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

2009SE081 石原脩平　2009SE275 高野寛　2009SE309 山口隼平
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が報告されている [7]．モデル検査では，システムの振舞いを網羅的に走査し，満たすべき性質の真偽を自動的に判定する．判定結果が偽の場合は反例を出力する．反例とはシステムが満たすべき性質に反するときのイベントの列のことである．反例はフォールトの特定において重要な情報であるが，反例からのフォールトの特定は困難な作業であり，検証コスト増大の要因の一つとなる．

本研究の目的は，事例から並行システムにおける典型的なフォールトの抽出，およびその有用性を確認することである．フォールトとは，システムを異常な状態に導く可能性のあるソフトウェアの欠陥である [3]．フォールトパターンを用いることにより，構文解析レベルでフォールトを検出する．モデル検査前にフォールトパターンを適用し典型的なフォールトを検出することにより，検証コストの削減を目指す．

本研究では，フォールトの発生箇所を明確にするために，最も基本的なシステムを設計し，それを段階的に検証する．基本システムでの全ての検証項目が真であるならば，基本システムに機能を一つ追加した上で，改めてそれぞれの検証を行なう．本稿では，プリンタシステムと自動販売機システムのを事例として扱う．発見したフォールトを分析してパターン化を行ない，パターンによるフォールトの検出方法を提示する．

本研究の結果として，プリンタシステムと自動販売機システムを事例として検証を行なった．競合未対応，イベントキュー満杯，イベント順序逆転，Req(要求)-Rsp(応答)間の割り込み，プロトコルの相違，モード切り替え時のイベント消失，論理的な設計ミスという 7 種類のフォールトを提示した．

2　背景技術

2.1　計算モデル

本研究では，並行システムを並行に動作する状態遷移機械 (State Transition Machine，以下 STM と呼ぶ) の集合と捉える．各 STM は，一つずつ有限長のキューを持ち，STM 同士でキューを介した非同期通信をするものとする．本計算モデルでは，イベント送信とイベント受理をシステムの振舞いとして記述している．本研究における計算モデルを図 1 に示す．

イベント送信　STM を指定し，イベントを相手のキューに積む

イベント受信　キューが満杯でなければキューの最後尾にイベントを追加する

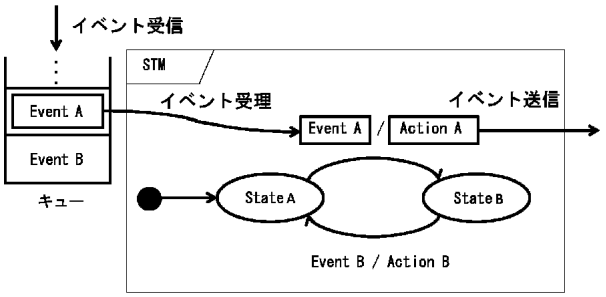

図 1　計算モデル

イベント受理　キューの先頭から順に走査し，受理可能なイベントを見つけてキューから取り出し，アクションを実行する

イベント受理によるアクションによりイベント送信を行なう．なお，本計算モデルではイベント送信とイベント受信は同時に行なわれるものとみなしている．

2.2　検証の枠組み

本研究では，並行システムを形式的に記述するプロセス代数である CSP[1] と，CSP の代表的なモデル検査器である FDR[2] を用いる．本研究での検証の枠組みを図 2 に示す．

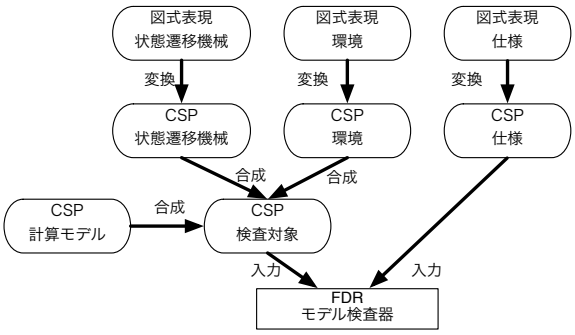

図 2　検証の枠組み

図式表現の対象システムと，環境，仕様を CSP へ変換する．環境とはシステムの外部からシステムへ送信するイベントの集合であり，仕様とは環境を受けてシステムに期待する振舞いである．計算モデルと，対象システム，環境の CSP を合成した検査対象と，仕様の CSP を FDR へ入力し検証を行なう．

