パターンを用いたモデル検査に関する研究
– フォールトパターンの分析 –

2008MI066 石田 空士
指導教員　張 漢明

1　はじめに

並行システムを検証する手段にモデル検査がある．モデル検査ではシステムの満たすべき性質の真偽を自動的に判定し，判定が偽の場合は反例を出力する．反例とはシステムの満たすべき性質が偽になるイベント列である．反例は誤りを特定するための重要な情報であるが，反例からの誤りの特定は本質的に難しい作業であり，検証に高いコストがかかる要因の一つになっている．

本研究の目的は並行システムにおける典型的なフォールトをパターンを用いて検出する方法を提示することである．フォールトとはシステムを異常な状態に導く可能性があるソフトウェアの欠陥のことである．パターンを用いることで構文レベルの解析で典型的なフォールトを検出できるようにする．パターンによって典型的なフォールトをモデル検査前に取り除くことで検証にかかるコストを軽減する．

本研究の方針は並行システムの既知の相互排除問題を分析し，典型的な誤りをフォールトパターンとして定義することである．パターンは共有資源の操作に関する振舞いを正規表現で記述する．誤りをパターンにすることで，フォールト検出をパターン照合問題に帰着することができる．並行システム記述をラベル付きオートマトンとみなすことによりフォールト検出を自動化できる．

本稿では相互排除問題として生産者-消費者問題 [2] を題材としてフォールトパターンの定義を示す．セマフォを用いたプログラムを事例に，パターンによるフォールトの検出を行なう．事例に適用することでフォールト検出ができ，モデル検査のコスト軽減に有用であることを確認した．また，パターンによるフォールトの特定の考察および読み書き問題 [2][3]，共有変数による相互排除 [2] のパターンの定義について考察を行なった．

2　基本的なアイデア

基本的なアイデアを図 1 に示す．本稿では図 1 の太枠で囲まれた部分を研究対象とする．

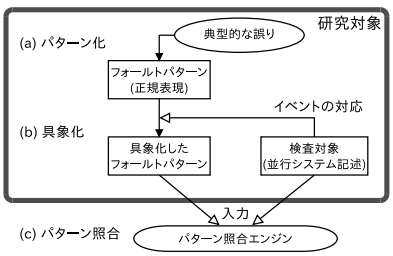

図 1　基本的なアイデア

(a) **パターン化**：既知の相互排除問題を分析し，典型的な誤りをフォールトパターンとして定義する．フォールトパターンは共有資源のプロセスとその共有資源の使用者であるプロセスから構成される．
(b) **具象化**：フォールトパターンに検査対象とするシステムのイベントを対応させる．
(c) **パターン照合**：フォールトパターンと検査対象をオートマトンに変換し，パターンが検査対象に含まれるかを調べる．

3　フォールトパターンの定義

生産者-消費者問題を題材としてフォールトパターンについて説明する．生産者-消費者問題は生産者プロセスが生産したデータを消費者プロセスが受け取ることを目的とする．ここでは有限バッファを介してデータをやりとりする．バッファに対してデータを渡すことを put イベント，データを受取ることを get イベントとする．

3.1　フォールトパターンの記述方法

生産者-消費者問題におけるフォールトパターンの記述方法を以下に示す．

```
shared_resource
  BUF = put for input, get for output
  path (BUF.put - BUF.get) ↑ n end
process P for Producer;
process C for Consumer;
fp(b:BUF,p:P,c:C) = 正規表現
```

共有資源 BUF への操作として put,get があることを表している．put は BUF に対する入力であり，get は出力である．

path,end で囲まれた部分は共有資源の正しい振舞いを表す順路式 [3] である．path (BUF.put-BUF.get) ↑ n end は順路式の数値型要素であり，BUF.put と BUF.get の選択的実行が $n \geq \#(\text{BUF.put}) - \#(\text{BUF.get}) \geq 0$ の範囲で繰り返されることを意味する．#(p) は p の実行回数を表し，BUF.e は BUF に対する操作 e の受信イベントを表す．n はバッファサイズを表す．

P は生産者のプロセス，C は消費者のプロセスである．BUF の誤った操作順序を正規表現で記述する．

3.2　生産者-消費者問題の誤り

誤りとして以下の 4 つを予測する．

生産者イベント誤り　生産者プロセスが get を送信
消費者イベント誤り　消費者プロセスが put を送信
バッファエンプティ　バッファが空のときに get を送信
バッファフル　バッファが満杯のときに put を送信

以降, 4 つの誤りのうち, 生産者イベント誤りとバッファエンプティのフォールトパターンについて説明する.

