パターンを用いた並行システムにおけるフォールトの検出に関する研究

M2011MM066 杉浦友紀
指導教員：野呂昌満

1　はじめに

並行システムは大規模化，複雑化により，仕様を満たしているか把握することが困難である．モデル検査は，システムの満たすべき性質の真偽を自動的に判定するので，並行システムの検証に有用である．モデル検査の結果が偽となる場合，満たすべき性質が偽となるまでの実行列を反例として出力する．反例はシステム記述のフォールトを特定する上で重要であるが，反例からフォールトを特定することは労力のかかる作業であり，検証コストが高くなる要因の一つである [3].

本研究の目的は事例から並行システム記述の典型的なフォールトを分析し，フォールトパターンの有用性を確認することである．典型的なフォールトをフォールトパターンとして提示する．フォールトパターンとシステム記述のパスをパス照合することで，フォールトの特定を構文レベルの解析で行なうことができる．本研究では，組込みシステムの事例から典型的なフォールトを分析し，フォールトパターンを抽出する．

本研究では，並行システムの事例を設計し，モデル検査を通じて典型的なフォールトを分析する．並行システムの事例としてプリンタシステム，自動販売機システムをあげる．フォールトを分析する際，複数のフォールトが混入しないよう段階的に機能を追加し，検証を行なった．事例を段階的に検証することで機能ごとのフォールトを分析する．典型的なフォールトのパターンを抽出する上で，パターン化できるもの，できないものに分類する．

本研究の結果として，プリンタシステム，自動販売機システムの事例に対し，242 回モデル検査を行ない，67 個のフォールトを検出した．67 個のフォールトから 6 種類の典型的なフォールトを分析し，3 種類のフォールトパターンを抽出した．

本研究の評価として，デバッグ時におけるパターン検出の利用方法，事例を用いてフォールトパターンの有用性を考察する．

2　背景技術

並行システムの計算モデル，モデル検査の枠組みについて述べる．

2.1　計算モデル

本研究で扱う計算モデルを説明する．システムを並行に動作する状態遷移機械 (以下，STM) の集合と捉える．各 STM は，有限長のキューを一つ持ち，STM 同士でキューを用いた非同期通信を行なう．本研究における計算モデルを図 1 に示す．

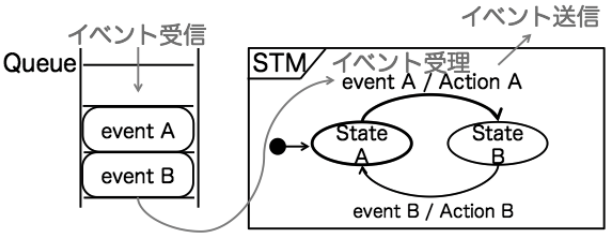

図 1　計算モデル

イベントの送信，受信，受理は以下のように定義する．

イベント送信
　イベントを送信先の STM のキューに送信

イベント受信
　キューの最後尾に送信されたイベントを格納

イベント受理
　受理可能なイベントをキューから取り出して，イベントに応じたアクションを実行

2.2　モデル検査の枠組み

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 CSP[1] と，CSP の代表的なモデル検査器 FDR[2] を用いてモデル検査を行なう．本研究で行なうモデル検査の枠組みを図 2 に示す．

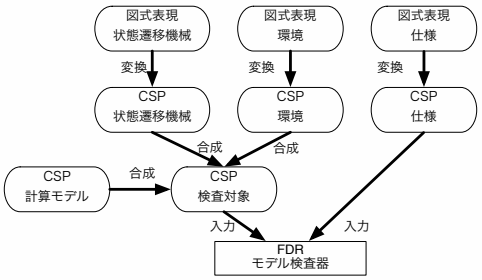

図 2　検証の枠組み

STM，環境，仕様をそれぞれ CSP に変換する．STM，環境，計算モデルを合成したものと仕様を FDR に入力し，モデル検査を行なう．FDR は，トレースモデル (trace model)，失敗モデル (failure model) での詳細化関係を調べることで安全性，活性の検証を行なうことができる．

3　研究方針

本研究では，並行システムの事例を設計し，モデル検査を通じて典型的なフォールトを分析し，パターン化する．

3.1 フォールトパターンの導出方法

本研究で行なうフォールトパターンの導出方法を以下に示す.

