フォールトのパターン化によるモデル検査支援に関する研究

M2012MM014 川瀬進吾
指導教員：野呂昌満

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し，判定結果が偽であった場合に反例を出力する．反例とはシステムが満たすべき性質に違反するイベントの生起系列である．一般に反例を解析することでフォールトの特定を行うが，反例解析によるフォールトの特定は一般に困難な作業であり，検証の高コストの要因のひとつである．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．[2]

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を自動的に検出する手法を提示することである．プログラミングにおけるコードインスペクションツールのように，並行システム記述の典型的なフォールトをモデル検査による検証の前に指摘することで，検証コストの削減を目指す．

本研究の基本的なアイデアは，フォールトの検出を正規表現による有向グラフのパス照合問題に帰着することである．典型的なフォールトをフォールトパターンとして，着目するイベントに対してフォールトを特定するための部分列を正規表現で定義する．並行システムに特有な問題を順路式 [6] を用いて定式化し，順路式からフォールトを導出する方法を定義して，典型的なフォールトパターンを提示する．順路式は正規表現の演算子に加えて，数値型要素や条件型要素などを記述することができる．典型的なフォールトをフォールトパターンとして定義するために，過去の研究 [5] や並行プログラミングに関する論文 [3][7] から既存の誤りの分析を行う．

本研究の成果として順路式における正規表現の演算子，数値型要素，条件型要素に対するフォールトを正規表現で導出する導出関数を定義した．フォールトパターンの定義を行うために STM 間の通信におけるフォールトおよび、一般的な同期問題について分析を行い，典型的なフォールトパターンを提示した．また，実例に対してフォールトパターンを適用し，有用性の確認を行った．

2　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 (以下，STM) の集合として捉える．各 STM はそれぞれひとつずつ有限長のキューを持ち，各 STM はキューを介した非同期通信を行うものとする．本研究における計算モデルを図 1 に示す．

イベント送信: アクション実行時に STM を指定し，イベントを送信する．送信先のキューが満杯の際には空きができるまで待機する．

イベント受信: キューの最後尾にイベントを追加する．

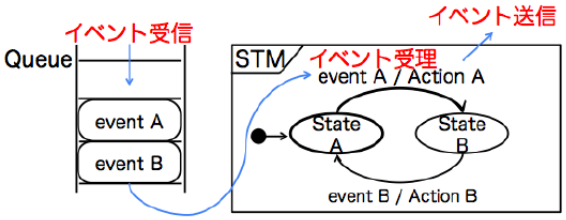

図 1　計算モデル

イベント受理: キューを先頭から走査し，受理可能なイベントを取り出す．

3　基本的なアイデア

モデル検査では仕様に違反しているという情報しか得ることができない．モデル検査による誤りの検出を図 2 に示す．

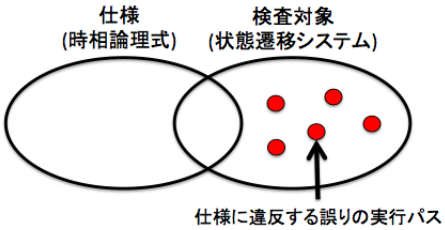

図 2　モデル検査によるフォールト検出

本研究の基本的なアイデアはフォールトの検出を正規表現による有向グラフのパス照合問題に帰着することである．典型的なフォールトを定式化し，効率的なフォールト検出手法を定義する．

3.1　フォールトパターン

本研究では並行システム記述における典型的なフォールト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着目するイベントに対してフォールトを特定する部分列として正規表現で記述する．並行システムに特有な問題を順路式を用いて定式化し，順路式からフォールトを正規表現で導出する．フォールトパターンの概要を図 3 に示す．

3.2　フォールトの導出

本研究では順路式からフォールトを正規表現で導出する．順路式は正規表現を拡張したもので操作の実行順序に加えて実行回数の制約や実行条件を記述することができる．

順路式の各演算子に対するフォールトを正規表現で導出する関数（以下，導出関数）を定義することでフォールトを提示する．

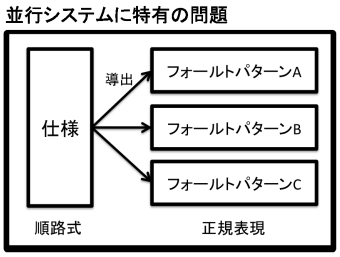

図 3 フォールトパターンの概要

3.3 フォールトの検出

検査対象の並行システム記述とフォールトパターンとでパス照合を行うことでフォールトの検出を行う．ここで，検査対象となる並行システムの振舞いはプロセス代数である CSP[1] で記述し，根付き有向グラフに変換できることを前提とする．本研究におけるフォールト検出の枠組みを図 4 に示す．

