる

という利点が得られる．

動作の正しさは設計の正しさと実装の正しさに依存する．実装の正しさの確認にも形式的手法を適用する．具体的には，実装時に作られたソフトウェアのプログラムをプロセス代数での表現に変換し，分析・設計時に作成されたシステム動作の記述と比較することで，設計に従って動作するよう実装されたか確認する．本論文では一つの簡単なリアルタイムオブジェクト指向言語を定義し，言語の各構文要素に対し π 計算による記述を与える．この記述はオブジェクト指向言語から π 計算による記述への変換規則となっている．この規則によりプログラムから π 計算での表現を得る．

1.3　構成

本論文の構成は以下の通りである．

2 章で実時間システムの従来手法による開発と形式的記述を用いる開発について述べる．3 章では形式的記述として採用した π 計算の概要と，π 計算に対する時間拡張について述べる．4 章でリアルタイムオブジェクト指向言語に対

し時間拡張したπ計算による記述を与える．5章で形式的記述を用いる開発の具体例を簡単化したペースメーカーを取り上げて示す．最後にまとめと今後の課題を述べる．

第2章　実時間システム開発

2.1　実時間システム

実時間システムとは時間制約を持つ並行システムである [5]．実時間システムは並行に動作する複数のプロセスからなり，プロセス間の相互作用により計算を行いシステムの振舞いを決定する．システムの振舞いの正しさは計算の結果と応答までの時間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ではGPS(Global Positioning System) により取得した位置情報に基づき進行方向や目的地までの距離などを計算してモニターに表示する．しかし車は移動しているため一定時間以内に計算が完了しなければ，カーナビゲーションシステムから得られる情報と実際の状況の誤差が大きくなる．カーナビゲーションシステムでは時間制約に対する違反が即座に大きな問題になるとは考えにくいが，ペースメーカーのように時間制約が守られなければ即座に人命にかかわるシステムもある．

実時間システムは時間制約に対する要求により以下のように分類できる [6, 7]．

ハード実時間システム

時間制約を必ず満たさなければならないシステムを指す．ある動作が時間制約を満たさない場合その動作は誤動作でありシステムが障害状態となる

ソフト実時間システム

常に時間制約を満たさなければならないということはなく，時には満た

されなくてもよい，あるいは多少の時間的なずれは容認されるシステムを指す

動作が時間に依存するシステムの開発では開発の初期段階から時間を明示することが重要である．簡単な例としてマウスのダブルクリックの判定について考える．マウスのボタンが続けて 2 回クリックされた時，それをダブルクリックと判定するかシングルクリック 2 回と判定するかは 1 度目のクリックから 2 度目のクリックまでに経過した時間に依存する．時間についての記述がなければこれらを区別することはできない．このように時間に依存した動作を正確に表すためには，時間をモデル化し設計段階から記述する必要がある．

2.2　従来手法による開発

実時間システムの開発にもオブジェクト指向開発を適用することが多い．オブジェクト指向開発では図 2.1 に示す作業を繰り返し行うことで開発を進める [8, 9].

分析

- 対象システムの本質的な特性を定義し，システムの機能の詳細を特定する
- システムの主要なオブジェクトを識別しそれらの間の関係を導出してシステムの静的構造を定義する
- システムを構成するそれぞれのオブジェクトの振舞いやオブジェクト間の相互作用など動的振舞いを定義する

設計

分析モデルに基づき，パッケージやタスクといった大きなレベルでのソ

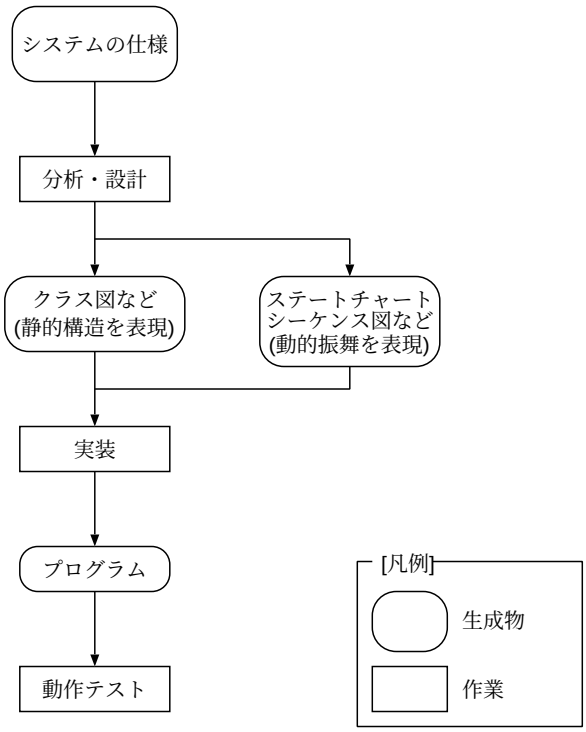

図 2.1: 従来手法による開発の流れ

フトウェア構造や複数オブジェクトの協調動作メカニズム，個々のクラス内のデータ構造やアルゴリズムを記述する

実装

設計に基づきオブジェクト指向言語によりプログラミングする

テスト

実装したシステムの動作が仕様に従っているか確認する

分析により定義されたシステムの構造や振舞いは UML[6] を用いて記述される．システムの静的構造の記述には主にクラス図を用いる．動的振舞いの記述にはステートチャートやシーケンス図などを用いる．ステートチャートはオブ

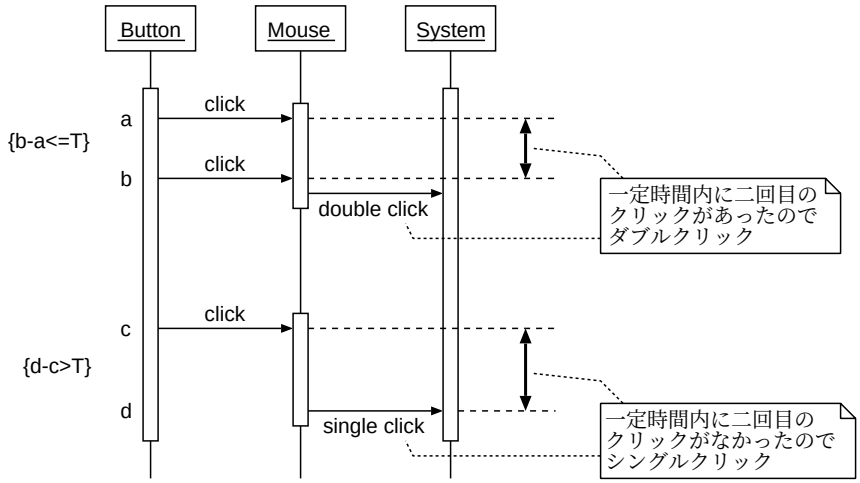

図 2.2: マウスのダブルクリックを表すシーケンス図

ジェクトの状態遷移を表現する．シーケンス図はオブジェクト間でやり取りされるメッセージの流れを時間軸に沿って表現する．

しかし，実時間システムの動作は非決定性により一意に決まらないがシーケンス図により記述されるのはある一つの動作についてのみである．ステートチャートでは個々のオブジェクトの動作の把握は容易だが，システム全体の時間軸に沿った動作の把握は難しい．UML を用いてシステムの動的振舞いを記述するとそれぞれ観点の異なる様々な図を用いることになり，一貫性を保つための管理コストが発生する．UML では時間制約を表現する統一的な方法が定められていないという問題もある．マウスのダブルクリックの動作をシーケンス図を用いると図 2.2 のように表すことができるが，これは一つの実行例に過ぎない．

