# 同時性を考慮した並行システムの振舞い検証支援に関する研究

M2013SE003 石原脩平
指導教員：沢田篤史

## 1　はじめに

マルチコアプロセッサの出現といった並行システムの普及に伴い，並行システムの検証の重要性が注目されている．しかし，並行システムのソフトウェア開発では，並行に動作する複数のプロセスが生起する事象による干渉が発生する可能性がある．もし，干渉を考慮せずにシステムを設計した場合，事象の生起順に起因するデッドロックやライブロックと言った不具合が発生する可能性がある．ソフトウェア検証手法の一つであるモデル検査では，干渉といった不具合を検証することが可能である．しかし，システムは複数事象を逐次プロセスの組み合わせによって把握するので，全ての実行系列の記述が複雑になり，システム全体の振舞いを把握することは困難となる．

本研究の目的は，並行システムの振舞い検証支援のための，同時事象を考慮した振舞い仕様記述法の提案である．本研究における同時事象とは，ある区間で複数の事象が生起することを示す．同時事象を考慮した振舞い記述法を提案することで，同時事象に関する検証を簡便にする．また，検証式の再利用性向上のために，同時事象の定義を用いた振舞い検証の枠組みを提案する．

本研究の基本的なアイデアは，並行システムの振舞いを区間の合成として定義し，区間を導入して，同時性を定義することである．同時性を定義することによって，区間に同時性を局所化する．また，環境に振舞い仕様を局所化することで検証式の再利用性向上を目指す．

本稿では，プロセス代数 CSP[1] を用いて並行システムの同時性を定義して，同時性を考慮した検証の枠組みについて説明する．また，自動販売機システムを例題に同時性の検証を行った．最後に，振舞い仕様記述法の有効性と振舞い検証の再利用性について議論する．提案した手法を用いることでアーキテクチャ段階における効率的な振舞い検証が，アーキテクチャ設計の信頼性向上に寄与することが期待できる．

## 2　基本的なアイデア

本研究で対象とする並行に動作するプロセスが複数事象発生による干渉と，同時性を考慮した振舞い検証の基本的なアイデアについて述べる．

### 2.1　並行システムにおける同時性

2 つのセンサをソフトウェアで制御する例を用いて，事象の同時性について説明する．図 1 はシステムが 2 つのセンサを制御するという構造である．ハードウェアセンサ 1，2 が外部から同時にイベントを検知したとする．それによって，センサ 1，2 はシステム制御に対して同時にイベントを通知する可能性がある．

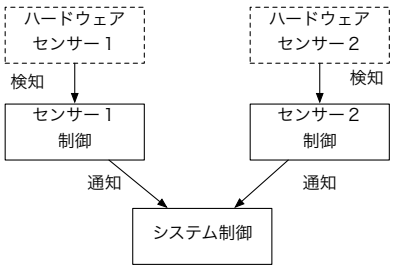


図 1　2 つのセンサ制御

### 2.2　ソフトウェアによる複数事象の制御

ソフトウェアによる複数事象の制御について説明する．センサからシステム制御にイベントが通知する時は通知は逐次プロセスで検知される．また，片方のセンサから通知があれば，もう片方には通知が送信されないように同期する必要がある．しかし，同時性を考慮した設計は困難である．設計の困難さとして，通知順序の逆転と際どい事象発生順序がある．

#### 2.2.1　通知順序の逆転

図 1 を用いて通知順序の逆転を説明する．ハードウェアセンサ 1 がイベント検知した後にハードウェアセンサ 2 がイベントを検知したとする．この場合，システム制御へはイベントは 1，2 の順番であることが想定されるが，センサ 1，2 を介することでシステム制御へのイベント通知が 2，1 となってしまう可能性がある．順序の逆転が起こらないことを仕様で記述する場合，同時の概念を捉える必要がある．

#### 2.2.2　際どい事象発生順序

図 2 に際どい事象発生順序の例を示す．センサがコントローラから通知開始イベントを受信し，環境に検知開始イベントを送信する．その後，センサがコントローラから通知終了イベントを受信し，その応答イベントを返信する．しかし，応答イベントよりも先に環境から検知イベントが送信される場合がある．その結果，際どい事象が発生し，センサはイベント検知ができないにも関わらず，検知イベントが送信され，取りこぼしてしまう．上記のような予期せぬ順序は検証によって検出される．

### 2.3　同時事象の定義

#### 2.3.1　区間

区間の概念を導入して，開始終了付き事象と区間の定義を説明する．

- 開始終了付き事象：START * END * EVENT
  開始終了付き事象は，区間で起こり得る事象である．事象を構成する要素は 3 項組で定義し，それぞれ

