フォールトパターンを利用した実行前検査の研究

M2010MM018 神谷浩翔
指導教員：野呂昌満

1　はじめに

マルチコアプロセッサの出現や分散システムの普及に伴い，並行プログラミング技術の重要性が注目されている．並行システムの開発では，対象システムの仕様をモデル検査器 [3] や定理証明器 [5] を用いて検証することが重要である．モデル検査では，誤りを含むシステム記述を検証した場合，反例を出力する．反例はシステム記述の誤りを特定するうえで重要な情報であるが，実際には反例からシステム記述の誤りを特定することは困難な作業であり，検証コストが高くなる要因の一つとなっている [4].

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 (fault) を，パターンを用いて検出する手法を確立することである．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である [6]．フォールトをパターンとして表現することにより，構文レベルの解析でフォールトの検出を可能にする．典型的なフォールトを事前に検出し取り除くことで，検証コストの削減をめざす.

基本的なアイデアは，並行システムの既知の相互排除問題を分析し，典型的なフォールトを正規表現で定義することにより，フォールトの検出をパターン照合問題に帰着させることである．フォールトを含むシステム記述をフォールトパターンとして定義することにより，記述の書き間違いや使用方法の誤りなどを，システム記述の意味を考えることなく検出することができる．フォールトパターンの検出は，このようにすれば自動化が可能である.

本稿では，相互排除問題として生産者-消費者問題 [7] をとりあげ，CSP[2] 記述に対するフォールトパターンの検出方法を提示して，その有用性を議論する．セマフォの使い方の誤りを含んだシステムに対してフォールトパターンをもとにフォールトを検出できることを確認した．また，生産者-消費者問題をもとにフォールトパターンを分類し，際どい領域に対する相互排除問題と読み書き問題に対して適用可能性を考察した．事前に典型的なフォールトの検出が期待できるので，検証コストの削減につながると考える.

2　基本的なアイデア

フォールトパターンを用いたフォールト検出の基本的なアイデアを図 1 に示す．フォールトの検出は，検査対象の記述がフォールトパターンのインスタンスであるかを調べることに相当する.

a. パターン化

フォールトパターンは，共有資源が満たすべき振舞いに対して，典型的な誤りを予測・分類し，正規表現で記述する．パターンの記述は，共有資源のプロセスとその使用者であるプロセスから構成される.

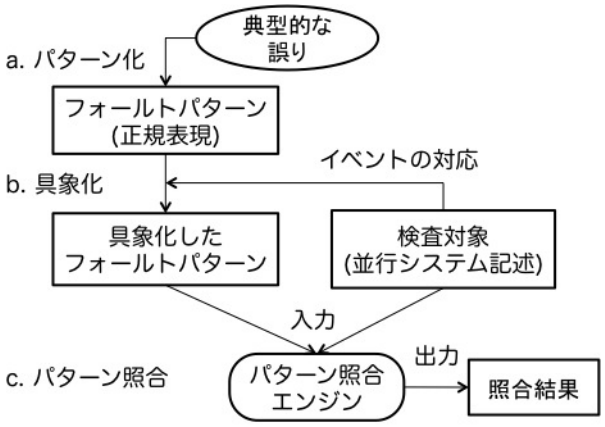

図 1　基本的なアイデア

b. 具象化

フォールトパターンのイベントと検査対象のイベントを対応させ，フォールトパターンを具象化する.

c. パターン照合

正規表現と検査対象の並行システム記述を有限オートマトンに変換し，正規表現を満たす検査対象のオートマトンのパスを照合することで検出する.

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を示す．生産者-消費者問題は，生産者プロセスが生成した情報を消費者プロセスが受け取る場合における同期問題である．ここでは，生産者と消費者は有限バッファを用いてデータの受渡しを行なう.

3.1　フォールトパターンの記述

はじめに共有資源の操作可能なイベントを定義する．生産者-消費者問題における共有資源のイベント定義は次のとおりである.

```
shared_resource
  B = get for output ,
      put for input
```

共有資源 B はバッファを表し，データを渡すことを put イベント，データを受け取ることを get イベントで表す.

