フォールトパターン検出ツールの実現に関する研究

2009SE031 二村太郎　2009SE140 熊崎将成　2009SE170 水野大
指導教員：張漢明

1　はじめに

並行システムの検査手法の一つとしてモデル検査があり，本研究室では典型的なフォールトをパターンとして検出する手法に関する研究を行っている，検証前にフォールトを取り除くことにより検証コストの削減が期待できる [7]．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解析ではなく，構文レベルの分析でフォールトを検出する．

本研究の目的は，フォールトパターン検出ツールの実現および，フォールトパターンによるフォールト検出の有効性を確認することである．ツールを作成することにより，机上でのフォールト検出を，実際にツールを使用してシステム記述から検出できるようにする．

本研究では，システム記述はプロセス代数 CSP に変換されることを想定し，CSP 記述に対する効率的なイベント生起パターン検出ツールを作成する．並行システム記述からグラフを生成するグラフ生成器，正規表現からオートマトンを生成するオートマトン生成器，パターン照合を行うパス照合エンジンの３つのツール群を作成した．フォールトパターンの適用事例として，自動販売機システムに適用し，その有効性について議論する．

2　研究の位置付け

2.1　想定する計算モデル

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 は外部に一つずつキューを持ち，STM 同士でキューを用いた非同期通信を行なう．STM 間のキューを用いた通信にイベント送信，イベント受信，イベント受理がある．

イベント送信とは，STM を指定してキューにイベントを送る事である．イベント受信とは，キューの最後尾にイベントを追加することである．イベント受理とは，キューの先頭から順に走査して受理可能なイベントを取り出し，イベントに応じたアクションを実行することである．

2.2　システム記述法

2.2.1　CSP

並行システムを相互作用する逐次プロセスの群として捉えて記述し，その正しさを数学的に証明するための仕様記述言語である．プロセスはイベントと通信によって状態変化し，その動きは代数演算子によって定義される．

2.2.2　CSP によるシステム記述

システムは STM の集まりで生成され，各 STM を状態遷移図で記述する．それぞれの STM の関係はクラス図を利用して表現し，STM が外部イベントを受け取った時の動作をシーケンス図を利用して表現する．また，入力となる環境と仕様はアクティビティ図を利用して記述する．検査対象の CSP を生成するには，システム全体の動きを記述した UML とシステムに対する入力となる環境，期待する動作を記述した仕様をそれぞれ作成する必要がある．

2.2.3　UML 表現から CSP 記述への変換

UML から CSP の生成は，先行研究において製作された既存のツール [6] を用いて自動的に行なうことが可能であるが，本研究では必要最低限の演算子で記述された CSP でグラフ生成できるようにツールを改変している．例として，ステートマシン図の CSP への変換を示す．

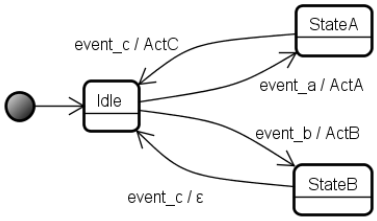


図 1　UML 表現

図 1 から生成される CSP は以下のようになる．

```
Idle = event_a -> ActA; StateA
       [] event_b -> ActB; StateB
StateA = event_c -> ActC; Idle
StateB = event_c -> Idle
```

2.3　検証の枠組み

並行システムの検証は，STM と計算モデルおよびシステムの振舞い仕様をプロセス代数 CSP[1] で表し，CSP の代表的なモデル検査器 FDR[2] を用いる．検証の枠組みを図 2 に示す．UML 表現された対象とする並行システム，環境，仕様を CSP へ変換を行なう．計算モデルと，対象システム，環境の CSP を合成した検査対象の CSP を，仕様の CSP と共に FDR へ入力する．以上のようにして，検査対象が仕様を満たすか否かについて，安全性と活性という二つの性質にもとづいて検証をする．

