同時性を考慮したアーキテクチャの振舞い検証手法に関する研究

2011SE077 井上啓佑　2011SE264 寺西祐斗　2011SE268 友松良輔
指導教員：張漢明

1　はじめに

並行システムの制御プログラムでは，同時の事象を何らかの同期機構を用いて，逐次で把握する必要がある．設計段階で同時の事象を逐次で把握することは難しく，テストで同時の事象を原因とする問題を発見することは困難といえる．並行システムにおいて設計段階で同時を考慮し設計することがシステムの品質を左右する．設計段階のシステムの振舞いを網羅的に検証する技術にモデル検査がある [5].

本研究の目的は並行システムにおけるアーキテクチャの振舞い仕様記述と検証を支援することである．アーキテクチャの振舞い仕様記述の支援として区間の概念を用いて同時の振舞い記述法を定義し，系統的な検証法を示す.

本研究のアイデアはアーキテクチャ記述の段階で同時性の検証をおこなうための区間の概念の導入による同時性の定式化である．定義した同時性を用いて逐次振舞いと同時振舞いを分離することである．分離した逐次振舞いと同時振舞いを用いて検証をおこなうことで網羅的な検証をおこなう.

本稿ではプロセス代数 CSP[1] による同時性の定義と振舞い仕様モデルおよび検証法を示す自動販売機システムを事例として実際に検証をおこない同時性を考慮した振舞いの定義，また提示した検証法の有用性について述べる.

2　同時性を考慮した並行システムの記述法

2.1　並行システムの計算モデル

並行システムのアーキテクチャは状態遷移機械の集合であり, 状態遷移機械はイベント送信とイベント受理でコンポーネント間のやりとりを記述したものである.
コンポーネントはイベントを送信し, 先に受信したイベントを実行するものとしている.

2.2　CSP

本研究では，モデルを記述する形式仕様言語として CSP(Communicating Sequential Process) を用いる. CSP は, プロセス代数の一つである. CSP はイベントの実行順序を逐次プロセスとして記述することでシステムの振舞いを記述する. CSP では, 複数の逐次プロセスを並行オペレータを用いることで並行動作させ並行システムを表現する.

2.3　同時性の定義

2.3.1　区間

本節では, イベントの実行順序を用いて定義した区間を CSP で定義したものを示す.

- 事象
  - 区間で起こり得る事象
    `EVENT_WITH_SECTION:START*END*EVENT`
    区間は区間の開始となる START イベント，区間の終了となる END イベント，区間内でおこるイベントの EVENT で構成される.
- 区間
  - 事象の集合
    `SECTION:set of EVENT_WITH_SECTION`
    事象の集合が区間となる.
  -「入力区間」と「出力区間」
    `INPUT_SECTION, OUTPUT_SECTION:SECTION`
    区間には入力と出力を行う区間がある.

2.3.2　振舞い

本節では逐次振舞いと同時振舞いを 2.3.1 項で定義した区間を用いて　どのように CSP で定義したかを述べる.

- **逐次**
  SECTION; SECTION
  SECTION が終わった後に，次の SECTION が発生する.
- **選択**
  SELECT(ss:set of SECTION)
  SECTION の集合から選択する.
- **再帰**
  P = SECTION1;SECTION2;P
  SECTION1 の後, SECTION2 が発生し, その後 SECTION1 に戻る.

2.4　振舞い記述

2.4.1　逐次振舞い記述

逐次振舞いは全ての入力区間内で任意のイベントが単一で生起することなので区間の定義を用いた場合, 区間の集合が下記のような振舞いであれば逐次振舞いといえる.

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

start, event, end の順で逐次的に発生し正常終了する.

2.4.2　同時振舞い記述

同時振舞いは全ての入力区間内で任意のイベントが複数生起することである.

下記は同時振舞いの一例である.

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

複数の区間で同時に event が発生し正常終了する.

2.5 振舞い仕様記述

振舞い仕様モデルをどのように CSP を用いて実現したかを述べる.

振舞い仕様モデルを実現するには, アーキテクチャ, 環境, 仕様の CSP 記述が必要である. 本節では逐次振舞い仕様モデルと同時振舞い仕様モデルそれぞれでどのように CSP を用いて実現したかを述べるが, アーキテクチャと仕様の記述は検証対象のシステムと求める仕様に依存した記述になってしまうので, 環境の CSP 記述について述べる.

2.5.1 逐次振舞い仕様記述

逐次振舞いモデルを実現するためには環境記述が同時の振舞いを行わないように制約をかける.
約をかけるために記述した制約式を下記に示す.

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

2.5.2 同時振舞い仕様記述

2.5.1 項では, 逐次振舞い仕様モデルを実現するための制約式を示した. 本節では, 同時振舞い仕様モデルを実現するための制約式を示す.