次に共有資源の振舞いを，順路式 [8] で記述する.

```
path (B.put - B.get) ↑ n end;
```

path,end で囲まれた部分が数値型要素の順路式であり，n ≧#(B.put)-#(B.get) ≧ 0 の範囲で，B.put と B.get の選択実行が繰り返されることを意味する．#(p) は p の実

行回数を表す．この順路式は，常に B.get の実行回数より B.put の実行回数が多いことを表す．

次に共有資源のプロセスに対する，使用者のプロセスを定義する．

```
process P for Producer;
process C for Consumer;
```

生産者 (Producer) のプロセスを P，消費者 (Consumer) のプロセスを C として記述する．

最後にフォールトパターンを定義する．

```
fault_pattern(b:B, p:P, c:C)=正規表現
```

引数のプロセスをもとに，プロセス間のイベント送受信の順序を正規表現で記述する．

3.2 生産者-消費者問題のフォールトパターン

生産者-消費者問題のフォールトとして，次の 4 つの誤りを予測した．

- 生産者イベント誤り：生産者が get を送信
- 消費者イベント誤り：消費者が put を送信
- バッファエンプティ：バッファが空のときに消費者が get を送信
- バッファフル：バッファが満杯のときに生産者が put を送信

以降では,「生産者イベントの誤り」と「バッファエンプティ」についてフォールトパターンを定義する．

3.2.1 生産者イベント誤り

生産者イベント誤りは，生産者プロセスが共有資源であるバッファに対して get を送信する場合のフォールトである．フォールトパターンの記述は次のとおりである．

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

`p->b.get` は，生産者 p からバッファb へ get イベントを送信することを表す．`B\{put}`は隠蔽を表し，バッファB のイベント put は対象としないことを表す．

3.2.2 バッファエンプティ

バッファエンプティは，共有資源のバッファが空のときに，消費者プロセスがバッファに対して get を送信する場合のフォールトである．フォールトパターンの記述は次のとおりである．

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

正規表現は，バッファサイズ n の値が決まれば記述できる．正規表現を n 回適用する関数を定義することにより記述を行なう．正規表現を n 回適用する関数は次のとおりである．

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

`BE^n(ε)` は関数の n 回適用であり，ε は空列である．関数 BE は正規表現を引数として，正規表現を返す．

3.3 パターンの対象以外のイベント

これまでのフォールトパターンは，共有資源のイベントを対象とし，イベント生起順序を記述している．システム記述に対して共有資源以外のイベントを考慮し，記述する必要がある．共有資源以外のイベントを考慮したフォールトパターンを以下に示す．

```
other = Events - {put,get}
fp_producer(b:B\{put}, p:P) =
  other*; p->b.get
fp_buffer_empty(b:B, p:P, c:C) =
   BufferEmpty(n)
BufferEmpty(n) = BE^n(ε); other*;c->b.get
BE(RE) =
  (other*; p->b.put; RE; other*;c->b.get)*
```

Events は全てのイベントの集合を表す．対象イベント以外のイベントを表す other を用いてシステム全体に対するフォールトパターンを記述できる．対象イベント以外のイベントの挿入は，各イベントの前すべてに「other*」を挿入すればよく，自動で付加可能である．

4 パターン照合

フォールトパターンによって表される振舞いが，検査対象に含まれるか調べる．パターン照合によって，検査対象のフォールトを検出する．

4.1 パターン照合問題

パターン照合問題は,「正規表現によって表されるパターンに検査対象が含まれるか」を調べることである．検査対象とパターンはそれぞれ有限オートマトンに変換し，検査対象のパスに対してパターンが受理可能か調べる．

4.2 パターン照合のアルゴリズム

パターン照合のアルゴリズムの概要を図 2 に示す．

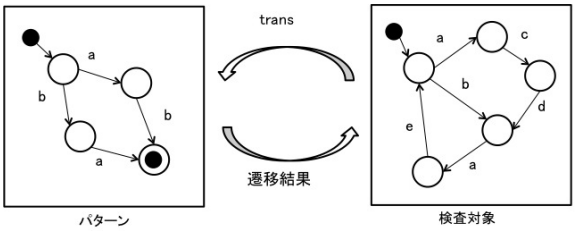

図 2 パターン照合のアルゴリズムの概要

アルゴリズムは，検査対象に対して深さ優先探索を適用し，探索時のイベントを入力として，パターンのオートマトンを遷移させ受理可能か調べる．パターンのオートマトンの遷移結果は，(1) 終了状態に遷移，(2) 次の状態に遷移，(3) 状態遷移に失敗とし，終了状態に遷移する場合がパターン照合の成功となる．パターン照合に成功した場合，終了状態までのパスを出力する．C 言語風のパターン照合のアルゴリズムを図 3 に示す．