2.3 フォールトパターンによるフォールトの検出

モデル検査を行なう前にフォールトパターンを検査対象に適用し，あらかじめフォールトを検出して修正することにより，反例分析のコストの軽減が可能となる．フォールトパターンによるフォールトの検出の枠組みを図 3 に示す．

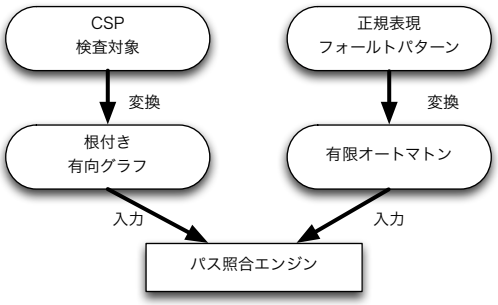

図 3 フォールトパターンによるフォールトの検出の枠組み

CSP 記述の並行システムを根付き有向グラフに変換し，また正規表現で記述されたフォールトパターンは有限オートマトンに変換する．パス照合エンジンは，有向グラフの根を先頭として，フォールトパターンと一致する振舞いを検査対象から検出することにより，パスの照合問題に帰着する．

3 モデル検査の方針

3.1 基本的なアプローチ

本研究では，事例とするシステムに対し実際にモデル検査を行なう．事例検証により発見されたフォールトを収集し分析を行なう．

事例とするシステムとして，印刷命令を格納するバッファにおけるバッファリングでの誤りを想定してプリンタシステムを扱った．また，複数な機能を段階的に追加していくことを想定して自動販売機システムを扱った．

自動販売機システムでの検証において，基本的な自動販売機システムに機能を段階的に追加していき検証を行なう．

検証項目として以下の項目を想定した．

単機能の繰り返し 単一の機能に着目した入力の繰り返し

機能の組合わせ 機能を組合わせて検証

機能の競合 複数の入力の同時の繰り返し

全入力の検証 デッドロックが起こらないか検証

3.2 事例：自動販売機システム

本稿では自動販売機システムについて説明する．本研究で事例として扱う基本的な自動販売機システム（以下，基本システム）は，商品が 1 つ，商品の価格がコイン 1 枚分である．基本システムのクラス図を図 4 に示す．

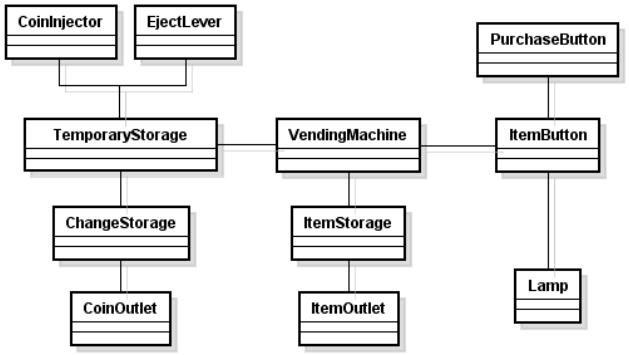

図 4 基本システムのクラス図

自動販売機システムでの検証では，まず基本システムの検証を網羅的に行なう．続いて，事例検証の方針に従って，単純なシステムを段階的に複雑にしていき検証を行なう．

基本システムに追加する機能として，商品複数，コイン複数，在庫管理，金庫管理，保守，実時間処理，例外処理といった機能を想定した [5]．自動販売機システムの機能についてのフィーチャ図を図 5 に示す．

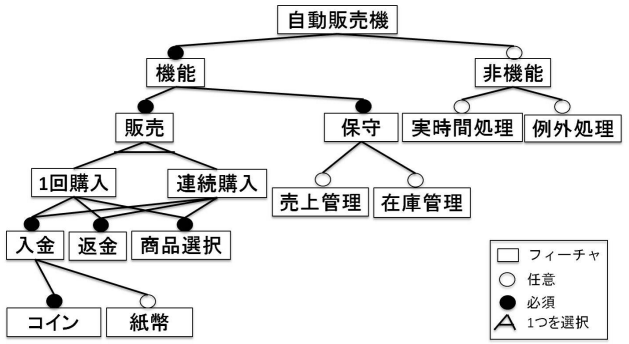

図 5 自動販売機システムの機能についてのフィーチャ図

4 フォールトの分析

プリンタシステム，自動販売機システムの事例に対し検証を行ない，67 個のフォールトが見つかった．

発見したフォールトの原因を分析し，7 種類のフォールトに分類をした．フォールトの分類を表 1 に示す．

表 1 フォールトの分類

1	競合未対応
2	イベントキュー満杯
3	イベント順序逆転
4	Req(要求)-Rsp(応答) 間の割り込み
5	プロトコルの相違
6	モード切り替え時のイベント消失
7	論理的な設計ミス