3.3 フォールトパターンの記述

共有資源の操作に着目した生産者イベント誤りのフォールトパターンを以下のように記述する.

```
fp_producer(b:BUF\{put},p:P) = p->b.get
```

p->b.get は生産者 p がバッファ b に対して get を送信するイベントを表す. S\{e} は共有資源 S のイベント e は着目しないことを表す.

共有資源への操作に着目したバッファエンプティのフォールトパターンを以下のように記述する.

```
fp_buffer_empty(b:BUF,p:P,c:C)
                        = BufferEmpty(n)
BufferEmpty(n) = BE^n(ε); c->b.get
BE(RE) = (p->b.put; RE; c->b.get)*
```

BufferEmpty(n) はバッファのサイズが n の場合のバッファが空になる振舞いを表している. BufferEmpty(n) は関数 BE を用いて記述されている. 関数は次のように記述する.

```
P^n(RE) = P^(n-1)(P(RE))   (n > 0)
P^0(RE) = RE
```

RE は正規表現を表す. 関数を用いることで正規表現の n 回の適用を記述することができる.

消費者イベント誤り, バッファフルについても同様にしてフォールトパターンを定義することができる.

これまでに示したフォールトパターンは共有資源の操作だけに着目した場合の誤りとなる振舞いを表している. システム全体の振舞いに対するフォールトパターンは対象以外のイベントを「other」で表し, 各イベントの前に「other*」を付加することで表すことができる.

4 フォールトパターンの事例への適用

セマフォを用いた生産者-消費者問題のプログラムを事例として用い, フォールトパターンによるフォールト検出を行なう. 生産者-消費者問題のプログラムの振舞いは CSP[1] で記述する.

セマフォを用いた生産者-消費者問題のプログラムは二つのセマフォ notEmpty, notFull を用いる. フォールトとして 3 種類のセマフォの使い方による誤りを想定した. 1) notEmpty に関する誤り, 2) notFull に関する誤り, 3) 二つのセマフォの組み合わせに関する誤り. 想定したフォールトの数は 24 個である.

結果として, 16 個のフォールトを検出できた. 検出できなかったフォールトはデッドロックのフォールトであり, それ以外のフォールトはすべて検出できた.

5 考察

5.1 フォールト特定

4 章で想定したフォールトをパターンと一致するイベント列で分類し, 分類を用いたフォールト箇所の特定について考察する. バッファエンプティパターンで検出された notEmpty に関する誤りを含むプログラムについて, パターンと一致するイベント列を分類した. 分類の一部を表 1 で示す. 表 1 を用いることで, 検出されたイベント列がフォールト箇所特定の指標になると考える.

表 1 イベント列の分類

想定したフォールト	イベント列
wait なし	get
wait を singal に	signal,get
初期値を 0 から N に	wait,get

デッドロックにいたるフォールトについても同様にイベント列を分類することで, 特定が可能になると考える.

5.2 生産者-消費者問題以外のフォールトパターン

本研究では生産者-消費者問題の他に, 読み書き問題のフォールトパターンと共有変数による相互排除問題のフォールトパターンを定義した.

読み書き問題のパターンの一つを以下に示す.

```
fp_write_write(s:S,w:W) =
 w->s.ws;other*;(w->s.we;other*;w->s.ws)*;
 other*;w->s.ws
```

ws が書き込み始めを, we が書き込み終わりを表し, 書き手 (w) が書き込み中に, 他の書き手が書き込みを行なう振舞いを表している.

共有変数による相互排除のパターンを以下に示す.

```
fp_mutual_exclusion(s:S,p:P,q:Q) =
 ((p->s.r; other*; q->s.r) +
  (q->s.r; other*; p->s.r)); other*;
 ((p->s.w; other*; q->s.w) +
  (q->s.w; other*; p->s.w))
```

二つのプロセス (p, q) が同時に読み込み (r) を行なったのち, 書き込み (w) を行なう振舞いを表している. これまでに定義したパターンはシステムの開始からのイベント列を検査し, パターンが現れることを検出することでフォールトを検出することができる. 共有変数による相互排除問題のパターンはシステム記述の途中にパターンが現れることを検出する必要があり, パターン照合のアルゴリズムの拡張を行なう必要がある.

6 おわりに

本研究ではパターンを用いたフォールト検出方法を提案し, フォールトパターンの定義を行なった.

今後の課題として並行システムの他の問題を分析してフォールトパターンを定義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] M. Ben-Ari, *Principles of Concurrent and Distributed Programming Second Edition*, Addison-Wesley, 2006.

[3] 土居範久, 相互排除問題, 岩波書店, 2011.