1. 典型的なフォールトを分析
   1.1 フォールトの収集
       事例を用いてモデル検査
   1.2 典型的なフォールトを分析
       イベントの生起順序に着目し，分析
2. フォールトパターンを提示
   2.1 典型的なフォールトをパターン化
       イベントの生起順序を正規表現で表現
   2.2 事例に適用し，有用性を確認
       2.2.1 自ら考えた事例に適用
       2.2.2 他者が考えた実用的な事例に適用

3.2 段階的な検証

本研究で行なう段階的な検証を示す．段階的に検証する手順を図 3 に示す.

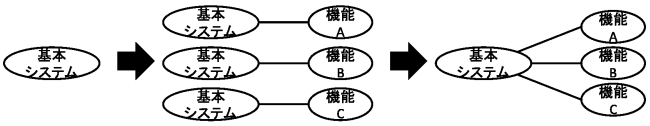

図 3 段階的な検証

基本的な機能を実現するシステムを基本システムとして設計する．基本システムの検証を行い，段階的に機能追加を行なう．基本的なシステムから機能を追加したシステムを段階的に検証することで，プリミティブなフォールトを分析する.

3.3 検査項目

本研究でモデル検査を行なう際の検査項目を示す．検査項目に対して期待する振舞いが繰り返し行なわれるかを検査する．検査項目を以下に示す.

単機能
　一つの機能の振舞い

機能の組合わせ
　一つの機能と他の機能を組合わせた際の振舞い

競合
　複数の入力を同時に行なう

全入力
　考えられる全ての入力をランダムに行なう

競合とは，1 つの STM に複数のイベントが同時に送信されることである．全入力の検査ではデッドロックフリーであることを検査する.

3.4 並行システムの事例

本研究での並行システムの事例を示す.

3.4.1 プリンタシステム

本研究の並行システムの事例であるプリンタシステムの概要を示す．印刷と印刷するフォーマットの変更を基本的な機能とする．印刷中に送られた印刷データは，バッファに格納され，印刷終了時にバッファから取り出す．プリンタシステムのクラス図を図 4 に示す.

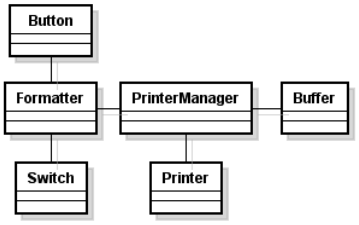

図 4 プリンタシステムのクラス図

3.4.2 自動販売機システム

本研究の並行システムの事例である自動販売機システムの概要を示す．商品購入と返金を基本的な機能とする．オブジェクト指向モデリングに関する文献 [4] を基に作成した自動販売機システムのフィーチャ図を図 5 に示す.

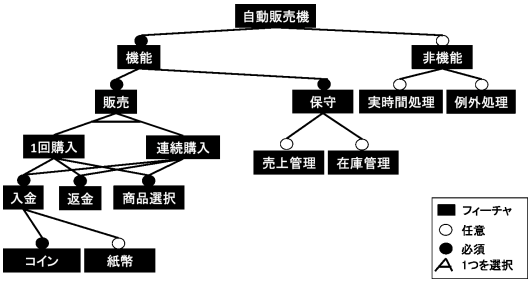

図 5 自動販売機システムのフィーチャ図

4 典型的なフォールト

本研究で抽出した典型的なフォールトを示す．本研究でのモデル検査の結果として，検査数とフォールトの数を表 1 に示す.

表 1 検査数とフォールトの数

対象	プリンタシステム	自動販売機システム	
		基本	機能追加
検証数	54	61	127
フォールトの数	16	22	29

67 個のフォールトを検出し，以下の 6 種類の典型的なパターンを抽出した.