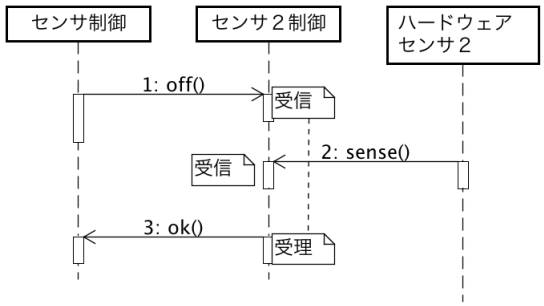


図 2 際どい事象発生順序の例

START は開始，END は終了，EVENT は事象を指す．事象内では，開始と終了の間に事象が有るか無いかを定義している（図 4）．

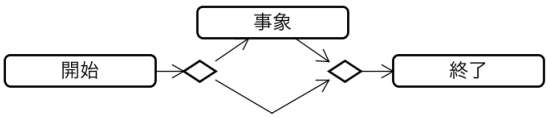


図 3 事象の概念

- 区間：set of EVENT_WITH_START_AND_END
  区間は開始終了付き事象の集合で定義する．

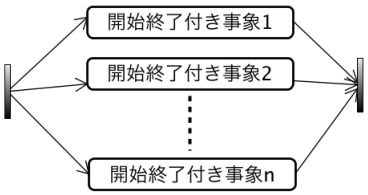


図 4 区間の概念

#### 2.3.2 区間事象発生の定義

- 単一事象選択区間
  単一事象選択区間は区間中で任意の単一の事象が生起すること
- 複数事象発生区間
  複数事象発生区間は区間中で任意の複数の事象が生起すること

#### 2.3.3 区間の合成

システムの振舞いを区間の合成として定義する．区間の合成には，逐次，選択，再帰の 3 つがある．

### 2.4 振舞い検証

#### 2.4.1 検証の枠組み

本研究では，同時性をプロセス代数 CSP を用いて記述し，CSP の代表的なモデル検査器 FDR[2] を用いて検証する．FDR は「実現」が「振舞い仕様」を満たすこと（詳細化関係）を自動的に検証することが出来る．

振舞い仕様は外部イベントの振舞いとなる．実現は対象となるシステムと環境，制約から構成される．制約は入力区間振舞いで環境の事象を記述する．

## 3 同時性を含意した振舞い記述

2 章で述べた同時性をプロセス代数 CSP を用いて定義する．

### 3.1 区間

#### 3.1.1 事象

事象の定義を以下にする．

```
EVENT_WITH_START_AND_END(start, end, event)=
  start->
   (event-> end-> SKIP [] end-> SKIP)
```

である．EVENT_WITH_START_AND_END は開始イベントと終了イベントの間に入力イベントが有るか無いかを記述している．

#### 3.1.2 区間

区間の定義を以下にする．

```
SECTION(s) =
  ||| (start, end, event):s @
    EVENT_WITH_START_AND_END(start, end, event)
```

である．SECTION は EVENT_WITH_SECTION の集合の順不同な実行を記述している．

#### 3.1.3 単一事象選択区間

単一事象選択区間 SEL_SECTION を定義する．

```
SEL_SECTION(s) =
  [] event:OccurredEvents(s)
       @ event -> SKIP
```

である．SEL_SECTION は，区間中で OccurredEvents から単一の任意のイベントが実行することである．

- OccurredEvents：
  EVENT_WITH_START_AND_END
  内のイベント入力の集合

#### 3.1.4 複数事象生起

複数事象生起 PL_SECTION を定義する．

```
PL_SECTION(occurred, not_occurred) =
  (||| (st, end, ev):occurred
               @ st -> ev -> end -> SKIP)
   |||
  (||| (st, end, ev):not_occurred
               @ st -> end -> SKIP)
```

である．PL_SECTION は区間内にイベントがある集合とイベントが無い集合の順不同の実行を記述している．

- occurred：入力イベントが存在する区間の集合
- not_occurred：入力イベントが存在しない区間の集合

### 3.2 区間振舞い

区間振舞いの CSP 記述以下の通りに定義する．

#### 3.2.1 逐次

```
SEQ(s1, s2) = s1; s2 -> SKIP
```

#### 3.2.2 選択

```
SEL(ss) = [] s:ss @ s -> SKIP
```

#### 3.2.3 再帰

```
REC(P) = P; REC(P)
```

## 4 振舞い検証

### 4.1 検証式

基本的なアイデアで説明した振舞い検証の検証式について説明する．実現が振舞い仕様を満たすことは以下の通りに記述する．

```
assert SPEC [FD= SYSTEM || ENV
                        || CONSISTANT
```

