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

M2011MM025 稲垣尋紀
指導教員：野呂昌満

1　はじめに

並行システムの振舞いを把握するには，モデル検査による検証が有用である [8]．モデル検査では，システムの状態を網羅的に走査し，満たすべき性質の真偽を自動的に判定する．検証結果が偽の場合は，反例を出力する．反例はシステム記述の誤りを特定するのに重要な情報であるが，反例から誤りを特定することは本質的に困難な作業であり，検証コストが高くなる要因の一つである．

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 (fault) をパターンを用いて検出することにより，フォールト特定作業を支援することである．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 [3]．モデル検査の前にフォールトを検出することで，検証全体にかかるコスト削減を目指す．

本研究のアイデアは，典型的なフォールトをパターン化し，フォールト検出を有向グラフにおけるパスの照合問題に帰着させることである．並行プログラミングに関する文献 [4][7] における正しいイベント生起順序（以下，振舞い仕様）を分析し，フォールトに成り得るイベント生起順序をパターンとして定義する．フォールトのパターン化により，並行システム記述の意味解析をすることなく構文レベルの解析によりフォールト検出ができる．

本稿では，フォールトパターンをアプリケーション依存のものと一般的なものに分類して提示し，その有用性を議論する．一般的なパターンとして，計算モデル固有のもの，典型的な同期問題のものを定義した．単純な自動販売機システムにパターンを適用することでフォールトを検出できることを確認した．フォールト特定に必要な情報やフォールトパターンの拡張について考察した．

2　基本的なアイデア

2.1　フォールトパターンを用いた検証のプロセス

パターンを用いた検証のプロセスを図 1 に示す．

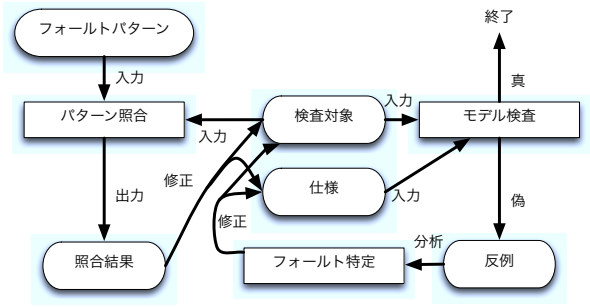

図 1　フォールトパターンを用いた検証のプロセス

本研究では，モデル検査の前にフォールトを検出することで，検証全体のコスト削減を目指す．

2.2　フォールト検出の枠組み

フォールト検出の枠組みを図 2 に示す．

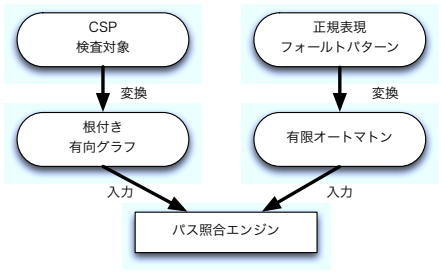

図 2　パターンによるフォールト検出の枠組み

検査対象（有向グラフ）とフォールトパターン（有限オートマトン）をパス照合することでフォールトを検出する．プロセス代数 CSP[1] で記述された検査対象を有向グラフに，正規表現で記述された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．

2.3　フォールトパターンの検出例

パターンを用いたフォールト検出を，単純な自動販売機システムを例に説明する．自動販売機の基本的な振舞いを図 3 に示す．自動販売機は，ボタン，および，タイマーの状態が有効か無効かどうかを制御している．

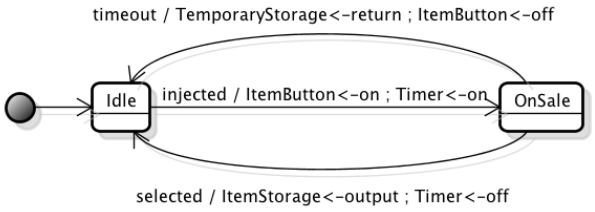

図 3　自動販売機の状態遷移機械

自動販売機はコインが投入されると，ボタンを有効 (ItemButton<-on) にしてタイマーを有効 (Timer<-on) にする．ボタンが押されたら (selected)，商品を排出してタイマーを無効にする (Timer<-off)．タイムアウトしたら (timeout)，コインを返金してボタンを無効にする (ItemButton<-off)．

