パターンを用いたフォールト検出支援に関する研究

2007MI252 上田 仁　　2008MI104 川瀬 進吾
指導教員　張 漢明

1　はじめに

並行システムの振舞いを検査する手法としてモデル検査がある．モデル検査はシステムが満たすべき性質の真偽を自動で判定する．判定結果が偽であった場合，反例を出力する．反例とはシステムが満たすべき性質に違反するまでのイベント列である．反例をもとにフォールトを特定しシステムを修正する．反例はフォールトを特定する際の重要な情報であるが実際には反例からのフォールト特定は困難である．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 [2] である．

本研究の目的は並行システム記述における典型的なフォールトのパターンを用いた検出を自動化することである．フォールトとなる振舞いをフォールトパターンとして定義することで構文レベルの解析でフォールトを検出可能にする．フォールトの検出をツールで自動化し，典型的なフォールトを事前に取り除くことでフォールト特定にかかるコストを削減する．

本研究のアイデアは，フォールト検出を正規表現を用いることでパターン照合問題に帰着することである．パターン照合ではフォールトパターンの正規表現を有限オートマトンに，検査対象のシステム記述を有向グラフにそれぞれ変換する．変換した検査対象の有向グラフを辿りながらパターンの有限オートマトンが受理状態に達するパスを検出することでパターン照合を行なう．

本稿では検査対象のシステムの振舞いをプロセス代数である CSP[1] で記述した．有向グラフと有限オートマトン間で照合を行なうツールを試作した．事例にツールを適用し，フォールトパターンの検出を行なうことで作成したツールの妥当性を確認した．また，パターン照合のアルゴリズム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．図 1 のうち太枠で囲まれた範囲が本研究の対象である．

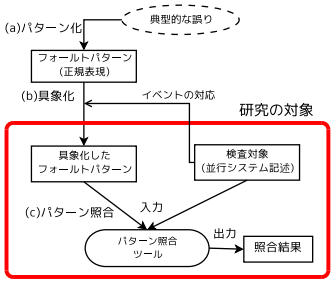

図 1　基本的なアイデア

(a) **パターン化**: 典型的な誤りを分析し，共有資源の操作に対する誤った操作をフォールトパターンとして正規表現で記述する．

(b) **具象化**: 定義したフォールトパターンに検査対象の並行システム記述のイベントを対応をさせる．

(c) **パターン照合**: 検査対象のシステムにフォールトパターンが合致する振舞いがあるかを調べる．

パターン照合を行なうために検査対象のシステム記述を有向グラフに，フォールトパターンを有限オートマトンにそれぞれ変換する．有向グラフを辿って有限オートマトンが最終状態に達する有向グラフのパスを検出することでパターン照合を行なう．

3　フォールトパターンの定義

フォールトパターンの記述方法を説明し，生産者-消費者問題のフォールトパターンを定義する．

3.1　フォールトパターンの記述方法

共有資源への操作に対する誤りをフォールトパターンとし，正規表現で記述する．

共有資源の操作の定義を以下に示す．

```
shared_resource
  SR = in for input ,
       out for output
  [共有資源への正しい操作（順路式）]
```

SR は共有資源を表し，in は共有資源に対する入力のイベントを，out は出力のイベントをそれぞれ表す．共有資源への正しい操作は順路式 [4] を用いて記述する．順路式は正規表現に加えて実行回数に関する制約を記述できる．

次に，フォールトパターンの記述方法を以下に示す．

```
process P1 for User_1;
process P2 for User_2;
    ...
process Pn for User_n;
fp(sr:SR, p1:P1, p2:P2, ... , pn:Pn) =
  [フォールトパターン（正規表現）]
```

P1, P2, ..., Pn は共有資源の使用者 (User_1, User_2, ..., User_n) のプロセスを表す．fp(sr:SR, p1:P1, p2:P2, ... , pn:Pn) = [フォールトパターン (正規表現)] はフォールトパターンを表す．引数のプロセスでフォールトパターンを記述する．

正規表現と順路式の構文とそれぞれの意味を以下に示す．

正規表現

`p;q` ：**逐次実行**
`p+q` ：**排他選択**

p* ：0 回以上の繰り返し

順路式

path RE end：正規表現 RE の 0 回以上の繰り返し
path (p-q) ↑ n end：実行回数の制限 n>=#(p)-#(q)>=0
#(p)，#(q) は p および q の実行回数