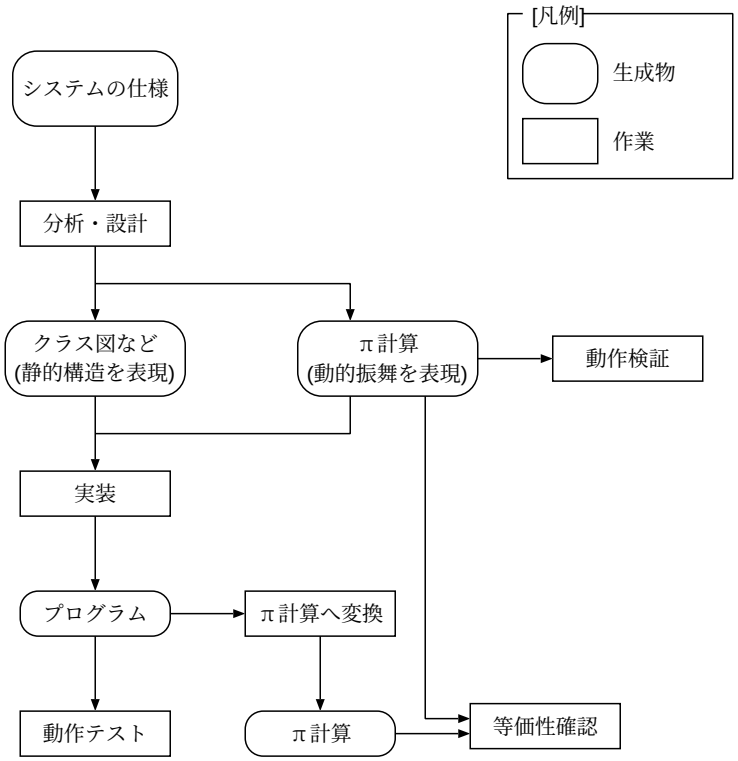

図 2.3: 形式的記述を用いた開発の流れ

2.3　形式的記述を用いた開発

実時間システムの多くは並行システムであり，並行計算モデルを用いたモデル化が可能である．プロセス代数を用いて形式的に記述することでシステムの動作を実時間性も含めてより厳密に表現できる．本論文では実時間システムを記述するプロセス代数として時間拡張したπ計算を提案する．π計算はポート名の送受信や動的生成を表すことが可能であるためオブジェクトの生成やメッセージ通信の表現が容易である．実時間システムの時間制約を表すために時間の概念を導入して拡張する．

形式的記述を用いる開発の流れを図 2.3 に示す．オブジェクト指向開発を適

用する点は従来手法と同様である．以下の変更を従来手法に加える．

(a). 動的振舞いの記述に図的表現ではなく π 計算を用いる

(b). 実装前に (a) の記述に対して検証を行う

(c). 実装で作成されたプログラムを π 計算の記述に変換し，(a) の記述との等価性を確認する

プロセス代数を用いることで動作検証をモデルチェッキングにより数学的, 機械的に行うことが可能になる．設計時に作られた動的振舞いの記述に対して動作検証を行い，仕様を満たす設計がなされているか確認する．

例えば，マウスのダブルクリックの動作は以下の式で表される．

$$\begin{aligned} Mouse &= click.M_1 \\ M_1 &= click.\overline{double}.Mouse + t[n].\overline{single}.Mouse \end{aligned}$$

ここで $click$ はボタンがクリックされたことを表し，$t[n]$ は長さ n の時間待ちを表す．初期状態は $Mouse$ で，1 度目のクリックがあると状態 M_1 に遷移する．M_1 では時間 n だけ 2 度目のクリックの発生を待機する．時間 n が経過する前に 2 度目のクリックがあればダブルクリックと判定し，クリックがなければシングルクリックと判定する．

システムの設計の正しさだけではなく実装の正しさも形式的記述を利用して確認する．オブジェクト指向開発では実装はオブジェクト指向言語によって行う．そこでオブジェクト指向言語に対し π 計算による記述を与え，実装で作成されたプログラムを π 計算による表現に変換する．得られた記述と設計段階の動的振舞い記述の等価性を調べることで，振舞いに関して設計に従って正しく実装されたか確認する．

第3章　π計算に対する時間拡張

3.1　π計算

Milner らによって提案された π 計算 [10] はプロセス代数の一つであり並行計算のモデルである．並行に動作するプロセスそれぞれの振舞いと，プロセス間の相互作用を記述する．π 計算ではプロセス間通信に用いるリンクを**名前**と呼ぶ．名前が最も基本的な要素である．同様なプロセス代数として CCS[11] や CSP[12] などがあるが，π 計算では名前自身をメッセージとして送受信することでプロセス間の接続の変更を表現できる点が異なり，それにより高い表現能力を持つ．

3.1.1　構文

名前の集合を $\mathcal{N}$ で表し，$\overline{\mathcal{N}} = \{\overline{x} \mid x \in \mathcal{N}\}$，$\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}}$ とする．

π 計算のプロセス式 P は以下のように定義される．ここで $a, x \in \mathcal{N}$，$\vec{y}$ は名前の並びを表す．

$$
\begin{aligned}
\pi &::= x(\vec{y}) \mid \overline{x}\langle\vec{y}\rangle \mid \tau \\
P &::= M \mid P_1 | P_2 \mid \boldsymbol{\nu} a\, P \mid !\, P \\
M &::= \mathbf{0} \mid \pi.P \mid M_1 + M_2
\end{aligned}
$$

プレフィックス π はプロセスが次に実行可能なアクションを表し，以下に示す 3 種類がある．

- $x(\vec{y})$: $\vec{y}$ を x を通して受信
- $\overline{x}\langle\vec{y}\rangle$: $\vec{y}$ を x を通して送信
- τ : 外部から不可視な内部アクション

演算子の意味は以下に示す通りである.

- $P+Q$: 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．ともに実行可能であれば非決定的に選択する
- $P|Q$: プロセス P,Q の並行動作を表す
- $\boldsymbol{\nu} a\, P$: プロセス P 中の自由な名前 a を束縛する
- $!P$: 無限回の繰り返しを表す．無限個のプロセス P が $|$ 演算子によって結合しているとみなす

π 計算のプロセス式全体の集合を $\mathcal{P}$ で表す．またアクションの集合を $Act = \mathcal{L} \cup \{\tau\}$ で表す.

3.1.2　動作意味

リアクション

$\mathcal{P}$ 上のリアクション関係 $\rightarrow$ は表 3.1 のリアクション規則によって定義される．$\equiv$ は $+$ 演算と $|$ 演算の交換結合関係から生成される構造等価性を表す.