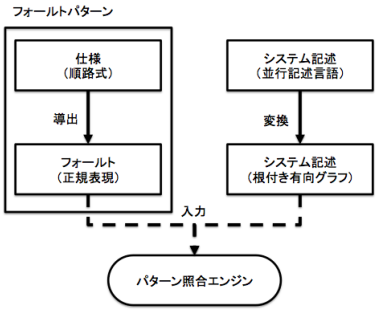

図 4 フォールト検出の枠組み

3.3.1 パターン照合エンジン

検査対象のシステムの根付き有向グラフと仕様から導出したフォールトの正規表現とでパターン照合を行う．パス照合は既存の研究の手法 [8] を利用する．

検出したイベントの生起系列に対して付加情報を追加することで，どのような誤りか明示する．

4 フォールトの導出

順路式の各演算子に対するフォールトを導出する関数を定義する．導出関数には仕様を記述した順路式を入力として与え，仕様に対するフォールトを正規表現で出力させる．

4.1 順路式

本研究では仕様を順路式で記述する．順路式は正規表現を拡張したもので，正規表現の記述に加えて数値型要素，条件型要素，並行実行などを記述することができる．順路式の各演算子に対して導出関数を定義する．

ここで，対象とするイベントのアルファベットを α と定義する．システムがこれ以上進行しないことを表す特別なイベントとして Stop を定義する．

4.1.1 演算子

本研究で扱う正規表現の演算子を以下に示す．なお，e はイベントを表し，P1 と P2 および P はそれぞれ正規表現の式を表している．

イベント生起: `P = e`

逐次: `P = P1;P2`

選択: `P = P1|P2`

0 回以上の繰り返し: `P = P*`

4.2 導出関数

まず，イベント生起とシステムの停止に対する導出関数を以下に示す．

- `V( e ) = Stop | OTHER( { e } )`
- `V( Stop ) = CHOICE( α )`

関数 V は導出関数である．V(e) はイベント e が生起するという仕様に対するフォールトを導出する．この場合，フォールトはシステムが停止する，もしくはイベント e 以外が生起するということを導出する．同様に，V(Stop) はシステムが停止するという仕様に違反するフォールトを導出する．この場合，フォールトはシステムのイベントのうち，いずれかのイベントが生起することである．

次に正規表現の各式に対する導出関数を以下に示す．

- `V( P1;P2 ) = V( P1 ) | P1;V( P2 )`
- `V( P1|P2 ) = (V( P1 ) - P2) | (V( P2 ) - P1)`
- `V( P* ) = P*;V( P )`

4.3 補助関数

導出関数を補助する関数として補助関数を以下のように定義する．ここで，S はイベントの集合を表す．

- `CHOICE( { e } ) = e`
- `CHOICE( S ) = e <- S @ e | CHOICE( S\{ e } )`
- `OTHER( S ) = CHOICE( α - S )`

関数 CHOICE は引数のイベントの集合のうちのどれかのイベントが生起することを表し，関数 OTHER は引数のイベントの集合以外のイベントが生起することを表している．

4.4 フォールトの導出例

上記の導出関数を用いたフォールトの導出の例を示す．仕様を以下のように定義する．

```
Spec = ( e1;e2 )*
```

Spec はイベント e1，e2 がこの順番で交互に生起し続けるという仕様である．Spec を導出関数 V に引数として与えると以下の様になる．

```
V( Spec ) = V( ( e1;e2 )* )
  = ( e1;e2 )*;V( e1;e2 )
  = ( e1;e2 )*;( V( e1 ) | e1;V( e2 ) )
  = ( e1;e2 )*;( Stop | e2 |
                    e1;( Stop | e1 ) )
```

```
= ( e1;e2 )*;( Stop | e2 |
              ( e1;Stop ) | ( e1;e1 ) )
```