3.2 生産者-消費者問題の誤り

生産者-消費者問題で想定するフォールトを説明する．生産者-消費者問題は生産者が生産した情報を消費者に渡し，消費者は消費する情報を生産者から受け取る．生産者と消費者の情報は有限バッファを介して通信する．生産者が情報を渡すことを put イベントで表し，消費者が情報を受け取ることを get イベントで表す．有限バッファを用いた生産者-消費者問題で想定する誤りを以下に示す．

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

3.3 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パターンについてバッファエンプティを例に説明する．生産者-消費者問題のフォールトパターンは，生産者プロセス (P) と消費者プロセス (C)，および，有限バッファプロセス (B) から構成される．バッファが持つ put イベントと get イベントに着目する．

バッファエンプティはバッファが空の状態で消費者プロセスが get イベントを送信する誤りである．バッファエンプティのフォールトパターンを以下に示す．

```
fp_buf_empty(buf:B, prod:P, cons:C) =
    BufferEmpty(n); cons->buf.get
```

ただし，

```
BufferEmpty(n) = BE^n(ε)
BE(RE) = (prod->buf.put; RE; prod->buf.get)*
```

buf.put，buf.get はそれぞれ buf への put イベントの送信，get イベントの送信を表している．n はバッファの大きさを表し，get の送信の数は put の送信の数を越えてはいけない．

BufferEmpty(n) は大きさが n のバッファが空になる振舞いを表している．ここで，BufferEmpty(n) は関数 BE を用いて記述している．BE は正規表現を引数として，正規表現を返す関数である．BE^n は関数 BE の n 回適用を表している．ε は空列である．

4 フォールトパターンの変換

フォールトパターンを正規表現から決定性有限オートマトンへ変換する方法を説明する．正規表現で記述したフォールトパターンを非決定性有限オートマトンへ変換し，非決定性有限オートマトンから決定性有限オートマトンへ変換する．

正規表現には「逐次実行」「排他選択」「繰り返し」の構文がある．正規表現をアルゴリズム [3] にもとづいて非決定性有限オートマトンに変換する．

P，Q を正規表現，N(P)，N(Q) をそれぞれ P，Q に対応する非決定性有限オートマトンとした場合の変換を以下に示す．

逐次実行 逐次実行は演算子「 ; 」で表す．
例として P;Q は図 2 に変換できる．

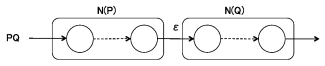

図 2 P;Q の非決定性有限オートマトンへの変換

排他選択 排他選択は演算子「 + 」で表す．
例として P+Q は図 3 に変換できる．

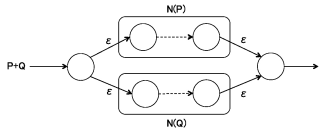

図 3 P+Q の非決定性有限オートマトンへの変換

繰り返し 繰り返しは演算子「 * 」で表す．
例として P* は図 4 に変換できる．

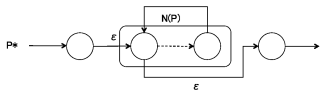

図 4 P* の非決定性有限オートマトンへの変換

非決定性有限オートマトンから決定性有限オートマトンへの変換はアルゴリズム [3] を用いて自動化できる．非決定性有限オートマトンから決定性有限オートマトンへの変換のアルゴリズムは省略する．

5 並行システム記述の変換

検査対象の CSP 記述から有向グラフへ変換する方法を説明する．CSP で記述した並行プロセスを逐次プロセスに変換し，逐次プロセスを有向グラフに変換する．

CSP には並行プロセスを逐次プロセスに変換する法則 [1] がある．並行プロセスを逐次プロセスに変換することで有向グラフに一意に変換することができる．CSP ではインターリーブや同期を用いて並行システムの振舞いを表現する．インターリーブと同期の逐次プロセスへの変換について説明する．

- インタリーブ
  インタリーブは以下の法則で逐次プロセスに変換する．

```
(x -> P)|||(y -> Q) = (x -> (P|||(y -> Q))
                    []  y -> ((x -> P)|||Q))
```