ここでは，SPEC が振舞い仕様，SYSTEM が実現，ENV が環境，CONSISTANT が制約となる．

## 5 事例検証

本章では，事例検証として自動販売機システムを例題として取り上げて検証する．本研究では，同時性を検証するために振舞い検証を逐次振舞い検証と同時振舞い検証の段階に分けて検証する．自動販売機システムのクラス図は図??となる．自動販売機システムの仕様は，コインを投入し，ボタンを押すと商品が出るというものである．

### 5.1 逐次振舞いモデルの検証

本節では，自動販売機システムの逐次仕様振舞いモデルの検証について説明する．

#### 5.1.1 区間の定義

2.3.3 項で定義した区間振舞いを使用して，自動販売機システムの入力を記述する．

```
DEPOSIT = SECTION({COIN})
SELECT = SECTION({BUTTON, LEVER})

INPUT_PURCHASE = DEPOSIT; SELECT;
                 INPUT_PURCHASE
```

ここで DEPOSIT が入金区間，SELECT が選択区間を示している．

#### 5.1.2 逐次振舞いモデルの記述

逐次振舞いの場合の自動販売機システムの制御部分は図 5 となる．図 5 を 3.1.3 節の逐次区間振舞い仕様を用いて逐次仕様モデルを記述する．

```
SEQ = SEL_SECTION({COIN});
      SEL_SECTION({BUTTON, LEVER});
      SEQ
```

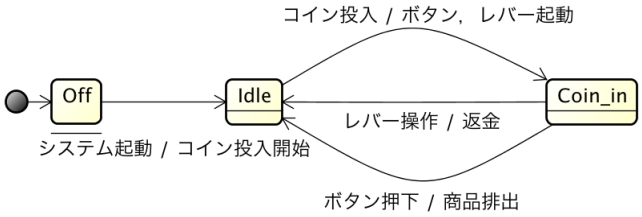


図 5 逐次振舞いモデルの状態遷移機械

SEQ は 2 つの逐次振舞いから構成している．SEL_SECTION({COIN}) は区間内で選択されるのは COIN のみ，SEL_SECTION({BUTTON, LEVER}) は選択される対象が BUTTON と LEVER であるのでどちらかが選択される．

### 5.2 環境の振舞い

自動販売機システムにおけるアーキテクチャから独立した環境の振舞い ENV を定義する．

```
ENV_(c, e) =
  env_ev.c.on ->
    (send.c.e -> env_ev.c.off ->
                 env_f.c.sent -> SKIP []
     env_ev.c.off -> env_f.c.nosent -> SKIP);
    ENV_(c, e)
```

である．ENV の第 1 引数 c はコンポーネント，第 2 引数 e は入力イベントとなる．

- env_ev.c.on：区間開始イベント
- env_ev.c.off：区間終了イベント
- send.c.e：入力イベント

この定義を自動販売機の事例に当てはめると以下の記述になる．

```
ENV_COIN   = ENV_(Coin, inserted)
ENV_BUTTON = ENV_(Button, pushed)
ENV_LEVER  = ENV_(Lever, pulled)

ENV_CTRL = ENV_COIN ||| ENV_BUTTON |||
           ENV_LEVER
```

事例の自動販売機システムにおける入力があるコンポーネントは，COIN, BUTTON, LEVER であるので環境の振舞いは 3 つとなる．それらを順不同に実行することで環境の取り得る振舞いを記述した．

#### 5.2.1 逐次振舞いモデルの検証結果

逐次振舞いの振舞い仕様は以下の通りに記述した．

```
SEQ_SPEC = EXT_INSERTED;
           (EXT_PUSHED; EXT_ITEM []
            EXT_PULLED; EXT_CHANGE);
           SEQ_SPEC
```

SEL_SECTION はコイン投入後，ボタン押下して商品排出，もしくはレバー操作してコイン返金という仕様である．この仕様は実現を満たすことが検証で確認できた．

```
assert SEQ_SPEC [FD=
       TARGET(SEQ_MODEL, ExternalEvents)
assert TARGET(SEQ_MODEL, ExternalEvents)
        [FD= SEQ_SPEC
```