(a) でパターンのオートマトンの遷移を行い，結果を取得している．バックトラックは (b) で行なっており，trace

```
#define N 辺をたどる回数の上限
Stack trace;
void visit(Vertex v,Vertex p){
  Event e; Vertex z; Result r;
  for(v を始点とする辺それぞれについて){
    if ( trace の中に辺が N 個あるとき) //(c)
      continue;
    e=辺のイベント; z=辺の行き先の頂点;
    r = trans(e, p);        // (a)
    if (r が次の状態に遷移){
      trace.push(辺);      // (b)
      visit(z, オートマトンの遷移先の状態);
      trace.pop();       // (b)
    }
    else if (r が終了状態に遷移)
      照合成功;
  }
  照合失敗;
}
```

図 3 パターン照合のアルゴリズム

にたどった辺を格納している．(c) では，二回以上同じ辺をたどった後に受理可能となるパターンを考慮して，たどる辺の回数の上限を判定している．

5 フォールトパターンの適用例

生産者-消費者問題の記述に対して誤りを想定し，3.2 節で定義した生産者-消費者問題のフォールトパターンによる検証を行なう．Ben-Ari の文献 [7] の生産者-消費者問題をもとにした正しい振舞いを図 4 に示す．

semaphore notEmpty ← 0 semaphore notFull ← N	
producer	consumer
loop forever	loop forever
p1: wait(notFull)	q1: wait(notEmpty)
p2: append	q2: take
p3: signal(notEmpty)	q3: signal(notFull)

図 4 生産者-消費者問題のアルゴリズム

生産者 (producer) と消費者 (consumer) は，バッファの追加 (append) と獲得 (take) を繰り返す．append と take の実行は，2 つのセマフォ変数 (notEmpty,notFull) を用いて制御する．notEmpty セマフォはバッファが空の時 take しないように，notFull セマフォはバッファが満杯の時 append しないように制御する．

図 4 のアルゴリズムをもとに重複しない誤りを想定し，3.2 節で定義した 4 つのフォールトパターンについて，バッファサイズ 2 として検証を行なった．ここでは想定した誤りの内，notEmpty セマフォ(初期値 0) に関する誤りと notFull セマフォ(初期値 N) に関する誤りに対するフォールトパターンのパターン照合結果を表している．(表 1)

表 1 において生産者イベント誤りと消費者イベント誤りの結果は省略している．デッドロックとなる項目以外はすべてフォールトパターンを検出した．

表 1 検証結果

セマフォ	想定した誤り	BE	BF	
notEmpty	(1)signal なし	×	×	DL
	(2)wait なし	○	×	
	(3)wait 二つ	×	×	DL
	(4)signal 二つ	○	×	
	(5)wait を signal に	○	×	
	(6)signal を wait に	×	×	DL
notFull	(7)signal なし	×	×	DL
	(8)wait なし	×	○	
	(9)wait 二つ	×	×	DL
	(10)signal 二つ	×	○	
	(11)wait を signal に	×	○	
	(12)signal を wait に	×	×	DL

(BE:バッファエンプティ,BF:バッファフル,DL:デッドロック)

6 考察

6.1 フォールトパターンの分類

生産者-消費者問題のフォールトパターンを基に分類したフォールトパターンを図 5 に示す．

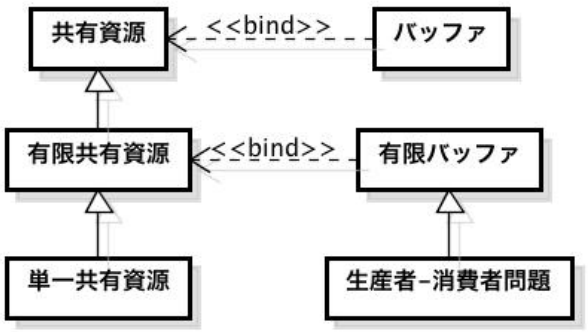

図 5 フォールトパターンの分類

フォールトパターンを汎化した概念として共有資源のフォールトパターンを定義する．

```
shared_resource
 S = inc for input, dec for output
 path (S.inc-S.dec) ↑ end
process P,Q for Process;
fp_shared_resource(s:S, p:P, p:Q)=
 zero(p->c.inc,q->c.dec);p->c.dec
```

共有資源に共通の概念として，共有資源のアクセス数を定義する．生産者-消費者問題ではバッファが情報を保持する数となる．バッファが情報を保持する数は，バッファへのイベントによって決まる．共有資源の数を扱う操作をイベント inc,dec として定義する．inc イベントは値を 1 増加させて，dec イベントは値を 1 減少させるイベントである．フォールトパターンは inc と dec の回数が同じ場合に dec を行なった場合を誤りとして表しており，バッファエンプティのフォールトパターンと同じである．zero