リアクションはプロセス内部で発生し外部から観察不可能なアクションによる遷移を意味する．TAU 規則は内部アクション τ による遷移，REACT 規則は並行動作するサブプロセス間での名前通信による遷移を表す．これらの規則が適用できる時リアクションが可能である.

$$\text{TAU}: \ \frac{}{\tau.P + M \to P}$$

$$\text{REACT}: \ \frac{}{(x(\vec{y}).P + M) \mid (\overline{x}\langle\vec{z}\rangle.Q + N) \to P\{\vec{z}/\vec{y}\} \mid Q} \text{ if } |\vec{y}| = |\vec{z}|$$

$$\text{PAR}: \ \frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad \text{RES}_{\text{R}}: \ \frac{P \to P'}{\boldsymbol{\nu} x\, P \to \boldsymbol{\nu} x\, P'}$$

$$\text{STRUCT}: \ \frac{P \to P'}{Q \to Q'} \text{ if } P \equiv Q,\ P' \equiv Q'$$

表 3.1: リアクション規則

ラベル付き遷移

$\mathcal{P}$ 上の遷移関係 $\{\xrightarrow{\alpha} \mid \alpha \in Act\}$ は表 3.2 の遷移規則によって定義される．

ラベル付き遷移はプロセスが他のプロセスとどのように通信を行うことができるかを示す．$P \xrightarrow{\alpha} P'$ は入力，出力，内部アクションのいずれかのアクションにより P から P' に遷移することを表す．

3.2　時間拡張

時間の経過が意味を持つ動作を記述するため，遅延とタイムアウトの性質を記述できるように構文と動作意味を拡張する [13]．

3.2.1　構文

従来の π 計算に時間経過アクション t を導入する．タイムアウトまでの時間を $[n]$ と表し，n 単位時間でタイムアウトする時間待ちを $t[n]$ と記述する．プロセス式は以下のように定義される．

$$\pi \ ::= \ x(\vec{y}) \mid \overline{x}\langle\vec{y}\rangle \mid \tau \mid t[n]$$

$$\text{ACT}: \frac{}{\alpha.P \xrightarrow{\alpha} P}$$

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

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

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

$$\text{RES}: \frac{P \xrightarrow{\alpha} P'}{\boldsymbol{\nu} x\, P \xrightarrow{\alpha} \boldsymbol{\nu} x\, P'} \text{ if } \alpha \notin \{x, \overline{x}\} \qquad \text{REP}: \frac{P \mid !P \xrightarrow{\alpha} A}{!P \xrightarrow{\alpha} A}$$

表 3.2: 遷移規則

$$P \;::=\; M \mid P_1 | P_2 \mid \boldsymbol{\nu} a\, P \mid !\, P$$

$$M \;::=\; \mathbf{0} \mid \pi.P \mid M_1 + M_2$$

例えばプロセス $t[5].P$ は 5 単位時間後にプロセス P へ遷移する．時間経過アクションと非決定的選択を用いることでタイムアウトによる分岐を記述することができる．$a.P + t[5].Q$ というプロセスはアクション a の発生まで 5 単位時間待機する．5 単位時間経過する前に a が発生すればプロセス P，発生しなければタイムアウトしプロセス Q へ遷移する．

3.2.2　動作意味

表 3.1 のリアクション規則，表 3.2 の遷移規則を拡張して時間経過アクションに関する規則を導入する．また時間経過に関する規則を新たに導入する．この時間拡張では以下の条件を前提に持つ．

- 時間の経過は時間経過規則による遷移によってのみ発生し，他の規則に

$$\text{TIMEOUT}_{\text{R}}\text{-S}: \ \frac{}{t[n].P + M \to P} \text{ if } M \not\to$$

$$\text{TIMEOUT}_{\text{R}}\text{-P}: \ \frac{}{t[m].P \mid t[n].Q \to P \mid t[n'].Q} \text{ if } m \leq n$$

表 3.3: 追加するリアクション規則

$$\text{TIMEOUT-L}: \ \frac{}{t[0].P + M \xrightarrow{\tau} P}$$

$$\text{TIMEOUT-R}: \ \frac{}{M + t[0].P \xrightarrow{\tau} P}$$

表 3.4: 追加する遷移規則

よる遷移では時間は経過しない

- τ 遷移は時間経過に優先する
- 離散時間を扱う

リアクション

新たに追加するリアクション規則を表 3.3 に示す．リアクションはプロセス間で発生するメッセージ通信にのみ着目し，単位時間ごとの時間の経過には注目しない．そのためリアクションでは，時間経過アクションに対しタイムアウトまでの時間待ちと次のプロセスへの遷移を一度の遷移で実行する．

ラベル付き遷移

新たに追加する遷移規則を表 3.4 に示す．プレフィックス $t[0]$ は τ と同等であるとみなし，タイムアウトは τ 遷移であるとする．

ラベル付き遷移では単位時間ごとの時間の経過にも注目するため，表 3.5 に

$$\mathrm{PASS_T}:\ \frac{}{t[n].P \rightsquigarrow t[n-1].P}\ \text{if } n>0$$

$$\mathrm{OUT_T}:\ \frac{}{\overline{x}\langle y\rangle.P \rightsquigarrow \overline{x}\langle y\rangle.P} \qquad \mathrm{IN_T}:\ \frac{}{x(y).P \rightsquigarrow x(y).P}$$

$$\mathrm{SUM_T}:\ \frac{P \rightsquigarrow P' \quad Q \rightsquigarrow Q'}{P+Q \rightsquigarrow P'+Q'} \qquad \mathrm{PAR_T}:\ \frac{P \rightsquigarrow P' \quad Q \rightsquigarrow Q'}{P \mid Q \rightsquigarrow P' \mid Q'}\ \text{if } P \mid Q \overset{\tau}{\not\to}$$

$$\mathrm{RES_T}:\ \frac{P \rightsquigarrow P'}{\boldsymbol{\nu} x\, P \rightsquigarrow \boldsymbol{\nu} x\, P'} \qquad \mathrm{REP_T}:\ \frac{P \rightsquigarrow P'}{!P \rightsquigarrow !P'}\ \text{if } P \mid P \overset{\tau}{\not\to}$$

$$\mathrm{STRUCT_T}:\ \frac{P \rightsquigarrow P'}{Q \rightsquigarrow Q'}\ \text{if } P \equiv Q,\ P' \equiv Q'$$

表 3.5: 時間経過規則

示す時間経過規則を新たに導入する．1 単位時間の経過による遷移を $\rightsquigarrow$ によって表す．時間はすべてのプロセスにおいて同時に経過する．時間経過によりプレフィックス $t[n]$ は $t[n-1]$ となり，タイムアウトまでの時間が 1 単位時間短くなったことを示す．プレフィックスでない $t[n]$ の n の値は変化しない．τ 遷移が時間経過に優先するという条件により，$\mathrm{PAR_T}$ 規則と $\mathrm{REP_T}$ 規則には τ 遷移が発生しない場合という条件が付加される．

3.3　性質

ここでは 2 つのプロセスの振舞いが時間の経過による遷移を含めて等しいならば，時間の経過を無視した場合の振舞いも等しいことを示す．

定義 1 名前 μ に対しプロセス P が μ による入力あるいは出力アクションが可能であることを $P \downarrow_\mu$ と表す．

定義 2 以下を満たす対称な関係 $\mathcal{S}$ で最大の関係を $\dot{\sim}_t$ とする．

$(P, Q) \in \mathcal{S}$ の時，

- $P \downarrow_\mu$ ならば $Q \downarrow_\mu$
- $P \rightsquigarrow^n \xrightarrow{\tau} P'$ ならば $\exists Q'.Q \rightsquigarrow^n \xrightarrow{\tau} Q'.(P', Q') \in \mathcal{S}$　$(n \geq 0)$　□

$\rightsquigarrow^n \xrightarrow{\tau}$ は n 回連続して $\rightsquigarrow$ により遷移した後 $\xrightarrow{\tau}$ で遷移することを表す．$\rightsquigarrow^0 \xrightarrow{\tau} = \xrightarrow{\tau}$ とする．

定義 3 以下を満たす対称な関係 $\mathcal{S}$ で最大の関係を $\overset{*}{\sim}_t$ とする．

$(P, Q) \in \mathcal{S}$ の時，

- $P \downarrow_\mu$ ならば $Q \downarrow_\mu$
- $P \longrightarrow P'$ ならば $\exists Q'.Q \longrightarrow Q'.(P', Q') \in \mathcal{S}$　□