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

3 同時性を考慮した検証法

3.1 FDR

本研究で用いるモデル検査器には, CSP に対応したモデル検査器 FDR を使用する. FDR では, 形式仕様言語で書かれたアーキテクチャと環境, 仕様の詳細関係を調べることにより検証対象のシステムが仕様を満たすかを調べる. アーキテクチャと環境の記述で検証対象システムの振舞いを表し, 仕様の記述で検証項目を表す. またアーキテクチャと環境の振舞い記述を網羅的に探査することにより, 検証対象のシステムが仕様を満たしていることを保証する.

3.2 検証の枠組み

検証とする対象は並行システムの並行に動作する状態遷移機械の集合と入力の全振舞いを記述した環境を並行合成したシステムとする. 各状態遷移機械は, キューを介した非同期通信をおこなう. 本研究では下記の検証をおこなう.

1. デッドロックフリー
   各コンポーネント内で一つの状態遷移機械から遷移しなくなる状況が起こるか調べる検証
2. ライブロックフリー
   複数の状態遷移機械の遷移を繰り返ししかしなくなる状況が起こるか調べる検証
3. システムが外部仕様を満たしている
   入力した振舞いから期待する出力を外部仕様としたときシステムが等しい振舞いをするかを検証

3.2.1 システムの記述

CSP でシステムを記述する場合, 状態遷移機械と環境の記述を並行合成する. 状態遷移機械は設計されたアーキテクチャに依存する.
環境は入力の全振舞いを表している.

3.2.2 システムが外部仕様を満たしているか調べる検証

検証対象のシステムが外部仕様を満たしている検証をおこなうには, 下記のの検証が成り立てば良い

外部仕様 = システム\{**外部仕様以外のイベント**}

CSP でこの検証をおこなうには外部仕様と検証対象システムが双方で詳細関係が成り立てばよい. 双方での詳細関係を調べる検証が下記である.

1. 外部仕様 ⊑ システムの振舞い
   検証対象のシステムが求められたいる外部仕様を満たしているか調べる検証
   外部仕様の振舞いの詳細化が検証対象のシステムの振舞いであれば検証対象のシステムは求められた外部仕様を満たしていると言える.

2. 外部仕様 ⊑ システムの振舞い\{**外部仕様以外のイベント**}
   検証対象のシステムの記述が正しいか調べる検証.検証対象のシステムの振舞いを外部仕様で記述されているイベント以外を隠蔽する.隠蔽したシステムの振舞いが外部仕様との詳細関係を満たせばシステムの記述は正しいと言える.

```
assert SPEC [FD= TARGET(MODEL, ExternalEvents)
assert TARGET(MODEL, ExternalEvents)
[FD= SPEC
```

3.3 検証手順

本節では検証手順を図 1 で示したことを説明する.
四角は本研究で定義したモデルを示している. 楕円は逐次仕様を示していて, 角のない四角は同時仕様を示している. 点線で囲われている部分は検証手順 1, 検証手順 2 を示し

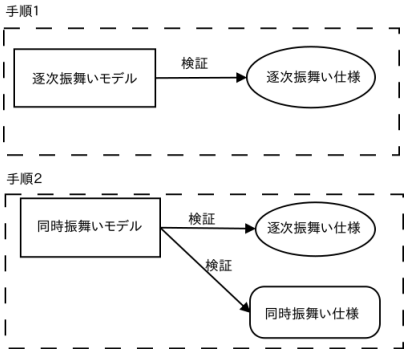

図 1　検証手順

ている．検証手順 1 は逐次振舞いモデルを作成して，逐次仕様を検証をおこなう．

検証手順 2 は逐次振舞いモデルを利用して，同時振舞いモデルを作成する．検証に関しては逐次振舞いモデルで利用した逐次仕様を再利用し，同じ検証をおこなう．その後に同時仕様を検証するという手順で検証をおこなう

.

3.3.1　逐次振舞い仕様モデルの検証

振舞い仕様モデルには，逐次振舞い仕様モデルと同時振舞い仕様モデルがあることを 2.5 節で述べた．本節では，検証項目の作成順序に合わせて逐次振舞い仕様の検証を系統的におこなう方法を提示する．

1. 逐次振舞いモデルを CSP で作成

2. 作成した検証項目を逐次仕様になるように記述し検証

3.3.2　同時振舞い仕様モデルの検証

本節では，アーキテクチャ，環境の CSP 記述は 3.3.1 項で記述したものを使用し検証をおこなう．3.3.1 項で記述した逐次振舞い仕様モデルから同時振舞い仕様に変更する変更点と同時振舞い仕様モデルの系統的な検証法を述べる．