入力を「コイン投入後，ボタン押下またはタイムアウトの繰り返し」，期待する出力を「商品排出またはコイン返金の繰り返し」としてモデル検査した結果は偽となった．

図 4 のようなイベント生起順序があるとボタン押下（selected）とタイムアウト（timeout）の競合が起こる場合がある．自動販売機がボタン押下を受理した後，タイマー無効（off）を送信する前にタイムアウトが送信される．自

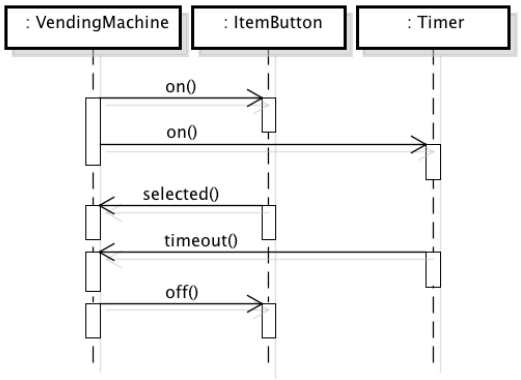

図 4　イベント競合

動販売機の待機状態 (Idle) でタイムアウトがキューに残ることで，後にデッドロックが発生する可能性がある．

フォールトパターンでは，イベント送信とイベント受理の関係に着目して「送信されたイベント受理がない」というフォールトを検出することができる．イベント受理がない場合，次のイベント送信まで受理されない可能性がある．イベント送信が二回続くことを，フォールトパターンとして以下のように定義する．

```
(send.timeout ; recv.timeout)* ;
  send.timeout ; send.timeout
```

タイムアウト送信 (send.timeout) と受理 (recv.timeout) に着目することで，フォールトを検出できる．

3　フォールトパターンの基本構成要素

フォールトパターンは，以下の要素から構成される．

- コンポーネント
- イベント
- 振舞い仕様記述
- フォールトパターン記述

フォールトに関係するコンポーネントとイベントに着目する．振舞い仕様は，正しいイベントの生起順序を正規表現を用いて記述する．フォールトパターンは，振舞い仕様を満たさない振舞いのうち，典型的なフォールトの可能性があるイベントの生起順序を記述する．

3.1　想定する計算モデル

本研究では，システムを並行に動作するコンポーネントの集合として捉え，状態遷移機械でモデル化する．コンポーネント間の通信は，キューを介して非同期に行なう．計算モデルの概要を図 5 に示す．

通信するイベントの単位は，送信，受信，受理がある．イベント送信は，コンポーネントを指定してイベントを送る．イベント受信は，キューの最後尾にイベントを追加する．イベント受理は，キューを先頭から走査して受理可能なイベントを取り出し，イベントに対応するアクションを実行する．本計算モデルでは，イベント送信と受信は同時と捉え，イベント送信と受理のみに着目する．

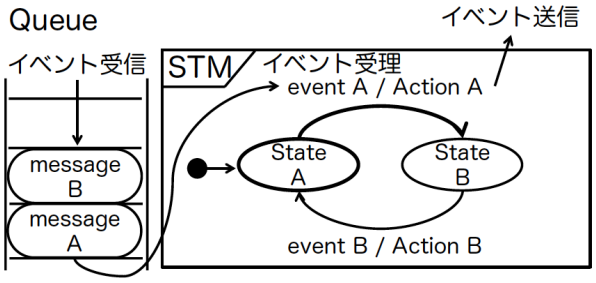

図 5　想定される計算モデル

3.2　振舞い記述

イベントの送信，受理は，送信側のコンポーネントを S，受理側のコンポーネントを R，および，イベントを event として，以下のように記述する．

```
R<-event@S    :  イベント送信（＝受信）
event@R       :  イベント受理
```