直感的には，$(P, Q) \in \dot{\sim}_t$ であればプロセス P と Q は時間が n 経過した後も振舞いは等しい．$(P, Q) \in \overset{*}{\sim}_t$ であれば P と Q は時間を無視すれば同じように振舞う．時間の概念のない従来の π 計算において定義 3 により得られる関係を $\overset{*}{\sim}$ とすると $\overset{*}{\sim} \subset \overset{*}{\sim}_t$ となる．拡張された部分は時間待ちアクション $t[n]$ に関する規則によるもののみであり，時間待ちアクションがなければ $\overset{*}{\sim}$ と等価である．

$\dot{\sim}_t \subset \overset{*}{\sim}_t$ であることを示す．このことにより 2 つのプロセスの振舞いが時間の経過を含めて等しい時，時間を無視した場合も等しいことがわかる．

補題 1　$P \rightsquigarrow^n \xrightarrow{\tau} P' \ \Rightarrow \ P \longrightarrow P'$　$(n \geq 0)$

証明： P の構造に関する帰納法による．

基底段階

- $P \equiv \tau.P'$ の時，TAU 規則より明らか．
- $P \equiv t[n].P'$ の時，$\mathrm{TIMEOUT_R}$-S 規則より明らか．

帰納段階

- $P \equiv Q + R$ の時，帰納法の仮定から $Q \rightsquigarrow^n Q' \xrightarrow{\tau} Q'', R \rightsquigarrow^n R' \xrightarrow{\tau} R''$ とできる．$\mathrm{SUM_T}$ 規則の適用により $P \rightsquigarrow^n Q' + R'$ となる．ここで $Q' = \tau.Q''$ とできるので TAU 規則から $Q' + R' \xrightarrow{\tau} Q'' = P'$ となる．さらに帰納法の仮定から $Q \longrightarrow Q''$ である．これは TAU 規則あるいは $\mathrm{TIMEOUT_R}$-S 規則の適用であり，同様の規則を用いて $P \longrightarrow Q'' = P'$ が得られる．
- $P \equiv Q \mid R$ の時，$n = 0$ ならば帰納法の仮定から $P' = Q' \mid R$ とできる．この時帰納法の仮定と PAR 規則から $Q \mid R \longrightarrow Q' \mid R = P'$ が得られる．$n \geq 1$ ならば帰納法の仮定から $Q \rightsquigarrow^n Q' \xrightarrow{\tau} Q'', R \rightsquigarrow^n R' \xrightarrow{\tau} R''$ とすると $P' = Q'' \mid R'$ とできる．この時 $\mathrm{TIMEOUT_R}$-P 規則により $P \longrightarrow P'$ となる．
- $P \equiv \boldsymbol{\nu} x\, Q$ の時，帰納法の仮定から $Q \rightsquigarrow^n Q' \xrightarrow{\tau} Q''$ で，$\mathrm{RES_T}$ 規則と RES 規則から $P \rightsquigarrow^n \boldsymbol{\nu} x\, Q' \xrightarrow{\tau} \boldsymbol{\nu} x\, Q'' = P'$ となる．また帰納法の仮定から $Q \longrightarrow Q''$ であり，$\mathrm{RES_R}$ 規則を適用すると $\boldsymbol{\nu} x\, Q \longrightarrow \boldsymbol{\nu} x\, Q''$ が得られる．よって $P \longrightarrow P'$ が成り立つ．
- $P \equiv !Q$ の時，$n = 0$ ならば帰納法の仮定から $P' = Q'' \mid !Q$ とできる．この時帰納法の仮定と STRUCT 規則，PAR 規則から $!Q \longrightarrow Q'' \mid !Q = P'$ が得られる．$n \geq 1$ ならば帰納法の仮定から $Q \rightsquigarrow^n Q' \xrightarrow{\tau} Q''$ とすると $P' = Q'' \mid !Q$ とできる．この時 $\mathrm{TIMEOUT_R}$-P 規則により $P \longrightarrow P'$ となる．　□

定理 1 $\dot{\sim}_t \subset \overset{*}{\sim}_t$

証明： $(P, Q) \in \dot{\sim}_t$ とする．この時 $P \rightsquigarrow^{n\,\tau\!\rightarrow} P'$ とすると，$Q \rightsquigarrow^{n\,\tau\!\rightarrow} Q'$ かつ $(P', Q') \in \dot{\sim}_t$ を満たす Q' が存在する．補題 1 により $P \longrightarrow P'$ かつ $Q \longrightarrow Q'$

である．よって $(P, Q) \in \overset{*}{\sim}_t$ であるので $\dot{\sim}_t \subset \overset{*}{\sim}_t$ ．　□

第4章　リアルタイムオブジェクト指向言語とπ計算

本章ではπ計算を用いてリアルタイムオブジェクト指向言語をどのように表現できるかを1つの具体的な言語を用いて示す．簡単なリアルタイムオブジェクト指向言語 OOL_{RT} を定義し，OOL_{RT} の各構文要素に対しπ計算による記述を与える．

4.1　リアルタイムオブジェクト指向言語 OOL_{RT}

OOL_{RT} は簡単なオブジェクト指向言語 OOL[3] から型の取り扱いを省き，時間待ちとタイムアウトの構文を追加した言語である．OOL_{RT} ではメンバ変数への直接的なアクセスは不可能でカプセル化を実現している．

OOL_{RT} の構文を図 4.1 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 は文，X は変数名を表す．

構文要素の意味を以下に述べる．

$Pdec$：プログラム全体．各クラスの定義とプログラム開始時に実行される式からなる

$Cdec$：クラスの定義．メンバ変数の宣言とメソッドの定義からなる

$Mdec$：メソッドの定義．X_i が引数，S がメソッド本体となる

$Vdecs$：変数宣言

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

図 4.1: OOL_{RT} の構文

wait：時間待ち．E が待ち時間の長さを示す

within〜timeout：時間制約のある処理を記述する．主処理 S_1 が E で指定された時間内に終了しなければ S_2 に処理を移す

true, false：それぞれ標準で組み込まれる Boolean クラスのインスタンスで，true が真，false が偽を表す

$Numeral$：数値．10 進整数を扱うことができる

new：クラス A のインスタンスを生成

$E!M(E_1, \ldots, E_n)$：メソッド呼出し．あるクラスのインスタンス E のメソッド M を呼び出す

4.2　π 計算によるリアルタイムオブジェクト指向言語 OOL_{RT} の記述

図 4.1 の構文要素それぞれに対し π 計算による記述を与える．時間に関係する要素もあるため時間拡張した π 計算を用いる．構文要素 A に対応する記述を $[\![A]\!]$ と表す．

プログラム $Pdec$ の π 計算記述を図 4.2 に示す．$\widetilde{k}$ はプログラム中で定義される各クラスの名前を表す．Boolean はブール値を表すクラスでありプログラムには標準で組み込まれる．Boolean クラスの名前は k_{Bool} で $k_{Bool} \in \widetilde{k}$ である．

クラス定義 $Cdec$ の π 計算記述を図 4.2 に示す．名前 a がクラスのインスタンスを表す．名前 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し，新しい名前 a を返す．$\boldsymbol{\nu}$ 演算子によりインスタンスを表す名前

はすべて異なる．インスタンス生成要求の回数は不定なので ! 演算子が必要である．$\widetilde{x}$ はメンバ変数の名前，$\widetilde{m}$ はメソッドの名前，$\widetilde{c}$ は各メンバ変数の初期値を表す．メンバ変数にはインスタンス自身を示す $this$ が含まれ初期値は a である．