本稿では制御側の競合未対応とイベントキュー満杯のフォールトの原因について説明する．

4.1 制御側の競合未対応

競合未対応とは競合を考慮していないことにより送信されたイベントが受理できないフォールトである．

競合とは，ある状態のときに複数のイベントが送信されることである．競合について図 6 に示す．

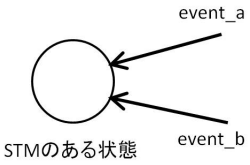

図 6 競合

競合には制御側と被制御側がある．自動販売機システムを例に紹介する．VendingMachine は ItemButton をオン，オフにすることで制御を行なう．このとき VendingMachine は制御側コンポーネント，ItemButton は被制御コンポーネントとなる．

制御側の競合未対応とは制御コンポーネントである STM に競合を考慮した記述がないことによるフォールトである．制御側の競合未対応はタイマーを追加した自動販売機システムで見つかった．VendingMachine は ItemButton，Timer という複数のコンポーネントの制御を行う．VendingMachine の STM を図 7 に示す．

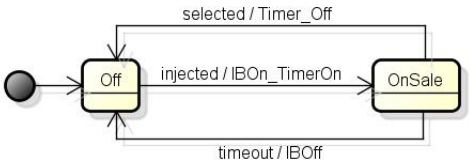

図 7 VendingMachine の STM

VendingMachine は ItemButton，Timer をそれぞれ有効状態にすると，1 つのイベントを待つ．ItemButton からのイベントを受理すると Timer をオフに，Timer からのイベントを受理すると ItemButton をオフにする．

制御側の競合を図 8 に示す．

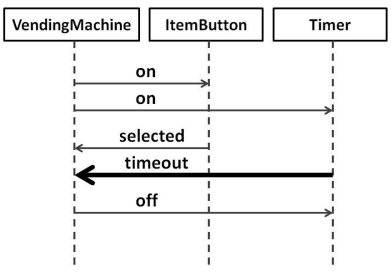

図 8 制御側の競合

VendingMachine が ItemButton からのイベントを受理し，Timer がオフになるまでの間に Timer がイベントを送信してしまうと競合が起きる．VendingMachine の STM には selected イベントを受理した後に timeout イベントを受理できる記述がないので，イベントは受理できずキューに格納されてしまう．

4.2 イベントキュー満杯

イベントキュー満杯は，STM のキューが受理不可能なイベントで満たされ，デッドロックとなるフォールトである．キュー満杯のフォールトの原因を図 9 に示す．

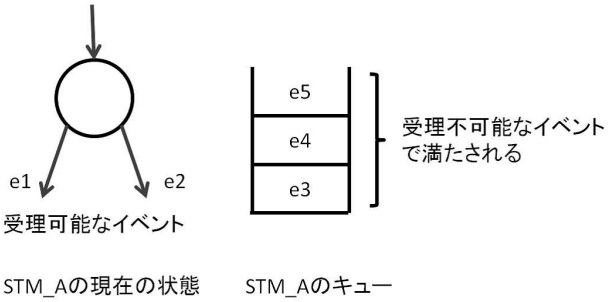

図 9 イベントキュー満杯

STM_A はある状態のとき，e1 イベントと e2 イベントが受理可能である．この状態のときキューが受理可能なイベント以外で満たされるとイベントは送信できず，キューが空くまで待機する．受理可能なイベントを送信することができなくなり，デッドロックになってしまう．

5 考察

分析したフォールトからフォールトパターンにできるか考察した．競合未対応について既存のフォールトパターンの有用性を確認し，キュー満杯，Req(要求)-Rsp(応答) 間の割り込みの 2 つを新たにフォールトパターンとして定義した．本稿では既存のフォールトパターンの有用性確認とキュー満杯のフォールトパターンについて説明する．