振舞いの記述に用いる正規表現を以下に示す．

```
p ; q         :  p の後に q を逐次実行
p + q         :  p もしくは q を排他選択
p*            :  p の 0 回以上の繰り返し
```

正規表現の拡張として順路式，イベント未生起，および，連結を以下に示す．

```
(p - q) ↑ n   :  n ≧ #(p) ≧ #(q) ≧ 0
e ; STOP      :  e の後にイベント生起がない
e ^ A.s       :  イベントとオートマトンを連結
A.s ^ A.s     :  オートマトン同士を連結
```

#(e) はイベントの生起回数を，A.s はオートマトンのある状態を表す．

4　フォールトパターンの定義

本研究では，フォールトパターンをアプリケーション依存と一般的なパターンに分類した．一般的なフォールトパターンとして，計算モデル固有，典型的な同期問題のパターンを定義する．フォールトパターンを分類した結果，順序型，バッファ型，割込み型の三つを提示する．

本稿では，順序型である送信-受理，バッファ型である生産者消費者問題，割込み型であるレースコンディションのフォールトパターンを提示する．

4.1　送信-受理（順序型）

計算モデルにおいてイベント送信に対してイベント受理がなければ，受理側のコンポーネントで想定しないイベントがキューに残る．このことはフォールトとして認識されず，後にイベントキューが詰まってデッドロックが起こる原因となる．

送信-受理は，順序型により定義する．振舞い仕様を以下に示す．

□着目するコンポーネント
　S（送信側），R（受理側）
□着目するイベント
　R<-e@S（イベント送信），e@R（イベント受理）

□振舞い仕様

```
order(R<-e@S, e@R) = (R<-e@S ; e@R)*
```

振舞い仕様を基に，以下のフォールトパターンを定義する．

■フォールトパターン

```
cont_first(R<-e@S, e@R) =
  order(R<-e@S, e@R) ; R<-e@S ; R<-e@S
no_second(R<-e@S, e@R) =
  order(R<-e@S, e@R) ; R<-e@S ; STOP
```

cont_first は，イベント送信が二回連続で生起することを表している．イベント送信に対してイベント受理されなければ，そのイベントは次のイベント送信まで受理されない可能性がある．ただし，イベントキューの長さは 2 以上が必要である．no_second は，イベント送信の後にイベント受理が生起されないことを表している．イベント送信に対するイベント受理がない場合はデッドロックが起こる可能性がある．

4.2 生産者-消費者問題（バッファ型）

生産者-消費者問題とは，情報を作成する生産者と情報を消費する消費者間の同期問題である．バッファを介して生産者が作成した情報を消費者に渡す．有限バッファ (Bounded Buffer) を想定する場合，バッファが満杯のときに生産者が情報を追加しようとするとフォールトとなる．

生産者-消費者問題は，バッファ型により定義する．振舞い仕様を以下に示す．

□着目するコンポーネント

B（Bounded Buffer），P（生産者），C（消費者）

□着目するイベント

B<-put@P（情報追加），B<-get@C（情報消費）

□振舞い仕様

```
buf(B<-put@P, B<-get@C) =
  (B<-put@P - B<-get@C) ↑ n
```

振舞い仕様を基に，以下のフォールトパターンを定義する．

■フォールトパターン

```
full(B<-put@P, buf(B<-put@P, B<-get@C).n) =
  B<-put@P ^ buf(B<-put@P, B<-get@C).n
```