V(Spec) の結果から Spec のフォールトとして「システムが停止する」,「e1 が生起する前に，e2 が生起する」,「e1 が生起した後，システムが停止する」,「e1 が生起した後，再び e1 が生起する」が導出できた．

4.5 数値型要素

順路式は正規表現の記述に加えて，操作の実行回数の制約や実行条件を記述することができる．例えば以下の式はイベントの生起回数の制約を記述している．これを図示したものが図 5 である．

```
Spec = ( p - q ) ↑ n
```

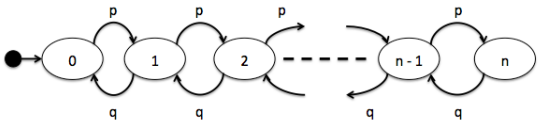

図 5 数値型要素の例

`#( e )` をイベント e の生起回数とすると上記の式は `n >= #( p ) >= #( q ) >= 0` を常に満たすことを記述している．この仕様に対するフォールトは「q の生起回数が p の生起回数を越えること」と「p の生起回数と q の生起回数の差が n を越えること」である．このフォールトを記述すると以下になる．ここで `diff(p, q, i)` を `#(p) - #(q) = i` となる振舞いを表すとする．

```
V(Spec) = V(( p - q ) ↑ n)
        = diff(p, q, 0);q | diff(p, q, n);p
```

`diff(p, q, i)` を正規表現で記述することはできないが，n を固定することで記述することができる．

4.6 条件型要素

実行条件は以下のように記述する．

```
[d=i:P1, P2]
```

これは d=i のとき P1 を実行し，それ以外の場合は P2 を実行することを表す．この条件を `diff(p, q, d)` に追加すると以下になる．

```
[d=i:P1, P2] =
   diff(p, q, i);P1 |
   diff(p, q, {1..n} - i);P2
```

上記の仕様に対するフォールトは「d=1 のときの P1 のフォールト」と「d=1 以外のときの P2 のフォールト」であり記述すると以下になる．

```
V( [d=i:P1, P2] ) =
   diff(p, q, i);V(P1) |
   diff(p, q, {1..n} - i);V(P2)
```

5 フォールトパターン

5.1 型定義

既存の同期問題や並行計算モデルに特有の誤りを分析し，並行システムに特有なフォールトをフォールトパターンとして定義する．各問題において着目するイベントをパラメータ化して型として定義する．各パラメータに対して検査対象のシステムのイベントを与えることでインスタンス化する．

5.1.1 仕様

並行システムに特有な問題を順路式を用いて定式化する．順路式は並行システムの振舞いを記述するために正規表現を拡張したもので，正規表現の記述に加えて数値型要素や条件型要素，並行実行などを記述できる．

5.1.2 フォールトパターン

順路式からフォールトを導出関数を用いて導出し，典型的なフォールトパターンを定義する．フォールトパターンはフォールトを特定する部分列として正規表現で記述する．正規表現は既存のアルゴリズム [8] を用いて検出を行うことができる．

5.2 フォールトパターンの例

フォールトパターンの例として一般的な同期問題と非同期通信における計算モデル特有の誤りについて分析を行う．

- 一般的な同期問題
  - 相互排除
  - 生産者-消費者問題
  - 読み書き問題
- 計算モデル特有
  - 送信-受理
  - 要求-応答

分析した誤りのうち送信-受理について説明する．

5.2.1 送信-受理

送信-受理はイベントの送信と受理の関係を定義する．

5.2.2 着目するイベント

送信-受理において着目するイベントを以下に定義する．

- `e_send` : イベント e の送信
- `e_accept` : イベント e の受理

5.2.3 仕様

送信-受理の仕様は「イベントの送信」,「イベントの受理」がこの順番で繰り返されることである．送信-受理の仕様を順路式で記述すると以下になる．

```
Spec = ( e_send;e_accept )*
```

5.2.4 フォールト

導出関数を用いて送信-受理のフォールトを導出する．

```
V( Spec ) = V( ( e_send;e_accept )* )
  = ( e_send;e_accept )*;
    ( Stop | e_accept |
    ( e_send;Stop ) | ( e_send;e_send ) )
```