  例として以下のプロセス P とプロセス Q のインタリーブを逐次変換する．

```
P = a -> b -> STOP
Q = c -> STOP

P|||Q = a -> b -> c -> STOP
      [] a -> c -> b -> STOP
      [] c -> a -> b -> STOP
```

  インタリーブとは複数のプロセスが各プロセスの実行を任意の順番で実行することである．
- 同期
  同期は以下の法則で逐次プロセスに変換する．

  プロセス P，Q **はイベント** c，d **で同期する．**

```
(c -> P)||(c -> Q) = c -> (P||Q)
(c -> P)||(d -> Q) = STOP   (c ≠ d)
(a -> P)||(c -> Q) = a -> (P||(c -> Q))
```

  例としてイベント a で同期するプロセス P とプロセス Q の振舞いを逐次変換する．

```
P = a -> b -> STOP
Q = a -> c -> STOP
P||Q = a -> b -> c -> STOP
     [] a -> c -> b -> STOP
```

同期とは複数のプロセスに共通するイベントが存在する場合に全てのプロセスでそのイベントが実行可能になるまで待つことである．

6 パターン照合

パターン照合問題を定義し，パターン照合のアルゴリズムについて説明する．

6.1 パターン照合問題

本研究はパターン照合問題を，「有限オートマトンの最終状態を含む，有向グラフのパスの検出」と定義する．検査対象のシステム記述とフォールトパターンをパターン照合の対象とする．

6.2 パターンの照合アルゴリズム

パターン照合のアルゴリズムの概要を図 5 に示す．

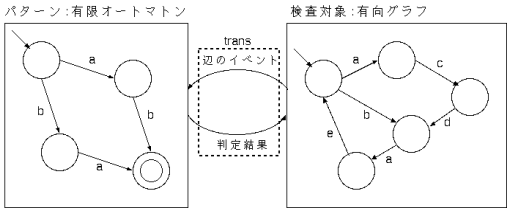

図 5　パターン照合のアルゴリズム

パターン照合は，検査対象の有向グラフを深さ優先探索 [3] で辿りながら有向グラフが次に辿る辺のイベントでフォールトパターンの有限オートマトンの状態が遷移可能か調べることで行なう．遷移可能かの判定結果は (1) 最終状態に遷移，(2) 次の状態に遷移，(3) 状態遷移に失敗の 3 種類である．判定結果が (1) の場合はパターン照合成功とする．(2) の場合は有向グラフの次の頂点を探索する．(3) の場合は別の辺を調べる．

パターン照合のアルゴリズムを C 言語風にした疑似コードで以下に示す．

```
void visit(vindex v,pindex p){
  event e; vindex z; result r;
  if(vertex[v].state == visited) 照合失敗;
  vertex[v].state = visited;
  for(v を始点とする辺それぞれについて){
    e = 辺のイベント; z = 辺の行き先の頂点;
    r = trans(e,p);
    if(r が最終状態に遷移) 照合成功;
    else if(r が次の状態に遷移)
      visit(z,r におけるパターンの遷移先の状態);
    else 照合失敗;
  }
  vertex[v].state = unvisited;
}
```

v が有向グラフの頂点，p が有限オートマトンの状態を表す．有向グラフの一度辿った頂点は visited にする．v が visited の場合は照合失敗とすることで同じ頂点を再度探索しないようにする．trans 関数によって v を始点とする辺のイベントで p が遷移可能か判定する．判定結果を返り値として r に代入する．v，p の遷移先を引数とした visit の再帰によって深さ優先探索を実現する．バックトラックの際に v を unvisited にすることで再度探索できるようにする．

7 考察

パターン照合アルゴリズムについて評価と考察を行なった．

7.1 パターン照合アルゴリズムの妥当性

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，セマフォの使い方の誤りを事例に混入させ，混入させた誤りをパターン照合のアルゴリズムを用いて検出した．検査対象のプログラムとして 16 個の誤りを混入させた事例を予測し，照合アルゴリズムを適用し検証した．

検証した結果，16 個全ての事例から期待するフォールトパターンを検出できた．この結果から照合アルゴリズムは妥当であることが確認できたと考える．

7.2 ループの扱い

6 章のアルゴリズムでは，一度辿った辺を辿ることはしない．しかしフォールトパターンはループを 2 回以上辿らないと検出できない場合があり，ループを考慮する必要がある．ループを辿るアルゴリズムについて考察する．

