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
―際どい実行順序の検証について―

2009SE135 近藤翔太
指導教員：張漢明

1　はじめに

並行システムを検証する手法として，モデル検査の有用性 [3] が報告されている．モデル検査では，対象とする並行システムが取りうる振舞いを網羅的に走査し，システムの満たすべき性質の真偽を自動的に判定する．モデル検査の問題点として並行プロセス特有の仕様を書く事の難しさがあげられる．

本研究の目的は，並行システムにおける際どい実行順序を考慮した検証手法を提示することである．本研究では際どい実行順序として，イベント競合に着目して，際どい実行順序を分析して定式化することにより，その仕様と検証式を生成する方法を提示する．イベント競合の振舞いの定式化と検証手順を提示することで，イベント競合の検証コストを低減し，定式化はイベント競合のテストケース作成の規準となる．

本稿ではプロセス代数 CSP[1] を用いてイベント競合の形式的な定義を示し，自動販売機システムの事例を用いて検証手法の有用性について議論する．

2　基本的なアイディア

本研究の基本的なアイディアはイベント競合の仕様と検証式を生成する方法を提示することである．本章では検証の枠組みとイベント競合について説明する．

2.1　検証の枠組み

並行システムを並行に動作する状態遷移機械の集合と捉える．各状態遷移機械は，キューを介した非同期通信を行う．本研究では，イベントの送受信に着目をして，仕様と環境と対象となるシステムを CSP を用いて記述する．環境にはシステムに対する外部からの入力を記述し，仕様にはシステムに期待する振舞いを記述する．環境，システム，仕様について以下の式が成り立つ場合，仕様が満たされているといえる．[2]

仕様　⊑　対象システム　‖　環境

2.2　イベント競合

イベント競合とは，複数のイベントが送信されることである．イベント競合の起こるイベントの実行順序について説明する．図 1 と図 2 はそれぞれ競合が起きない場合とコントローラ B で競合が起きる場合のイベントの実行順序の一例である．

制御コンポーネントはコントローラ A からのイベントを受理すると，コントローラ B を Off にする．Off になったコントローラはそれ以上制御コンポーネントへ通知イベントを送信しない．図 2 のようにコントローラ B が Off になるまでの間にイベントが送信されてしまった場合に競合が起きる．

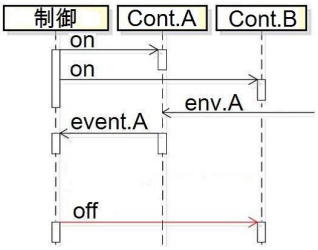

図 1　競合が起きない場合

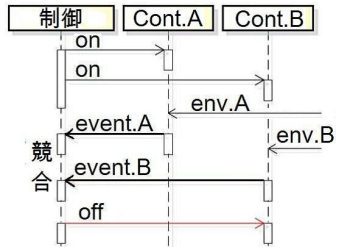

図 2　競合が起きる場合

3　イベント競合の定式化

イベント競合の振舞いを定式化し，CSP を用いて記述した．定式化したイベント競合の振舞いについて説明する．

3.1　競合に関連するイベント

競合を定義するために必要なコントローラに関するイベントは，以下の 3 種類のイベントである．これらはそれぞれ図 2 の env,event,off に対応している．

トリガーイベント　コントローラが検知するイベント

通知イベント　コントローラが通知するイベント

オフイベント　コントローラの検知を停止するイベント

トリガーイベントの前に必ずコントローラの on が送信されるので，on イベントには着目しなくてよい．上記のイベントを制御イベントとして 3 項組で表す．

3.2　競合の定式化

図 3 が定式化したイベント競合の CSP 式である．それぞれの CSP 式について説明する．

```
CONF_F(Cnts,first) =
  (TRIGGER({first}); NOTIFY({first});
   NOTIFY(Except(Cnts,{first})))
|||
  (TRIGGER_OFF(Except(Cnts,{first})))
CONF_N(Cnts) =
  []first:Cnts @ CONFLICT_F(Cnts,first)
CONF_MN(Cnts,Conf) =
  CONF_N(Conf)||| Off(Except(Cnts,Conf))
```