5.1 既存のフォールトパターンの有用性確認

既存のフォールトパターンについて本研究の競合未対応の事例に適応することができた．既存のフォールトパターンについて説明する．

既存のフォールトパターンは 1 つのイベントに着目し，イベントの送信と受理が繰り返す中でイベントの送信があるが，イベントの受理が存在しないというパターンである．

イベントの送信と受理は以下にのように記述する．

- 送信イベント：`R<-event@S`
- 受理イベント：`R.event@S`
  - 制御コンポーネント：R，被制御コンポーネント：S

このとき，期待する振舞いは以下のように定義される．

```
BH = (R<-event@S; R.event@S)*
```

BH:**期待する振舞い**

＊ :繰り返し

期待する振舞いに対してフォールトパターンは以下のように定義される．

```
FP = BH; R<-event@S; R<-event@S
```

FP:フォールトパターン

このフォールトパターンはイベントの送信に対して受理がないということをイベントの 2 回送信で表現している．1 回目のイベント送信が受理されない状態で 2 回目のイベント送信が存在することはフォールトとなりえる．

事例に挙げたタイマー付き自動販売機システムに対してこのパターンを当てはめた．結果, パターンを検出できた．従ってこのパターンはフォールトを見つけるのに役立つと考えられる．

5.2 イベントキュー満杯のフォールトパターン

フォールトの分析で分類したキュー満杯についてフォールトパターンを定義する．

このパターンでは前提条件としてバッファ (図 10) が存在する．

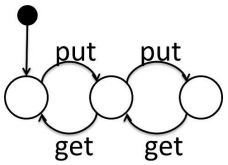

図 10 バッファの STM

バッファでやり取りされるイベントは以下の通りに定義する．

- データ格納イベント：put
- データ取得イベント：get

ここでの期待する振舞いは，put イベントの後に get イベントの送信が存在することである．イベントキューのサイズが 2 であると仮定した場合，フォールトパターンのオートマトン記述 (図 11) は以下のように定義される．

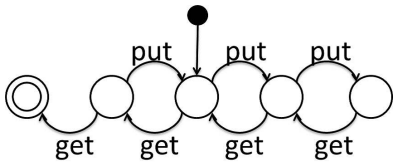

図 11 イベントキューのフォールトパターン

このフォールトパターンはイベントキューが受理不可能なイベントで満杯となったことを受理不可能なイベントの 2 回送信で表現している．

メンテナンス機能を追加した自動販売機システムの検査対象に対してこのパターンを当てはめた．結果，フォールト検出することができた．従ってこのパターンはフォールトを見つけるのに役立つと考えられる．

5.3 関連研究との比較

並行システムにおける既存のデバッグ支援の研究として，反例の視覚化 [6]，および，反例の自動解析 [4] に関する研究がある．本研究とこれらを比較することで，パターンによるフォールト検出の有用性について考察する．

反例の視覚化に関する研究では，反例におけるイベントの生起順序を図示することでデバッグを支援する．反例は見やすくなるが，フォールトを特定するにはシステムの意味を考えなくてはならない．本研究では，典型的なフォールトをパターンとして定義することで，構文レベルの解析によりフォールトを特定することができる．

反例の自動解析に関する研究では，一つの意味に限定して解析することでデバッグを支援する．自動的にフォールトの修正箇所まで指摘できるが，一つのフォールトしか検出できない．本研究では，フォールトの完全さを捨てる代わりに多くの可能性があるフォールトを検出することができる．

6 おわりに

本研究では，並行システムの事例に対しモデル検査を行ない，発見したフォールトの分析を行って 7 種類のフォールトに分類をした．また競合未対応のパターンの有用性を確認し，イベントキュー満杯，Req-Rsp 間の割り込みのフォールトのパターンを定義した．

今後の課題として，本研究で定義したフォールトパターンの有用性の確認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] I. Sommerville, *Software Engineering*, Addison-Wesley, 2007.

[4] T. Bochot, P. Virelizier, H. Waeselynck, and V. Wiels, *Paths to property violation: a structural approach for analyzing counter-examples*, 2010 IEEE 12th International Symposium on HASE, pp.74-83, 2010.

[5] 鰺坂恒夫, 池田健次郎, 中谷多哉子, 野呂昌満, “ OO’97 オブジェクト指向モデリングワークショップ報告, ” 情報処理学会研究報告. ソフトウェア工学研究会報告, pp.33-35，1997.

[6] 陳適，青木利晃, “ モデル検査ツールにより出力された反例に基づく誤り特定に関する研究, ” 情報処理学会研究報告，vol.2012-SE-177，no.6，pp.1-8，2012.

[7] 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.