full は，Bounded Buffer のオートマトンにおける状態 n と put 送信の連結を表している．

バッファ型は有限オートマトンにより記述する．

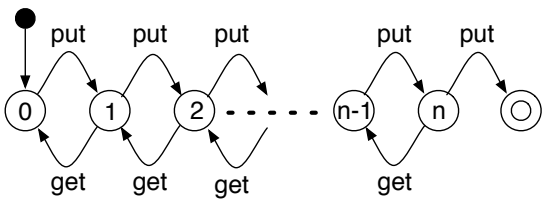

図 6 バッファ満杯

図 6 のバッファが n 状態の時に put イベントが送信されたことを表している．バッファが満杯のときに生産者が情報を追加しようとするとフォールトとなる．

4.3 レースコンディション（割込み型）

レースコンディションは，イベントの生起順序によって想定外の結果になる際に発生する．本研究では，データを保持するコンポーネントに対するデータの読み書きからフォールトをパターン化する．

レースコンディションは，割込み型により定義する．振舞い仕様を以下に示す．

□着目するコンポーネント

R（共有資源），A，B（利用者）

□着目するイベント

R<-r@A（読込み），R<-w@A，R<-w@B（書込み）

□振舞い仕様

```
no_interrupt(R<-r@A, R<-w@A, R<-w@B) =
  (R<-w@B + (R<-r@A ; R<-w@A))*
```

振舞い仕様を基に，以下のフォールトパターンを定義する．

■フォールトパターン

```
interrupt(R<-r@A, R<-w@A, R<-w@B) =
  no_interrupt(R<-r@A, R<-w@A, R<-w@B) ;
  R<-r@A ; R<-w@B
```

interrupt は利用者がデータを読込んでから書込むまでの間に，他の利用者による書込みが割込むことを表している．他の利用者による書込みが割込むと，仕様とシステムの振舞いとの整合性が損なわれる可能性がある．

5 考察

5.1 詳細なフォールト特定に必要な情報

単純な自動販売機システムを事例に，詳細なフォールト特定に必要な情報について考察する．イベント競合を検出するのに，送信-受理のフォールトパターンを用いる．R<-event@S にタイムアウト送信，event@R にタイムアウト受理を当てはめてパターン照合を行なった結果，照合結果は真となった．

「送信されたタイムアウトの受理がない」ことを特定することができたが「システム設計がイベント競合に未対応」であることまではわからない．図 4 のイベント生起順序を分析して，アプリケーション依存のフォールトパターンとしてイベント競合パターンを定義する．

振舞い仕様を以下に示す．

□振舞い仕様

```
no_rc(IB<-on@VM,T<-on@VM,VM<-s@IB,VM<-t@T) =
  (IB<-on@VM ; T<-on@VM ; VM<-s@IB + VM<-t@T)*
```

この振舞い仕様を基に，定義したフォールトパターンを以下に示す．

■フォールトパターン

```
rc(IB<-on@VM, T<-on@VM, VM<-s@IB, VM<-t@T) =
  IB<-on@VM ; T<-on@VM ;
  (VM<-s@IB ; VM<-t@T) + (VM<-t@T ; VM<-s@IB)
```

rc は，自動販売機 (VM) がボタン (IB) とタイマー (T) に on を送信した後，ボタンから商品選択 (s) が送信され，タイマー からタイムアウト (t) が送信されることを表している．このパターンを検出すればイベント競合の発生

がわかるが，パターンの照合結果が真だとしても，必ずしも間違いであるとは限らない．

システム設計がイベント競合に未対応であることを検出するのに，段階的にフォールトパターンを適用する．

1. イベント競合の有無を検出
2. 送信されたイベントの受理がないことを検出

アプリケーション依存フォールトパターンの後に一般的なフォールトパターンを適用することで，より詳細にフォールトを特定することができる．

5.2 フォールトパターンの拡張

フォールトパターン順序型の拡張について考察する．順序型は二つのイベント間における生起順序の関係を定義したが，着目するイベントの数を増やせるように拡張する．着目するイベントを増やすことで，何番目のイベントがフォールトかまでを特定することができる．