変数宣言 $Vdecs$ の π 計算記述を図 4.2 に示す．$\widetilde{x}$ が各変数の名前，$\widetilde{c}$ が各変数の初期値を表す．変数にアクセスするためには変数を表す名前 x を用いて 2 つの名前 r, u を渡しそのいずれかを用いる．r が値の参照，u が値の更新を意味する．

メソッド定義 $Mdecs$ 及び $Mdec$ の π 計算記述を図 4.3 に示す．すべてのメソッドは必ずいずれかのインスタンスに属している．そのインスタンスを名前 a によって表す．a を通して要求があるとそのインスタンスに属すメソッドの名前をすべて返し，返したメソッドの名前を通してメソッド呼び出しが発生する．$\widetilde{y}$ は実引数を表す名前，$\widetilde{r}, \widetilde{u}$ は実引数にアクセスするための名前である．

文 文要素の π 計算記述を図 4.4 に示す．l は文の実行完了を示す名前，r は return 文による値の返却を示す名前，$\widetilde{x}$ はメンバ変数，$\widetilde{k}$ はプログラム中で定義されているクラスの名前を表す．

式 式要素の π 計算記述を図 4.5 に示す．名前 $l, \widetilde{x}, \widetilde{k}$ が指すものは文要素と同じである．$[\![\mathsf{new}(A)]\!]$ の k_A はクラス A を示す名前，$[\![X]\!]$ の x_X は変数 X を示す名前である．メソッド呼び出しでは呼び出し先のインスタンスと引数の評価とインスタンスが持っているメソッドの名前の要求をし，その中の目的のメソッドを示す名前に対し引数を送る．r は return 文によって値を返すための名前，l' は値を返さないメソッドの完了を通知するための名前を表す．

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

図 4.2: プログラム，クラス定義，変数宣言の π 計算記述

$$
\begin{aligned}
[\![Mdecs]\!](\widetilde{m}, a, \widetilde{x}, \widetilde{k}) &\equiv !\,a(n).\overline{n}\langle\widetilde{m}\rangle.([\![Mdec_1]\!](m_1, \widetilde{x}, \widetilde{k}) \mid \cdots \\
&\qquad\qquad \mid [\![Mdec_m]\!](m_n, \widetilde{x}, \widetilde{k})) \\
[\![Mdec]\!](m, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu}g, \widetilde{y}, \widetilde{r}, \widetilde{u}) \\
&\qquad (m(\widetilde{v}, r, l).\overline{y_1}\langle r_1, u_1\rangle.\overline{u_1}\langle v_1\rangle.\cdots.\overline{y_l}\langle r_l, u_l\rangle.\overline{u_l}\langle c_l\rangle.\overline{g}\langle r, l\rangle \\
&\qquad \mid [\![\mathsf{var}\ Y_1]\!](y_1, \mathsf{nil}) \mid \cdots \mid [\![\mathsf{var}\ Y_l]\!](y_l, \mathsf{nil}) \\
&\qquad \mid g(r, l).[\![S]\!](l, r, \widetilde{x} \cup \widetilde{y}, \widetilde{k}))
\end{aligned}
$$

図 4.3: メソッド定義の π 計算記述

Boolean **クラス**　Boolean クラスの π 計算記述 [14] を図 4.6 に示す．k が Boolean クラスを表す名前，b_t が真，b_f が偽を表す名前である．BoolClass は Boolean クラスにアクセスするためのインターフェースであり，名前 k を通してアクセスする．

$$
\begin{aligned}
[\![X := E]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l')([\![E]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)(l'(v).\overline{x_X}\langle r, u\rangle.\overline{u}\langle v\rangle.\overline{l})) \\
[\![E]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l')([\![E]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\overline{l}) \\
[\![S_1; S_2]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l')([\![S_1]\!](l', r, \widetilde{x}, \widetilde{k}) \mid l'.[\![S_2]\!](l, r, \widetilde{x}, \widetilde{k})) \\
[\![\mathsf{wait}\ E]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l')([\![E]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(n).t[n].\overline{l}) \\
[\![\mathsf{return}\ E]\!](l, r, \widetilde{x}, \widetilde{k}) &\equiv (\boldsymbol{\nu} l')([\![E]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\overline{r}\langle v\rangle)
\end{aligned}
$$

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

図 4.4: 文要素の π 計算記述

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

図 4.5: 式要素の π 計算記述

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

図 4.6: Boolean クラスの π 計算記述

第5章　π計算を用いた開発例

本章では形式的記述を用いる開発がどのように行われるか一例を示す．簡単化したペースメーカーを対象として，開発に伴って生成される成果物を示しながら説明する．

5.1　問題と仕様

ペースメーカーは心拍数が低すぎるか拍動が停止した時に心臓の機能を補助する体内埋め込み型の装置である．ペースメーカーには心臓の拍動を検知するするとペーシングを抑制する抑制モードと，ペーシングを開始する同期モードがある．監視対象とペーシング対象として心房，心室のいずれかかあるいは両方があるがここでは特に決めない．ペースメーカーはほとんどの時間，拍動の検知を行っている．抑制モードでは拍動が検知できない時ペーシングを行い電気的刺激を発生する．その後の一定時間，ペースメーカーは不応状態に入り拍動の検知を行わない．

本章では抑制モードのペースメーカーを対象として，直前の拍動の検知から一定時間n以内に次の拍動が検知されない場合に電気的刺激を発生させるという動作の実装を取り上げる．

5.2　開発の流れ

開発は図 2.3 の流れに沿って行われる．

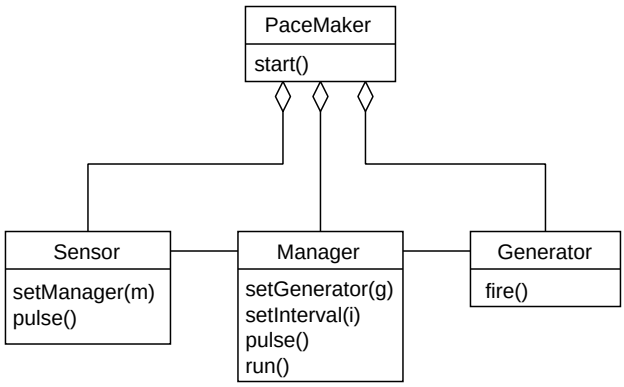

図 5.1: ペースメーカーのクラス図

5.2.1　分析・設計

分析・設計段階では，システムの仕様からシステムの静的構造を表すクラス図と動的振舞いを表す π 計算による式を得る．

システムのクラス図を図 5.1 に示す．ペースメーカーは $PaceMader$ クラスのインスタンスであり，$PaceMaker$ はセンサを表す $Sensor$ クラス，電気的刺激の発生を行う $Generator$ 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 $Manager$ クラスからなるとする．センサや刺激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，ここではハードウェアについては考えない．それらを抽象化したクラスを使いソフトウェアについてのみ扱う．

動作を表す π 計算による記述を図 5.2 に示す．$Sensor$ は心臓の動作を監視し，拍動があれば $Manager$ に通知する．拍動は名前 $pulse$ で表し，$Manager$ への通知は m_s を用いて行う．$Manager$ は時間を計測しながら $Sensor$ からの通知を待つ．$Sensor$ からの通知を受けると拍動があったと判断し時間をリセットする．一定時間 n の間 $Sensor$ からの通知がなければタイムアウトが発生し，拍動がなかったと判断して電気的刺激の発生を m_g により指示する．その後一定時間 m だけ時間待ちを行う．これはペーシング後の不応状態を表す．

$$
\begin{aligned}
PaceMaker &= Manager \mid Sensor \mid Generator \\
Manager &= m_s.Manager + t[n].\overline{m_g}.t[m].Manager \\
Sensor &= !\,pulse.\overline{m_s} \\
Generator &= !\,m_g.\overline{fire}
\end{aligned}
$$