ループを辿るアルゴリズムとしてループに許容回数を設定することを考える．辿る辺をループした回数が許容回数を越えた場合に照合失敗とする．ループ許容回数を加えた場合のパターン照合アルゴリズムを以下に示す．

```
#define N ループ許容回数
Stack trace;
void visit(vindex v,pindex p){
  event e; vindex 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) 照合失敗;//(a)
    e = 辺のイベント; z = 辺の行き先の頂点;
    r = trans(e,p);
    if(r が最終状態に遷移) 照合成功;
    else if(r が次の状態に遷移){
      trace.push(辺);
      visit(z,r におけるパターンの遷移先の頂点);
      trace.pop();
    }
    else 照合失敗;
  }
}
```

アルゴリズムの (a) においてループの許容回数を確認している．trace にはパスを保存する．パスの中に含まれているこれから調べる辺の数がループ許容回数だった場合，照合失敗とする．

7.3 共有変数に対する相互排除問題

共有変数に対する相互排除問題とは共有変数を複数のプロセスが同時に読み書きすることによって起こる問題のことである．アルゴリズムを図 6 に示す．

int common ← 0	
Process p	Process q
int x	int y
loop forever	loop forever
p1 : x ← common	q1: y ← common
p2 : common ← x + 1	q2: common ← y + 1

図 6　1 変数に対する読み書きのアルゴリズム

これはふたつのプロセス p，q が共有変数 common の値を読み込み，読み込んだ値に 1 を加えて common の値を書き換えるアルゴリズムである．p1，q1 は common の値の読み込み (read) であり p2，q2 は common の値の書き換え (write) を表している．フォールトは図 6 の p1，p2 と q1，q2 の順序が同時にある場合である．p と q が同時に common の値を読み込んだ後，common の値を書き換える．

フォールトパターンを以下に示す．

```
((p->common.read;q->common.read) +
  (q->common.read;p->common.read));
 ((p->common.write;q->common.write) +
  (q->common.write;p->common.write))
```

上記のフォールトパターンの検出を考える．6 章で提案したパターン照合アルゴリズムは検査対象の有向グラフの先頭の頂点を始点としてパターン照合を行なっているが，このフォールトパターンは検査対象の先頭ではなく途中に含まれている場合がある．その場合，先頭からの照合ではフォールトパターンを検出することができない．検査対象の途中に含まれているフォールトパターンを検出するには，先頭の頂点以外の頂点を始点として照合アルゴリズムを適用する必要がある．

深さ優先探索で有向グラフを辿りながら，全ての頂点をそれぞれ始点としてパターン照合を行なうことを考えた．アルゴリズムを以下に示す．

```
void v_start(vindex start_v,pindex p_init){
  vindex next_v;
  if(vertex[start_v].check == checked) 探索失敗;
  vertex[start_v].check = checked;
  visit(start_v,p_init);
  for(start_v のもつ辺それぞれについて){
    next_v = 辺の行き先の頂点;
    v_start(next_v,p_init);
  }
}
```

v_start 関数は照合の始点をきめる関数であり，visit 関数は 6 章で提案した照合を行なう関数である．start_v は始点となる有向グラフの頂点を表す．p_init はパターンの有限オートマトンの先頭の状態を表す．一度始点にした頂点は checked とし，start_v が checked の場合には探索失敗としてバックトラックする．start_v と p_init を引数とした visit 関数によって start_v を始点としたパターン照合を行なう．next_v と p_init を引数とした v_start 関数の再帰によって深さ優先探索を実現する．深さ優先探索によって全ての頂点を辿り，各頂点でパターン照合を行なうことで検査対象の途中に含まれているフォールトの検出を可能にする．

8 おわりに

本研究ではパターン照合のアルゴリズムを提案し，有向グラフと有限オートマトン間で照合を行なうツールを試作した．今後の課題として変換アルゴリズムをもとに検査対象の並行システム記述の有向グラフへの変換とフォールトパターンの正規表現の有限オートマトンへの変換をツールに実装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] I. Sommerville, *Software Engineering*, Addison-Wesley, 2007.

[3] 石畑清, アルゴリズムとデータ構造, 岩波書店, 1989.

[4] 土居範久,“ 順路式, ”情報処理, vol.19, no.8, pp.779-787, Aug. 1978.