上記の結果が送信-受理のフォールトである．

6　フォールトの有用性に関する考察

実例に対してフォールトパターンを適用することで有用性について考察する．実例は過去の研究 [4] から抜粋した．フォールトパターンを適用した実際のフォールトを以下に示す．

- キュー満杯
- 競合

6.1　キュー満杯

キュー満杯は各 STM において処理できないイベントによってキューが満杯になり，イベントが受け取れずデッドロックになってしまうことである．キュー満杯について図 6 に示す．

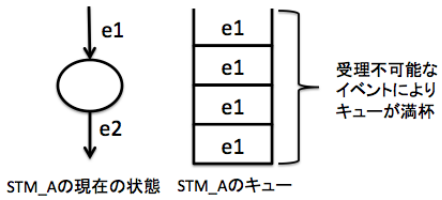

図 6　キュー満杯

図 6 において左が STM_A の現在の状態であり，受け取れるイベントが e2 であることを示している．e1 は STM_A が現在の状態に遷移するためのイベントであり，現在の状態では受け取れない．STM_A が現在の状態に遷移した後にも e1 がキューに送られ続けることでキューが満杯になり，e2 が受け取れなくなりデッドロックとなる．

フォールトパターン

キュー満杯によって引き起こされる誤りは,「キューの仕様に対する誤り」と「イベントの送信-受理に関する誤り」である．キューの仕様に対する誤りは数値型要素と同じであり，数値型要素のフォールトパターンで検出できた．送信-受理に関する誤りは送信-受理のフォールトパターンで検出できた．

6.2　競合

競合はひとつの STM に対して同時に複数のイベントが送信されることによって起こる誤りである．自動販売機システムを例にして図 7 に示す．

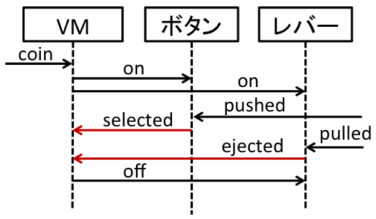

図 7　競合

ここでは VM の STM に着目し，自動販売機システムに対する返金と商品選択の同時入力を対象とする．また，coin を入金のイベント，selected を商品選択のイベント，ejected を返金のイベントとする．selected と ejected の同時生起に対して対策ができていないと競合が起こる．

フォールトパターン

競合によって引き起こされる誤りは「仕様に対する誤り」と処理されなかったイベントがキューに溜まることで起こる「キュー満杯」である．ここで，仕様は入金の後，商品選択か返金のいずれか一方だけが起こることである．仕様を順路式で以下に示す．

```
(s.coin;(s.selected | s.ejected))*
```

この式を導出関数に与えてフォールトを導出し，検出を行う．

```
V(Spec) = V((s.coin;(s.selected | s.ejected))*)
        = (s.coin;(s.selected | s.ejected))*;
          (Stop | s.selected | s.ejected |
                  (s.coin;Stop) | (s.coin;s.coin))
```

検出を行った結果，図 7 のパスを検出することができた．キュー満杯については 6.1 と同様に検出することができた．

6.1 と 6.2 の結果からフォールトパターンの有用性を確認できた．

7　おわりに

本稿では，順路式における正規表現の演算子および，数値型要素，条件型要素に対するフォールトを導出する導出関数を定義した．STM 間の通信におけるフォールトおよび、一般的な同期問題について分析を行い，フォールトパターンの定義を行った．また，実例に対してフォールトパターンを適用し，有用性の確認を行った．

今後の課題として実際に検証した事例からフォールトパターンの抽出を行うことが挙げられる．

参考文献

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

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

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

[4] 石原脩平，高野寛，山口隼平,“ 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，”南山大学情報理工学部ソフトウェア工学科 2012 年度卒業論文，2013.

[5] 稲垣尋紀,“ 並行システムにおけるフォールトのパターン化に関する研究，”南山大学大学院数理情報研究科 2012 年度修士論文，2013.

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

[7] 土居範久，相互排除問題，岩波書店，2011.

[8] 山内宏也,“ 並行システム記述に対するパスの照合の研究，” 南山大学大学院 2012 年度修士論文，2013.