図 5.2: ペースメーカーの動作記述

$Generator$ は $Manager$ からの指示を受けると電気的刺激の発生を行う．電気的刺激の発生は名前 $fire$ によって表す．設計段階では図 5.2 のような式を書く．その際に補助として図 5.3 のように従来手法による開発で用いるシーケンス図などを用いることでもできる．

実行の一例を以下に示す．ここでは $n = 2, m = 1$ とした．また $fire$ が常に可能なプロセスが並行に動作しているため，$\overline{fire}$ が時間経過に優先して実行可能とする．

$$
\begin{aligned}
PaceMaker &\xrightarrow{pulse} Manager \mid \overline{m_s} \mid Sensor \mid Generator \\
&\xrightarrow{\tau} Manager \mid Sensor \mid Generator \\
&\rightsquigarrow m_s.Manager + t[1].\overline{m_g}.t[1].Manager \\
&\quad \mid Sensor \mid Generator \\
&\rightsquigarrow m_s.Manager + t[0].\overline{m_g}.t[1].Manager \\
&\quad \mid Sensor \mid Generator \\
&\xrightarrow{\tau} \overline{m_g}.t[1].Manager \mid Sensor \mid Generator \\
&\xrightarrow{\tau} t[1].Manager \mid Sensor \mid \overline{fire} \mid Generator \\
&\xrightarrow{\overline{fire}} t[1].Manager \mid Sensor \mid Generator \\
&\rightsquigarrow Manager \mid Sensor \mid Generator \\
&\xrightarrow{pulse} \cdots\cdots
\end{aligned}
$$

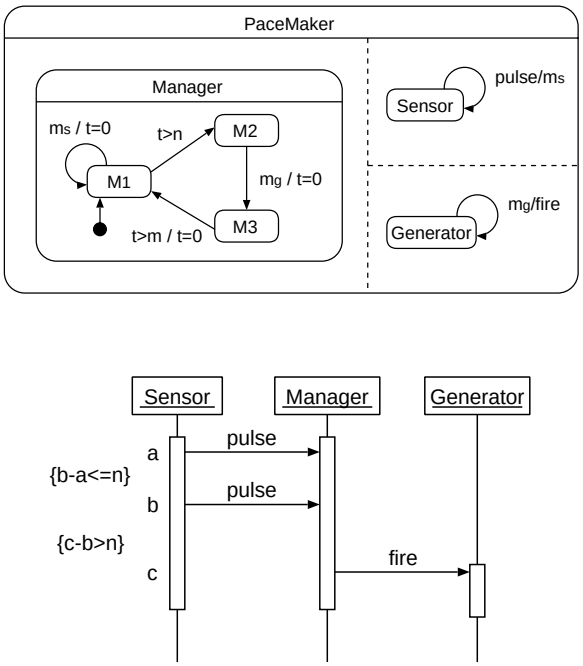

図 5.3: ステートチャートとシーケンス図による動作記述

$n = 2$ なので直前の拍動から 2 単位時間以内に次の拍動がなければ電気的刺激を発生させる．この例では初めに拍動があった後，2 単位時間経過したため電気的刺激を発生させたことを示している．このように π 計算による動作記述が仕様を満たし正しく動作するか検証する．

5.2.2　実装

分析・設計段階で作成したクラス図 (図 5.1) と動作記述 (図 5.2) に基づき実装を行う．実装にはリアルタイムオブジェクト指向言語 OOL_{RT} を用いる．実装で作成されるプログラムの一例を図 5.4 に示す．中心となるのは $Manager$ クラスにある run メソッドの中の `while` 文である．拍動に対する時間待ちとタイムアウトを `within` 文を使って実装した．時間待ちには `while` 文を使った．

拍動の通知を意味する pulse メソッドの呼び出しによりループから抜ける．電気的刺激発生後の不応状態は wait 文により実装した．

5.2.3　π 計算記述への変換

図 5.4 のプログラムを図 4.2，4.3，4.4，4.5 の記述に従って π 計算による表現に変換する (付録 A)．

拍動を待機している時の動作例を以下に示す．ここでは動作に関連するプロセスのみ示している．

$$
\begin{aligned}
& [\![S_1]\!](l', r', \widetilde{x}, \widetilde{k}) \\
& \mid ((r'(v).\overline{r}\langle v\rangle \mid l'.\overline{l}) + t[1000].[\![gen!fire(); \mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k})) \\
& \mid (\boldsymbol{\nu} g)(m_{pulse}(\mathsf{nil}, r, l).\overline{g}\langle r, l\rangle \mid g(r, l).[\![manager!pulse()]\!](l, r, \widetilde{x}, \widetilde{k})) \\
& \mid \cdots\cdots \\
\rightsquigarrow^{500} & \\
\xrightarrow{m_{pulse}} & \\
\xrightarrow{\tau}{}^{*} \quad & [\![S_1]\!](l', r', \widetilde{x}, \widetilde{k}) \\
& \mid ((r'(v).\overline{r}\langle v\rangle \mid l'.\overline{l}) + t[1000].[\![gen!fire(); \mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k})) \\
& \mid (\boldsymbol{\nu} g)(m_{pulse}(\mathsf{nil}, r, l).\overline{g}\langle r, l\rangle \mid g(r, l).[\![manager!pulse()]\!](l, r, \widetilde{x}, \widetilde{k})) \\
& \mid \cdots\cdots \\
\rightsquigarrow^{1000} \quad & t[0].[\![gen!fire(); \mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k}) \\
& \mid (\boldsymbol{\nu} g)(m_{pulse}(\mathsf{nil}, r, l).\overline{g}\langle r, l\rangle \mid g(r, l).[\![manager!pulse()]\!](l, r, \widetilde{x}, \widetilde{k})) \\
& \mid \cdots\cdots \\
\xrightarrow{\tau}{}^{*} & \\
\xrightarrow{m_{fire}} &
\end{aligned}
$$

```
 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:
16: class Manager is
17:   var gen, b, n
18:
19:   method setGenerator(g) is
20:     gen := g
21:
22:   method setInterval(i) is
23:     n := i
24:
25:   method pulse() is
26:     b := false
27:
28:   method run() is
29:     while true do
30:       within n do
31:         while b do nil done;
32:         b := true
33:       timeout
34:         gen!fire();
35:         wait 250
36:       done
37:     done
38:
39:
40: class Sensor is
41:   var manager
42:
43:   method setManager(m) is
44:     manager := m
45:
46:   method pulse() is
47:     manager!pulse()
48:
49:
50: class Generator is
51:   method fire() is
52:     // 電気刺激発生
53:     nil
54:
55:
56: with
57:   new(PaceMaker)!start()
```

図 5.4: ペースメーカーの実装

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

初めは 500 単位時間経過した時に拍動があり m_{pulse} アクション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 m_{fire} イベント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仕様を満たしている．

第6章　おわりに

6.1　まとめ

本論文では実時間システムの開発への適用のためにπ計算を時間に関して拡張した．π計算に時間経過を示すアクションと時間経過による遷移規則を導入することにより，時間が意味を持つ実時間システムの動作の記述を可能にした．実時間システムの振舞いを形式的に記述することで振舞いを厳密に表現できる．実時間システムの開発においては設計段階で振舞いの形式的記述を作成して動作検証を行う．