1. 競合未対応　2. イベントキュー満杯
3. 要求-応答間の割り込み　4. イベント順序逆転
5. プロトコルの相違　6. 論理的な設計ミス

本稿では，競合未対応，イベントキュー満杯，要求-応答間の割り込みのフォールト，およびパターンを示す．

4.1 競合未対応

競合未対応のフォールトを示す．競合未対応は競合時の振舞いを考慮せず，期待する振舞いと反してしまうフォールトである．一般例として STM_A 内で STM_B，STM_C から送信されるイベントが競合する例をあげる．STM_A の STM を図 6，STM_B の STM を図 7 に示す．STM_C は STM_B と同様の構造である．

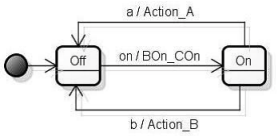

図 6 競合の例 (STM_A)

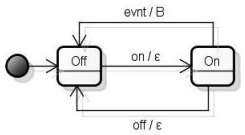

図 7 競合の例 (STM_B)

競合時の振舞いを図 8 に示す．

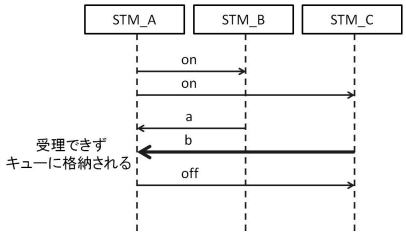

図 8 競合の振舞い

STM_A が a を受理し，STM_B に off を送信する間に b が送信され，受理できずキューに格納されたままとなる．

4.1.1 競合未対応のパターン

競合未対応のフォールトパターンを示す．あるイベントの送信があり，受理がないものをパターンとして以下のように記述する．

```
FP=BH;R<-event@S;!R.event@S --パターン
BH=(R<-event@S;R.event@S)* --正しい振舞い
```

R<-event@S はあるイベントの送信を表し，!R.event@S はそのイベントの受理が無いことを表す．

4.2 イベントキュー満杯

イベントキュー満杯のフォールトを示す．イベントキュー満杯は，STM のキューが受理できないイベントで満たされ，受理可能なイベントを受信することができないフォールトである．イベントキュー満杯のフォールトを図 9 に示す．

STM_A は受理できないイベントでイベントキューが満杯になると，受理できるイベントを受信することができず，デッドロックになる．

4.2.1 イベントキュー満杯のパターン

イベントキュー満杯のフォールトパターンを示す．STM がある状態で受理不可能なイベントがイベントキューのサイズ分送信されているものをパターンとして図 10 に示す．

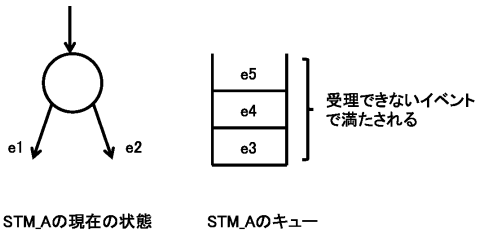

図 9 イベントキュー満杯

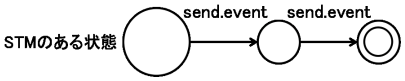

図 10 イベントキュー満杯のパターン

これはキューのサイズが 2 であるときのパターンである．STM がある状態のときの受理不可能なイベントの送信を send.event として表す．

4.3 要求-応答間の割り込み

要求-応答間の割り込みのフォールトを示す．要求-応答間の割り込みとは，要求，応答を表すイベントの間にあるイベントが割り込むことで期待する振舞いが行なわれないフォールトである．要求-応答間の割り込みのシーケンス図を図 11 に示す．

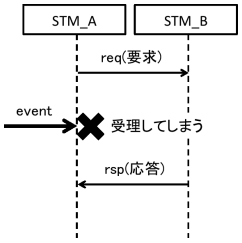

図 11 要求-応答間の割り込み

STM_A は STM_B に req(要求) を出し，STM_B はその要求に対する rsp(応答) を返す．要求-応答間にイベントを受理することで，期待する振舞いが行なわれない場合がある．

4.3.1 要求-応答間の割り込みのパターン

要求-応答間の割り込みのフォールトパターンを示す．要求を表すイベント，応答を表すイベントの間に，あるイベントの受理があるものをパターンとして以下のように記述する．

```
FP=BH;S.req@R;R.event@S
BH=(R.event@S+(S.req@R;R.rsp@S))*
```

S.req@R は要求の受理，R.rsp@S は応答の受理，R.event@S は要求，応答以外のあるイベントを表す．パターンは要求を受理した後，応答の受理ではなく，あるイベントの受理があることを表す．