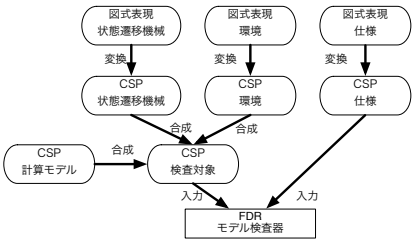


図 2 検証の枠組み

2.4 フォールトパターンによるフォールト検出

フォールト検出をグラフにおけるパスの照合問題に帰着させる．フォールトパターンによるフォールト検出の枠組みを図 3 に示す．

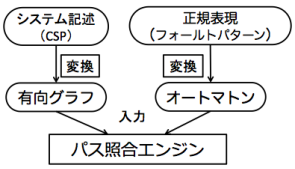


図 3 フォールトパターンによるフォールト検出の枠組み

パスの照合はグラフとオートマトンをパス照合エンジンに入力して照合を行なう．グラフは CSP 記述の検査対象を変換して作成し，オートマトンは正規表現で記述されたフォールトパターンを変換して作成する．正規表現で記述するのが困難であると考えられるパターンは，有限オートマトンで記述することを試みる．

本研究では，フォールトパターンによるフォールト検出を実現するためにパターン検出ツールを実現する．パターン検出ツールとは，パス照合エンジン，UML から CSP へ変換するツール，並行システム記述から根付き有向グラフに変換するグラフ生成器，正規表現からオートマトンに変換するパターンのオートマトン生成器というツール群の総称である．

3 パターン検出ツール

3.1 有向グラフ生成器

3.1.1 演算子の操作的意味

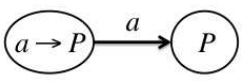
変換対象となる演算子の操作的意味 [3] と操作的意味を参考にしたグラフを示す．ここでは接頭辞と外部選択とシェアリングの演算子について説明する．

- 接頭辞

操作的意味

$$\frac{}{(a \to P) \xrightarrow{a} P}$$

グラフ



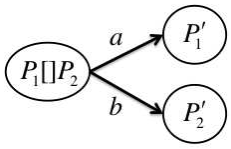
$(a \to P)$ という接頭辞のプロセスに対して外部イベント'a' が与えられた場合，プロセス $(a \to P)$ はプロセス P に遷移する．

- 外部選択

操作的意味

$$\frac{P_2 \xrightarrow{b} P_2'[a \neq b]}{P_1[]P_2 \xrightarrow{a} P_1'}$$
$$P_2[]P_1 \xrightarrow{a} P_1'$$

グラフ



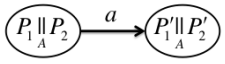
プロセス P_1 に対して外部イベント'a' が与えられた場合プロセス P_1' へと遷移し，プロセス P_2 に対して外部イベント'b' が与えられた場合プロセス P_2' へと遷移する．

- シェアリング

操作的意味 (同期あり)

$$\frac{P_2 \xrightarrow{a} P_2'[a \in A^{tick}]}{P_1 \| _A P_2 \xrightarrow{a} P_1' \| _A P_2'}$$

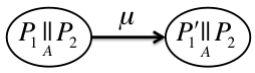
グラフ 1



操作的意味 (同期なし)

$$\frac{P_1 \xrightarrow{\mu} P_1'[\mu \notin A^{tick}]}{P_1 \| _A P_2 \xrightarrow{\mu} P_1' \| _A P_2}$$
$$P_2 \| _A P_1 \xrightarrow{\mu} P_2 \| _A P_1'$$

グラフ 2



プロセス P_1 と P_2 が同じイベントで遷移し，且つそのイベントが同期イベントである場合，両方のプロセスが同時に遷移をする．また，μ が同期イベントでなく tick でもない場合は，μ を受け取ったプロセスが個別に遷移をする．

3.2 パターンのオートマトン生成器

正規表現を既存のアルゴリズム [5] に従って非決定性オートマトンに変換し，非決定性オートマトンを決定性オートマトンに変換する．正規表現で扱われる演算子を表 1 に示す．

表 1 正規表現で扱われる演算子

演算子	働き
;	逐次実行
+	排他選択
*	直前の文字列 0 回以上の繰り返し
()	グループ化

4 パス照合エンジン

4.1 照合方法

パスの辿り方は検査対象を辿る方法とパターンを辿る方法があり，探索アルゴリズムは深さ優先探索と幅優先探索がある．照合方法はパスの辿り方と探索アルゴリズムの組み合わせで 4 つの方法がある．検査対象から辿る方法と深さ優先探索について以下に示す．