本論文では時間を離散時間としてモデル化した．時間経過遷移を導入して時間経過による振舞いの変化を定式化した．単位時間ごとの動作を解析することが可能である．2つのプロセスが時間経過による遷移が発生しても同等の振舞いをするならば，それらのプロセスの振舞いは時間の経過を無視しても同等であることを示した．このことから本論文の時間拡張により，プロセスの振舞いを従来よりもさらに詳細に表現することが可能になったといえる．

リアルタイムオブジェクト指向言語に対し時間拡張したπ計算による記述を与えた．この記述を用いてシステムの実装時に作成されるソフトウェアのプログラムを形式的表現に変換する．設計段階での動作記述と比較して動作の等価性を確認することで，設計に従った実装がなされたか確認できる．本論文ではOOL_{RT}に対する記述を与えたが，この記述はOOL_{RT}に本質的に依存しているわけではない．クラス定義，インスタンス生成，メソッド呼び出し，制御

構造など同様の実装能力を持つオブジェクト指向言語に対する一般的な記述となっている．

関連研究

Berger らはメッセージ消失，タイマー，プロセスの実行失敗などを導入して π 計算を拡張している [15]．タイマーは $\mathsf{timer}^t(Q, R)$ のようにプロセスとして実現されている．時間の経過は時間ステップ関数 ϕ を適用することにより発生する．動作は時間ステップ関数と動作意味定義によって定義される．離散時間による時間経過を表現することが可能である．Berger らはこの拡張した π 計算を用いて二相コミットプロトコルを記述している．

堀田らは π 計算に基づくプログラミング言語 Nepi[16] を提案している．Nepi は Allegro Common Lisp 上で実装されている．通信，非決定的選択，プロセスの並行動作をプリミティブとして記述できる構文を持つ．Nepi を用いることで π 計算のプロセス式を直接的に記述し実行することができる．Nepi は独自拡張として時間待ちを記述するためのプリミティブを備えているため，OOL_{RT} から変換して得られた π 計算による式の一つの実行系となっている．

6.2　今後の課題

π 計算を時間に関して拡張することで実時間システムの動作を形式的に記述できるようになった．形式的記述を用いてシステムが正しく動作するか機械的に検証できるが，具体的な検証手法については今後の課題である．検証したい性質を形式的に記述する方法が必要である．性質とシステムの振舞い記述からシミュレーションあるいは数学的操作などにより検証する方法を検討しなければならない．

π 計算に対する時間拡張では構文と動作意味に変更を加えたため従来の等価

性理論をそのまま適用することはできない．双模倣性や合同を定義して等価性について調べる必要がある．等価性に基づく検証方法も考えられる．

形式的記述を用いる実時間システムの開発ではシステムの動作を π 計算で記述する．しかしシーケンス図などの図的表現に比べ直感的にわかりにくい．システム規模が大きくなると多くの式が必要になる．実際の開発に適用するための環境を整えることも今後の課題である．

謝辞

本研究を進めるにあたり熱心にご指導頂いた名古屋大学大学院工学研究科情報工学専攻 阿草清滋教授，結縁祥治助教授に感謝致します．また，熱心に議論して頂いた愛知県立大学 山本晋一郎助教授，名古屋大学 濱口毅助手及び阿草研究室の皆さんに感謝致します．

参考文献

[1] Object Management Group. *OMG Unified Modeling Language Specification Version 1.4*, September 2001.

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

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

[4] E.M.Clarke, Orna Grunberg, and Doron Peled. *Model Checking*. The MIT Press, 1999.

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

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

[7] 井原廣一. リアルタイムシステムとは. 情報処理, 35(1):12–17, 1994.

[8] J. ランボー, M. プラハ, W. プメラニ, F. エディ, and W. ローレンセン. オブジェクト指向方法論 *OMT* モデル化と設計. トッパン, 1992.

[9] I. ヤコブソン, M. クリスターソン, P. ジョンソン, and G. ウーバーガード. オブジェクト指向ソフトウェア工学 *OOSE use-case* によるアプローチ. トッパン, 1995.

[10] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I/II. *Information and Computation*, 100:1–77, 1992.

[11] Robin Milner. *Communication and Concurrency*. Cambridge University Press, 1988.

[12] C.Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

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

[14] David Walker. Objects in the π-calculus. *Information and Computation*, 116(2):253–271, 1995.

[15] M.Berger and K.Honda. The two-phase commitment protocol in an extended π-calculus. In *Preliminary Proceedings of EXPRESS '00*, pages 105–130, 2000.

[16] E.Horita and K.Mano. Nepi: a network programming language based on the π-calculus. In *Coodinaion96*, volume 1061 of *LNCS*, pages 424–427. Springer, 1996.

付 録A　ペースメーカーのプログラムに対するπ計算記述

図5.4のプログラムをπ計算での表現に変換したものを以下に示す．Pがプログラム全体を表すπ計算の項である．名前k_Pによるインタラクションから実行が開始する．

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

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

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

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

$$
\begin{aligned}
& [\![m := \mathsf{new}(Manager)]\!](l, r, \widetilde{x}, \widetilde{k}) \\
& \quad = \; (\boldsymbol{\nu} l')([\![\mathsf{new}(Manager)]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)(l'(v).\overline{x_m}\langle r, u\rangle.\overline{u}\langle v\rangle.\overline{l}))
\end{aligned}
$$

$$
[\![\mathsf{new}(Manager)]\!](l, \widetilde{x}, \widetilde{k}) = (\boldsymbol{\nu} l')(\overline{k_M}\langle l'\rangle.l'(c).\overline{l}\langle c\rangle)
$$

$$
\begin{aligned}
& [\![s := \mathsf{new}(Sensor)]\!](l, r, \widetilde{x}, \widetilde{k}) \\
& \quad = \; (\boldsymbol{\nu} l')([\![\mathsf{new}(Sensor)]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)(l'(v).\overline{x_s}\langle r, u\rangle.\overline{u}\langle v\rangle.\overline{l}))
\end{aligned}
$$

$$[\![\mathsf{new}(Sensor)]\!](l,\widetilde{x},\widetilde{k}) = (\boldsymbol{\nu} l')(\overline{k_S}\langle l'\rangle.l'(c).\overline{l}\langle c\rangle)$$

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

$$[\![\mathsf{new}(Generator)]\!](l,\widetilde{x},\widetilde{k}) = (\boldsymbol{\nu} l')(\overline{k_G}\langle l'\rangle.l'(c).\overline{l}\langle c\rangle)$$

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

$$[\![s!setManager(m)]\!](l,\widetilde{x},\widetilde{k})$$
$$= \quad (\boldsymbol{\nu} h,h_1,g_1)([\![s]\!](h,\widetilde{x},\widetilde{k}) \mid g_1.[\![m]\!](h_1,\widetilde{x},\widetilde{k})$$
$$\quad \mid h(v).\overline{g_1}.h_1(v_1).(\boldsymbol{\nu} r,l',n)(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{setM}}\langle v_1,r,l'\rangle \mid r(v).\overline{l}\langle v\rangle \mid l'.\overline{l}))$$

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

$$[\![m!setGenerator(g)]\!](l,\widetilde{x},\widetilde{k})$$
$$= \quad (\boldsymbol{\nu} h,h_1,g_1)([\![m]\!](h,\widetilde{x},\widetilde{k}) \mid g_1.[\![g]\!](h_1,\widetilde{x},\widetilde{k})$$
$$\quad \mid h(v).\overline{g_1}.h_1(v_1).(\boldsymbol{\nu} r,l',n)(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{setG}}\langle v_1,r,l'\rangle \mid r(v).\overline{l}\langle v\rangle \mid l'.\overline{l}))$$