5　考察

本研究の考察として，デバッグ時におけるパターン検出の利用方法，フォールトパターンの有用性を考察する．

5.1　デバッグ時におけるパターン検出の利用方法

並行システム記述におけるデバッグを行なう際に有用なフォールトパターンの利用方法を考察する．デバッグ時におけるパターン検出の利用方法を以下に示す．

- 環境の間違いを検出
- 仕様と異なるイベントの生起順序を検出

環境の間違いを検出

環境の間違いをパターンを用いて検出する方法を示す．本研究でモデル検査を行なった際，環境が想定するものと異なり，正しく検査が行なわれないことがあった．環境の間違いを検出することは，並行システム記述におけるデバッグ時に有用であると考えられる．環境の間違いを検出するには，想定する環境をパターンとすることで，環境が正しいかを調べることができる．

仕様と異なるイベントの生起順序を検出

仕様と異なるイベントの生起順序を検出する方法を示す．本研究でモデル検査を行なった際，仕様とどう違反しているかを反例から特定することが困難なことがあった．仕様と異なるイベントの生起順序を検出することは，並行システム記述におけるデバッグ時に有用であると考えられる．仕様と異なるイベントの生起順序を検出するには，仕様から仕様を違反するイベントの生起順序を導き，フォールトパターンとすることで仕様とどう違反しているかを検出することができる．

5.2　フォールトパターンの有用性

フォールトパターンの有用性の評価として，競合未対応のパターンが事例に適用できるかを考察する．パターンを適用する事例として，複数の機能を組み合わせた自動販売機システムをあげる．複数の機能を組み合わせた自動販売機システムのクラス図を図 12 に示す．

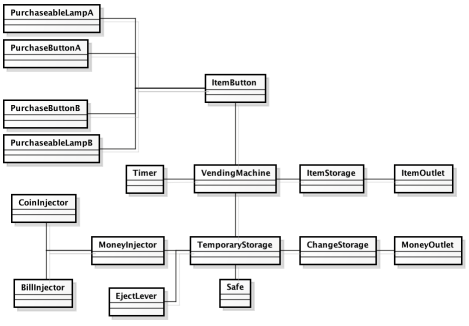

図 12　複数の機能を組合わせた自動販売機のクラス図

このシステムには競合が発生する STM が 10 個存在する．競合が発生する STM の例として，MoneyInjector をあげる．MoneyInjector はコイン投入口 (CoinInjector)，紙幣投入口 (BillInjector) を管理している．MoneyInjector の STM を図 13 に示す．

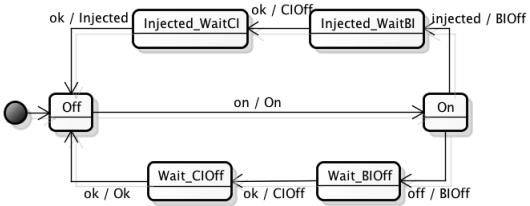

図 13　競合が起こる STM(MoneyInjector)

MoneyInjector が On 状態のとき，TemporaryStorage から送信される off と CoinInjector，BillInjector から送信される injected が競合する．この例では off，injected のどちらか一方しか受理することがない．これは競合未対応のパターンで検出できる．

MoneyInjector の例と同様に競合が起こる箇所全てが競合未対応のパターンにより検出できるので，競合未対応のパターンは有用であると考えられる．

6　おわりに

本研究のまとめとして，本研究の成果，今後の課題について述べる．本研究では，自動販売機システム，プリンタシステムを事例とし，モデル検査を通じてフォールトを収集し，6 種類の典型的なフォールトを抽出した．6 種類の典型的なフォールトから 3 種類のフォールトパターンを提示した．デバッグ時におけるパターン検出の利用方法，フォールトパターンの有用性を考察した．今後の課題として，他の事例からより多くのフォールトを収集し，フォールトパターンを抽出すること，他者が考案した実用的な事例に対して，パス照合エンジンを用いて実際にフォールト検出を行ない，有用性を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] T. Bochot, P. Virelizier, H. Waeselunck, and V. Wiels, *Paths to property violation: a structural approach for analyzing counter-examples,* 2010 IEEE 12th International Symposium on HASEHigh-Assurance Systems Engineering, vol.3, no.4, pp74-83, 2010.

[4] 鯵坂恒夫, 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.