拡張の例として，三つのイベント間における生起順序の関係を定義する．イベント a, b, c に着目した際の振舞い仕様を以下に示す．

□振舞い仕様

```
order(a, b, c) = (a ; b ; c)*
```

振舞い仕様を基に定義したフォールトパターンを以下に示す．

■フォールトパターン

```
order(a, b, c) ;
  b + c + STOP +
  a;a + a;c + a;STOP +
  a;b;a + a;b;c + a;b;STOP
```

イベント二つからイベント三つへの拡張を一般化すると，以下のようになる．

```
振舞い仕様 ;
  一番目のイベント以外 +
  二番目のイベント以外 +
  三番目のイベント以外 +
  …
```

この拡張方法を用いれば，n 個のイベントに着目した順序型を定義することができる．

5.3 フォールトパターン抽出プロセスの妥当性

パターン抽出プロセスには，「発見 ⇒ 記述 ⇒ 共有 ⇒ 評価」のライフサイクルがある [5]．

「発見」は，ソフトウェア開発知識から繰り返し出現する問題と解決方法を特定することである．本研究では，並行プログラミングの知見から計算モデルにおけるコンポーネント間の通信，および，典型的な同期問題を題材とし，フォールトを特定する情報として，コンポーネント間のイベント生起順序に着目した．

「記述」は，発見したパターンを特定の形式に従って記述することである．本研究では，正規表現，および，順路式をはじめとする正規表現の拡張によりフォールトパターンを記述した．

「共有」は，記述したパターンをしかるべき場所に提出し公開することである．本研究では，本論文を基に本学名古屋キャンパスにおいて最終審査 (研究発表) を行った．

「評価」は，公開された品質をレビューなどによって評価・洗練することである．本研究では，レビューを受けて，イベントキュー満杯パターン，および，バッファ満杯パターンにおける有限バッファ(Bounded Buffer) の記述方法の再考を行なった．したがって，フォールトパターン抽出のプロセスが妥当である．

5.4 関連研究との比較

関連研究は，大きく分けて反例の可視化 [2] と自動解析 [6] がある．反例の可視化では，イベントの生起順序を図示することで反例の理解を支援している．しかし，システム記述が複雑になると反例の理解が難しくなる．本研究では，並行システム記述が複雑になっても，フォールトパターンによりフォールトを特定することができる．

反例の自動解析では，正例と反例を解析することで自動的に誤り箇所を支援している．しかし，ある種の誤りに特化した自動解析を行なっていることから，誤りの特定が限定的である．本研究では，正規表現により，様々なフォールトパターンを定義している．同一の照合アルゴリズムにより様々なフォールトを検出することができる．

6 おわりに

本研究では，計算モデル，典型的な同期問題のフォールトパターンを定義した．イベント競合の特定に必要な情報．順序型の拡張について考察した．今後の課題として，大規模システムに対するフォールトパターンの適用，フォールト特定に必要な情報について検討する必要がある．

参考文献

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

[2] C. Artho, K. Havelund, and, S. Honiden, *Visualization of Concurrent Program Executions,* 31st COMPSAC 2007, pp.541 - 546, 2007.

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

[4] R. H. B. Netzer, B. P. Miller, *What are race conditions?: Some issues and formalizations,* LOPLAS, Vol.1, pp.74-88, 1992.

[5] 鷲崎弘宜，丸山勝久，山本里枝子，久保淳人，トップエスイー実践講座 2 ソフトウェアパターン パターン指向の実践ソフトウェア開発，近代科学社，2007.

[6] 陳 適，青木 利晃，”モデル検査ツールにより出力された反例に基づく誤り特定に関する研究，” 情報処理学会研究報告，vol.2012-SE-177，no.6，pp.1-8，2012.

[7] 土居範久，相互排除問題，岩波書店，2011.

[8] 中島震，“ モデル検査法のソフトウェアデザイン検証への応用，”コンピュータソフトウェア，vol.23，no.2，pp.72-86，2006.