検査対象を辿る　検査対象から辿る場合の照合アルゴリズムは，検査対象を探索して，探索時のイベントの入力によってパターンのオートマトンが遷移可能かを調べる．パス照合のアルゴリズムの概要を図 4 に示す．パターンの

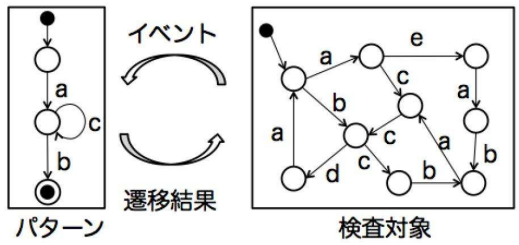


図 4　パスの照合

オートマトンの遷移結果は，(1) 受理状態に遷移，(2) 遷移可能，(3) 遷移失敗とし，受理状態に遷移した場合はパスの照合が成功したことを示す．パスの照合に成功した場合，受理状態までのパスを出力する．

4.1.1　深さ優先探索

検査対象に対して深さ優先探索を適用した際の照合アルゴリズムを以下に示す．

```
#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;
}
```

(a) でパターンのオートマトンの遷移を行い結果を取得する．(b) でバックトラックを行う．trace には辿ったの辺を格納する．(c) で二回以上同じ辺を辿って受理可能の状態に遷移する場合を考慮して辿る辺の回数の上限を判定する．

4.2　パスの出力で表示するイベント

フォールトの特定には，パターンで着目するイベント以外に参照したいイベントも必要となる．フォールトの特定に必要なイベント以外を隠蔽することでフォールト特定に必要なイベントのみを表示できる．

5　パターン検出ツールの考察

5.1　パターン検出ツールの正当性

グラフ生成器，オートマトン生成器，パス照合エンジンの正当性確認のためにテストを行なった．以下ではグラフ生成器のテストについて説明する．

構文木生成のテスト　CSP の構文をあらわした BNF を仕様としてテストを行なう．構文木生成でプロセスとして扱われる 9 種類の構文を無項演算子と単項演算子，二項演算子に分類してそれらの組み合わせでテストケースを考える．また，テストで確認する構文木の深さは 2 までのものとする．全てのプロセスは以上の組合わせを再帰的に組み合わせることで表現できる．結果として 417 通りのテストケースが考えられる．

グラフ生成のテスト　CSP の操作的意味を仕様としてテストを行う．テストケースは構文木生成のテストで使用したテストケースを利用する．テストの確認方法は，システムの CSP のイベント列をインタープリタ形式で確認できる probe を利用する．また，FDR を利用してグラフの状態数と遷移の数を確認して正確なグラフを作成する．

5.2　パターン検出ツールの評価

本研究室で作成した事例にフォールトパターンを適用してパターン検出ツールを用いて照合を行なった．適用した事例は下記の 2 つである．

(1)　基本の自動販売機システム

(2)　タイマー機能の付いた自動販売機システム

パターンで着目するイベント以外に参照するイベントは (1) は TemporaryStorage の受理イベント，(2) は VendingMachine の受理イベントとする．また，検査対象を辿るループの回数は 1 回とする．

パターン　今回の照合には，送信受理パターンを用いた．送信受理パターンを下記に示す．

(R<-event@S;R.event@S)*;R<-event@S;!R.event@S

送信受理パターンに下記のイベントを当てはめて照合したところ，どちらもフォールトが検出できた．

(1) の場合

R<-event@S = send.TemporaryStorage.off

R.event@S = recv.TemporaryStorage.off

(2) の場合

R<-event@S = send.VendingMachine.timeout

R.event@S = recv.VendingMachine.timeout

5.3　大規模なシステムへの適用

自動販売機システムにパターン検出ツールを適用し，事例から計算量について考察する．今回は STM が 10 個以

上あり，グラフ化したときの状態数が 100 万を越える大規模な自動販売機システム三つを対象とした．事例への適用結果から，FDR の検証時間と同等かあるいはやや遅い程度でパス照合が可能であることが分かった．