### 5.3 同時振舞いモデルの検証

本説では，自動販売機の同時振舞いモデルの検証について説明する．

#### 5.3.1 区間の定義

区間が必要な箇所は振舞いモデルと同様であることから，逐次振舞いモデルと同様の記述を使用する．

#### 5.3.2 同時振舞いモデルの記述

同時振舞いモデルの場合の自動販売機システムの制御部分は図 6 となる．図 6 を 3.1.3 節の同時区間振舞い仕様

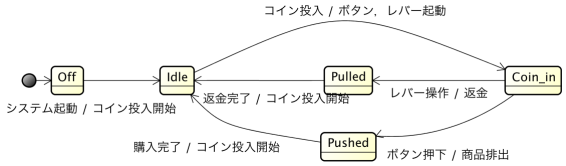


図 6 複数事象を考慮した状態遷移機械

を用いて同時仕様振舞いモデルを記述する．

```
PL = PL_SECTION({COIN},{});
     PL_SECTION({BUTTON, LEVER}, {}); PL
```

同時が起こる振舞いは，PL_SECTION({BUTTON,LEVER},{}) の BUTTON と LEVER とする．

#### 5.3.3 同時振舞いモデルの検証結果

同時振舞いの振舞い仕様は以下の通りに記述した．

```
PL_SPEC = EXT_INSERTED;
          ((EXT_PUSHED ||| EXT_PULLED);
           (EXT_ITEM |~| EXT_CHANGE));
          PL_SPEC
```

PL_SECTION はコイン投入，ボタン押下とレバー操作が有った後，商品排出，もしくはコイン返金という仕様である．この仕様は実現を満たすことが検証で確認できた．

```
assert PL_SPEC
       [FD= TARGET(PL_MODEL, ExternalEvents)
assert TARGET(PL_MODEL, ExternalEvents)
       [FD= PL_SPEC
```

## 6 考察

本章では，提案する記述法の有効性と振舞い検証の再利用性について考察する．

### 6.1 振舞い記述法の有効性

から，振舞い記述法を提案した．振舞い検証時，複数事象生起を検証するためには，環境への入力と複数事象発生を確認する制約が必要となる．しかし，入力と制約を一つにまとめて記述した場合，入力の振舞いに制約が散在する可能性がある．自動販売機システムを例にして，コイン 1，ボタン 3，レバー 1 の場合の入力の振舞いを区間事象生起を用いない場合の記述 (Input_ENV) は以下の通りになる．

```
Input_ENV =
   coin ->
   lever_on -> lamp_b1_on -> lamp_b2_on ->
   button2 -> button1 -> sync_b1 ->
   lamp_b3_on -> lever -> sync_b2 ->
   sync_ok -> sync_lever -> sync_ok ->
   button3 -> sync_b3 -> sync_ok ->
   Input_ENV
```

ここで，coin, button, lever が入力事象に相当し，lever_on, lamp_b1_on, lamp_b2_on, lamp_b3_on, sync_b1, sync_b2, sync_b3, sync_ok が制約記述に相当する．この記述では，3 個のボタンとレバーが複数事象を生起することを確認するために 10 個の制約が必要になる．制約の記述によって，入力の振舞いが複雑になる可能性がある．本研究で提案する振舞い記述法では，制約記述を関数を用いて定義することによって，容易に理解することを支援する．

### 6.2 振舞い検証の再利用性

振舞い検証の再利用性について説明する．本研究では，振舞い検証を逐次振舞い検証と同時振舞い検証の 2 つの段階に分けて検証した．その際，環境はアーキテクチャの設計から独立にしたことから，逐次振舞い検証で使用した環境が同時振舞い検証でも使用することができた．

本研究で提案した区間の概念は，例外処理の検証時に適応することが考えられる．

## 7 おわりに

本研究では，並行システムをアーキテクチャ段階で検証するために，区間の概念を導入し，同時事象を定義した．また，プロセス代数 CSP を用いて同時性を記述した．今後の課題として，区間の概念を拡張して例外処理が複数発生した場合でも，イベントの取りこぼしがなく検証できるようにすることがあげられる．

## 参考文献


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

[2] Formal Systems (Europe), “ *Formal Systems (Europe) Ltd,* ” http://www.fsel.com/, 2010.

[3] 石原脩平，高野寛，山口隼平，”並行システムにおける誤りの分析とパターン化に関する研究，” 南山大学 2012 年度卒業論文，2012.

[4] 中島震，”モデル検査法のソフトウェアデザイン検証への応用，” コンピュータソフトウェア，vol.23，no.2，pp.72-86，2006.