は，引数のイベントの実行回数が同じ場合を表している．

共有資源のフォールトパターンに上限を加え，有限共有資源のフォールトパターンを定義する．

```
path (S.inc-S.dec) ↑ n end
```

上記は有限共有資源の満たすべき振舞いの記述であり，上限を n として表現している．有限共有資源のフォールトパターンは，共有資源のフォールトパターンと次に示すフォールトパターンから表す．

```
fp_finite_counter(c:Counter, p:P, p:Q)=
 max(p->c.inc,q->c.dec,n);p->c.inc
```

この誤りは，上限を超える振舞いを行なう場合の誤りを表している．max は，引数のイベントの実行回数が上限 n にいたった場合を表している．

単一共有資源のフォールトパターンは，有限共有資源のフォールトパターンの上限が 1 の場合におけるフォールトパターンとなる．

共有資源のフォールトパターンを用いる事で，生産者-消費者問題を次のとおりに定義できる．

```
shared_resource
 Buffer=FiniteSharedResource
   {P<-Producer,Q<-Consumer}
   {inc<-put,dec<-get}
```

c<-c’ は名前の変更を表し，有限共有資源におけるプロセス P は生産者 (Producer)，プロセス Q は消費者 (Consumer) に対応し，イベント inc は生産 (put)，イベント dec は消費 (get) に対応する．生産者-消費者問題における有限バッファを有限共有資源のフォールトパターンを用いて記述する．

6.2 フォールトパターンの適用可能性

フォールトパターンの分類で定義した共有資源のフォールトパターンから，際どい領域に対する相互排除問題と読み書き問題のフォールトパターンについて適用し有用性を示す．

際どい領域 (critical section) に対する相互排除問題として，際どい領域の実行に対して相互排除していない場合を誤りとしてフォールトパターンを定義する．際どい領域の満たすべき振舞いを記述した順路式は以下となる．

```
path in;out end
```

際どい領域へ入る操作を in，際どい領域から出る操作を out として，際どい領域を実行するプロセスの数を単一とする．共有資源の実行プロセス数を単一共有資源のフォールトパターンを用いて定義する．

```
shared_resource
  CriticalSection=
    BinarySharedResource{inc<-in,dec<-out}
```

際どい領域を実行するプロセスの数が二つ以上の場合誤りとし，フォールトパターンで定義する．

読み書き問題は，同一の資源をアクセスする読み手と書き手のプロセスに対して，読み手同士は資源に同時にアクセスすること，書き手は排他的にアクセスすることを目的とし，相互排除問題を一般化したもので．読み書き問題の満たすべき振舞いを記述した順路式は次のとおりである．

```
path [#(S.rs)=#(S.re):
  (S.ws; S.we) + S.rs,(S.rs-S.re) ↑ n] end
```

資源 S に対して，読み手 R が読み込み開始と終了を行なうイベントを rs,re，書き手 W が書き込み開始と終了を行なうイベントを ws,we で表す．フォールトパターンは読み込み開始と終了を共有資源のフォールトパターンを用いてすることができる．

```
shared_resource
 S=SharedResource{P<-R,Q<-R}{inc<-rs,dec<-re}
```

読み書き問題では共有資源のフォールトパターンに加え，以下のフォールトパターンを定義する．

```
fp_writer_exception(s:S,r:R,w:W)=
 not_zero(r->s.rs,r->s.re,n);w->s.ws
```

このフォールトパターンは，読み込み中に書き込み開始を行なう誤りを表している．not_zero は引数のイベントの実行回数が同じではない場合を表している．fp_writer_exception を定義することで読み書き問題のフォールトパターンを定義できる．

7 おわりに

本研究では生産者-消費者問題のフォールトパターンを定義し，フォールトパターンの適用例をもとに有用性を示した．今後の課題として，既知の相互排除問題におけるフォールトパターンを定義し，オートマトンの変換ツールとパターン照合ツールの作成による自動化があげられる．

参考文献

[1] A. W. Roscoe, *The Theory and Practice of Concurrency,* Prentice-Hall, 1997.

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

[3] Formal Systems (Europe), “Formal Systems (Europe) Ltd, ” http://www.fsel.com/, 2010.

[4] 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 High-Assurance Systems Engineering, vol.3, no.4, pp74-83, 2010.

[5] Y. Isobe, and M. Roggenbach, *CSP-Prover - a Proof Tool for the Verification of Scalable Concurrent Systems,* Information and Media Technologies, vol.5, no.1, pp32-39, 2010.

[6] I. Sommerville, *Software Engineering,* Addison-Wesley, 2007.

[7] M. Ben-Ari, *Principle of Concurrent and Distributed Programming Second Edition,* Addison-Wesley, 2006.

[8] 土居範久，相互排除問題，岩波書店，2011.