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

$$[\![m!setInterval(1000)]\!](l,\widetilde{x},\widetilde{k})$$
$$= \quad (\boldsymbol{\nu} h,h_1,g_1)([\![m]\!](h,\widetilde{x},\widetilde{k}) \mid g_1.[\![1000]\!](h_1,\widetilde{x},\widetilde{k})$$
$$\quad \mid h(v).\overline{g_1}.h_1(v_1).(\boldsymbol{\nu} r,l',n)(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{setI}}\langle v_1,r,l'\rangle \mid r(v).\overline{l}\langle v\rangle \mid l'.\overline{l}))$$

$$[\![m]\!](l,\widetilde{x},\widetilde{k}) = (\boldsymbol{\nu} r,u)(\overline{x_m}\langle r,u\rangle.r(v).\overline{l}\langle v\rangle)$$

$$[\![s]\!](l,\widetilde{x},\widetilde{k}) = (\boldsymbol{\nu} r,u)(\overline{x_s}\langle r,u\rangle.r(v).\overline{l}\langle v\rangle)$$

$$[\![g]\!](l,\widetilde{x},\widetilde{k}) = (\boldsymbol{\nu} r,u)(\overline{x_g}\langle r,u\rangle.r(v).\overline{l}\langle v\rangle)$$

$$[\![1000]\!](l,\widetilde{x},\widetilde{k}) = \overline{l}\langle 1000\rangle$$

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

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

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

$$[\![g]\!](l,\widetilde{x}\cup\widetilde{y},\widetilde{k}) = (\boldsymbol{\nu} r,u)(\overline{y_g}\langle r,u\rangle.r(v).\overline{l}\langle v\rangle)$$

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

$$[\![n := i]\!](l, r, \widetilde{x} \cup \widetilde{y}, \widetilde{k}) = (\boldsymbol{\nu} l')([\![i]\!](l', \widetilde{x} \cup \widetilde{y}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)(l'(v).\overline{x_n}\langle r, u\rangle.\overline{u}\langle v\rangle.\overline{l}))$$

$$[\![i]\!](l, \widetilde{x} \cup \widetilde{y}, \widetilde{k}) = (\boldsymbol{\nu} r, u)(\overline{y_i}\langle r, u\rangle.r(v).\overline{l}\langle v\rangle)$$

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

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

$$[\![b := \mathsf{false}]\!](l, r, \widetilde{x}, \widetilde{k}) = (\boldsymbol{\nu} l')([\![\mathsf{false}]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)(l'(v).\overline{x_b}\langle r, u\rangle.\overline{u}\langle v\rangle.\overline{l}))$$

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

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

$$[\![Body_{run}]\!](l, r, \widetilde{x}, \widetilde{k})$$

$$= \quad (\boldsymbol{\nu} g, l', l_1, l_2)(\overline{g} \mid g.([\![\mathsf{true}]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} t, f)(l'(v).\overline{v}\langle \widetilde{n}\rangle.\overline{n_1}\langle t, f\rangle.(t.\overline{l_1} \mid f.\overline{l_2}))$$
$$\mid l_1.[\![within]\!](l, r, \widetilde{x}, \widetilde{k}).\overline{g} \mid l_2.\overline{l}))$$

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

$$= \quad (\boldsymbol{\nu} l', l'', r')([\![n]\!](l'', \widetilde{x}, \widetilde{k}) \mid l''(n).([\![S_1]\!](l', r', \widetilde{x}, \widetilde{k})$$
$$\mid ((r'(v).\overline{r}\langle v\rangle \mid l'.\overline{l}) + t[n].[\![gen!fire(); \mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k}))))$$

$$[\![n]\!](l, \widetilde{x}, \widetilde{k}) = (\boldsymbol{\nu} r, u)(\overline{x_n}\langle r, u\rangle.r(v).\overline{l}\langle v\rangle)$$

$$[\![S_1]\!](l, r, \widetilde{x}, \widetilde{k})$$

$$= \quad (\boldsymbol{\nu} l')([\![\mathsf{while}\ b\ \mathsf{do\ nil\ done}]\!](l', r, \widetilde{x}, \widetilde{k}) \mid l'.[\![b := \mathsf{true}]\!](l, r, \widetilde{x}, \widetilde{k}))$$

$$[\![\mathsf{while}\ b\ \mathsf{do\ nil\ done}]\!](l, r, \widetilde{x}, \widetilde{k})$$

$$= \quad (\boldsymbol{\nu} g, l', l_1, l_2)(\overline{g} \mid g.([\![b]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} t, f)(l'(v).\overline{v}\langle \widetilde{n}\rangle.\overline{n_1}\langle t, f\rangle.(t.\overline{l_1} \mid f.\overline{l_2}))$$
$$\mid l_1.[\![\mathsf{nil}]\!](g, r, \widetilde{x}, \widetilde{k}) \mid l_2.\overline{l}))$$

$$[\![b]\!](l, \widetilde{x}, \widetilde{k}) = (\boldsymbol{\nu} r, u)(\overline{x_b}\langle r, u\rangle.r(v).\overline{l}\langle v\rangle)$$

$$[\![b := \mathsf{true}]\!](l, r, \widetilde{x}, \widetilde{k}) = (\boldsymbol{\nu} l')([\![\mathsf{true}]\!](l', \widetilde{x}, \widetilde{k}) \mid (\boldsymbol{\nu} r, u)(l'(v).\overline{x_b}\langle r, u\rangle.\overline{u}\langle v\rangle.\overline{l}))$$

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

$$= \quad (\boldsymbol{\nu} l')([\![gen!fire()]\!](l', r, \widetilde{x}, \widetilde{k}) \mid l'.[\![\mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k}))$$

$$[\![gen!fire()]\!](l, r, \widetilde{x}, \widetilde{k}) = (\boldsymbol{\nu} l')([\![gen!fire()]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(v).\overline{l})$$

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

$$= \quad (\boldsymbol{\nu} h)([\![gen]\!](h, \widetilde{x}, \widetilde{k}) \mid h(v).(\boldsymbol{\nu} r, l', n)(\overline{v}\langle n\rangle.n(\widetilde{m}).\overline{m_{fire}}\langle r, l'\rangle \mid r(v).\overline{l}\langle v\rangle \mid l'.\overline{l}))$$

$$[\![\mathsf{wait}\ 250]\!](l, r, \widetilde{x}, \widetilde{k}) = (\boldsymbol{\nu} l')([\![250]\!](l', \widetilde{x}, \widetilde{k}) \mid l'(n).t[n].\overline{l})$$

$$[\![250]\!](l, \widetilde{x}, \widetilde{k}) = \overline{l}\langle 250\rangle$$

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

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

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

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

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

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

$$[\![m]\!](l, \widetilde{x} \cup \widetilde{y}, \widetilde{k}) = (\boldsymbol{\nu} r, u)(\overline{y_m}\langle r, u\rangle.r(v).\overline{l}\langle v\rangle)$$

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

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

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

$$[\![manager]\!](l, \widetilde{x}, \widetilde{k}) = (\boldsymbol{\nu} r, u)(\overline{x_{man}}\langle r, u\rangle.r(v).\overline{l}\langle v\rangle)$$

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

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

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

$$[\![\mathsf{new}(PaceMaker)]\!](l, \widetilde{x}, \widetilde{k}) = (\boldsymbol{\nu} l')(\overline{k_P}\langle l'\rangle.l'(c).\overline{l}\langle c\rangle)$$