1. 同時振舞いモデルを作成

2. 逐次振舞いモデルで使用した検証項目を再利用し検証

3. 作成した検証項目を同時仕様になるように記述し検証

4　自動販売機システム

本章では，2，3 章で提示した検証法を用いて自動販売機システムを検証をおこなった過程と結果を示す．

4.1　自動販売機システムの設計

検証をおこなうために設計した自動販売機システムのアーキテクチャの逐次振舞いモデルと同時振舞いモデルの状態遷移図で記述したものを提示する

図 2 と図 3 が設計した状態遷移機械である．

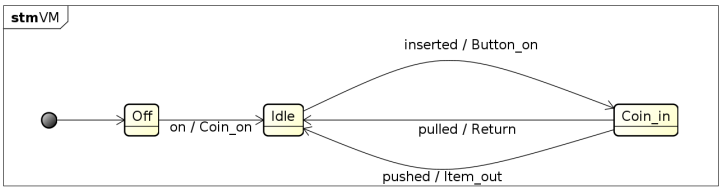

図 2　逐次振舞いモデルの状態遷移機械

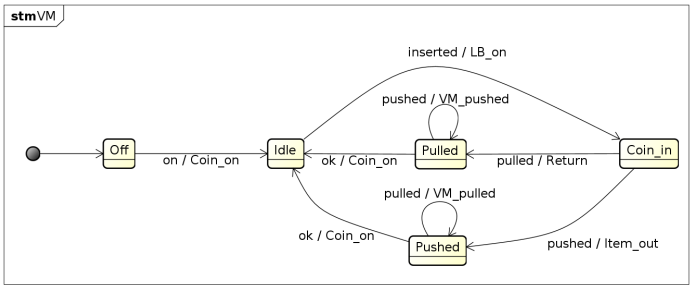

図 3　同時振舞いモデルの状態遷移機械

4.2　振舞い仕様モデルの記述

本節では 2.3.2 項で定義した区間の概念を事例にどう導入したかと 3.3 節で示した検証手順に合わせた際の検証結果を示す．

4.2.1　区間定義

2.3.2 項で示した `INPUT_PURCHASE = DEPOSIT; SELECT; INPUT_PURCHASE`
DEPOSIT は購入区間を表していて，SELECT は選択区間を表している．
本事例の同時振舞いモデルは SELECT に同時性を持たせて検証をおこなう．

4.3　逐次振舞い仕様モデル

2.5.1 項で示した方法と下記で使用する制約式を使用することで逐次仕様の検証を実現した．

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

4.4　同時振舞い仕様モデル

2.5.2 項で示した方法と下記で使用する制約式を使用することで同時仕様の検証を実現した．

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

4.5　検証結果

本節は，3 章で提示した検証法を用いて検証をおこなった．逐次振舞い仕様モデル，同時振舞い仕様モデルでの出

力を述べる.

- **逐次振舞い仕様モデル**コインを入力した後にボタンを押したら商品をだす．またはレバーを引いたらおつりがでるを繰り返す出力になっている.

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

  これを満たすことで確実に逐次の振舞いが満たされている事がわかる

- **同時振舞い仕様モデル**
  コインを入力した後にボタンとレバーを同時に押したら商品またはおつりが排出されるのを繰り返す出力になっている.

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

これらの結果により，設計した自動販売機システムが逐次振舞い，同時振舞いのどちらが環境からの入力としておこったとしてもシステムが止まることなく動く設計ができていることが保証できた.

5　考察

本章では区間の概念を導入することで自然な振舞い記述と提示した検証法についての考察をする.

5.1　仕様の再利用

自動販売機システムでは検証手順で示したように，同時を考慮していないシステムは逐次の振舞いを繰り返すことで，同時が発生しないときには正常に動くシステムであることを示した.

次に検証をおこなう同時を考慮したシステムでも逐次の振舞いを検証をおこなう必要がある．このときに使用する逐次振舞いの仕様は同時を考慮していないシステムで使用した仕様と検証式を再利用することができる.

仕様を再利用できる理由としては同時の振舞いを定義してそれを用いて定式化したことが挙げられる．これを用いることで対象となる設計を変更しても検証式を再利用が可能になった.

5.2　区間の導入

区間の概念を導入することで，振舞いの記述を逐次に記述することが可能になった.

これは実装時における同時性の記述の際にも応用できると考える.

5.3　ボタン複数に増やした場合の検証

区間の概念を導入し，入力を制御する部分を増やした場合の検証を系統的におこなえるといえ, 入力部分を複数に増やした場合の検証も容易におこなえると考える.

6　おわりに

本研究では，区間の概念を定義し，それを CSP に導入することで同時の概念を用いた自然な振舞い記述法とそれを利用した同時を考慮した振舞い記述に基づいた系統的な検証法を提示した．簡単な例を用いて，定義した概念の有用性を確認した．今後の課題として，引き続き事例を用いて定義した概念の有用性を示す必要性があると考える.

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] 近藤翔太, ” 並行システムにおけるソフトウェアの検証手法に関する研究ー際どい実行順序の検証についてー, ” 南山大学 2013 年度卒業論文，2013.

[5] 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.

[6] 吉岡信和, 田辺良則, 田原康之, 長谷川哲夫, 磯辺祥尚 ” モデル検査による設計検証, ” コンピュータソフトウェア, vol.31, no.4 pp.40-63, 2014 ”

[7] 米澤明憲, 柴山悦哉, モデルと表現, 岩波書店，1992.