図 3　定式化したイベント競合の振舞い

3.2.1 CONF_F

CONF_F は複数のコントローラでトリガーイベントが生起した場合，通知イベントが最初に通知されたコントローラの振舞いを表す．TRIGGER，NOTIFY，OFF はそれぞれトリガーイベント，通知イベント，オフイベントが順番に関係なく生起するプロセスを表している．EXCEPT(S,E) は集合 E 以外の集合 S を表している．TRIGGER_OFF はオフイベントの前にトリガーイベントが生起するプロセスを表している．Cnts は制御イベントの集合，first は通知イベントが最初に通知されたコントローラの制御イベントを表している．

3.2.2 CONF_N

CONF_N は n 個のコントローラで n 個のトリガーイベントが競合する振舞いを表す．

3.2.3 CONF_MN

CONF_MN は m 個のコントローラで n 個のトリガーイベントが競合する振舞いを表す．Conf は競合するコントローラの制御イベントの集合を表している．

3.2.4 拡張性

CONF_N は全てのコントローラで競合が発生する場合，コントローラの数を変化させても同様に振舞いを表すことができる．また CONF_MN は CONF_N の場合に加えて，競合が発生しないコントローラが含まれている場合の振舞いも表すことができる．

4 検証事例:自動販売機システム

本研究で事例として扱う自動販売機システムは，コントローラとして商品ボタン，返金レバーの 2 つを想定した．コインが投入された後，商品ボタンが押されると商品を排出し，返金レバーが引かれるとコインを排出する．商品ボタンと返金レバーが同時に操作された場合は必ず商品排出を優先する仕様を想定した．自動販売機システムを用いた検証の事例について示す．

4.1 定式化した競合の振舞いの適用

自動販売機システムの制御イベントを CONF_N に適用した．

```
CONF_N(Cnts) =
  []first:Cnts @ CONF_F(Cnts,first)
```

Cnts は競合するコントローラの制御イベントの集合を表す．

```
Cnts = {ItemButton , ReturnLever}
ItemButton  = (IB.pushed, VM.selected, IB.off)
ReturnLever = (RL.pulled, VM.ejected, RL.off)
```

SPEC_ITEM は同時に操作されたときの仕様である商品排出の振舞いを表す．

```
SPEC_ITEM = IS.out -> SPEC_ITEM
```

4.2 検証

システムとイベント競合の振舞いを合成したものを SYSTEM と定義し，検証を行った．

```
assert SPEC_ITEM [F = SYSTEM
```

結果は true となり，イベント競合の検証ができた．

5 考察

イベント競合の振舞いがより複雑なシステムへ適用できるかとテストケースの作成について考察する．

5.1 競合検証の有用性

定式化したイベント競合の振舞いがより複雑なシステムに対しても適用できるかどうかを考察する．例として図 4 の自動販売機システムを考えた．競合が起きる箇所はコンポーネントに対して複数の入力がある部分なので，太線部分の 4 箇所と考えられる．これらの部分に定式化したイベント競合の振舞いが適用できると考える．

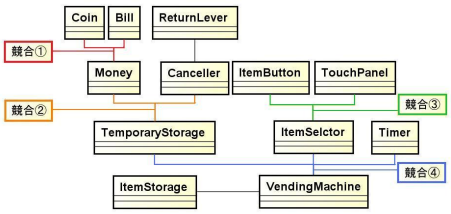

図 4 競合の起きる自動販売機

5.2 状態遷移テストのテストケースの作成

定式化したイベントの振舞いから実際にテストケースが生成できるか考察する．定式化したイベントの振舞いからイベント競合のトレースが生成できる．CSP から状態遷移図への変換が可能なので状態遷移テストのテストケースとして活用できると考える．

6 おわりに

本研究では自動販売機システムを事例として検証を行い，競合を CSP の制約で記述することと検証手法の提示を行った．今後の課題として，より複雑なシステムでの有用性検証を行うことと，競合以外の際どい実行順序の発見や分析を行っていくことがあげられる．

7 参考文献

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

[2] 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

[3] 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