5.4 パス照合エンジンの機能

5.4.1 状態数削減

パスの出力で表示するイベント以外を隠蔽する際に，隠蔽したイベントをグラフから取り除くことを考える．これによってグラフの状態数削減ができ，パス検出のための時間を削減することができる．

5.4.2 最短パスの検出方法

幅優先探索で照合を行えば最短パスは必ず見つかる．また，初めに正規表現において*のないパターンで照合を行なうことで，計算量の削減が期待できる．

5.5 グラフ生成器のアーキテクチャ

グラフ生成器の機能から要求として新たな演算子の追加の容易性とグラフ生成処理に対する解析性，グラフ生成処理の変更容易性の 3 つが挙げられる．

演算子の追加の容易性を保証するために，抽象構文木のデータ構造の設計には Composite パターンを適用する．

グラフ生成処理に適用するパターンとして，Chain of Responsibility パターン，Interpreter パターン，Visitor パターンの 3 つのパターンが考えられる．しかし，Chain of Responsibility パターンでは処理が各オブジェクトに散らばってしまうのでグラフ生成処理の解析性は低くなってしまう．また，Interpreter パターンでは新しいグラフ生成処理を追加する際に，すべてのオブジェクトに対してグラフ生成処理を新たに定義する必要があるので変更に手間がかかってしまう．一方で，Visitor パターンと Interpreter パターンを組み合わせた場合は，抽象構文木に対する変更が起こった場合は変更に手間がかかる．本研究では，演算子の追加による抽象構文木の変更が考えられるが，各演算子ごとのグラフ生成処理は意味定義によって独立して記述されており，演算子の追加による他の構文への影響は少ないと考えられる．

以上の理由からグラフ生成処理の設計に Visitor パターンと Interpreter パターンを組み合わせて適用した．

設計したグラフ生成器のアーキテクチャを図 5 に示す．

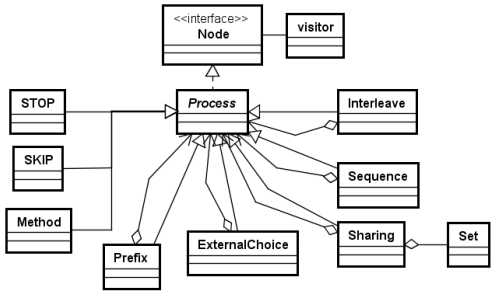


図 5 グラフ生成器のアーキテクチャ

5.6 関連研究との比較

並行システムにおける既存のデバッグ支援の研究として，反例の視覚化 [8]，および，反例の自動解析 [4] に関する研究がある．本研究とこれらを比較することで，パターンによるフォールト検出の有用性について考察する．

反例の視覚化に関する研究では，反例におけるイベントの生起順序を図示することでデバッグを支援する．反例は見やすくなるが，フォールトを特定するにはシステムの意味を考えなくてはならない．本研究では，典型的なフォールトをパターンとして定義することで，構文レベルの解析によりフォールトを特定することができる．

反例の自動解析に関する研究では，一つの意味に限定して解析することでデバッグを支援する．自動的に誤りの修正箇所まで指摘できるが，一つのフォールトしか検出できない．本研究では，誤りの完全さを捨てる代わりに多くの可能性があるフォールトを検出することができる．

6 おわりに

本研究ではパターン検出ツールを作成し，自動販売機システムにフォールトパターンを適応してフォールトを検出した．今後の課題として，グラフ生成時の記憶域消費量を抑えることで状態数の増えたグラフへの対応やループの処理に制限を与えない探索方法の実現があげられる．

参考文献

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

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

[3] S. Schneider, *Concurrent and Real-time Systems,* WILEY, 1993.

[4] T.Bochot, P.Virelizier, H.Waeselynck, and V.Wiels, “Paths to property violation: a structural approach for analyzing counter-examples,” *2010 IEEE 12th International Symposium on HASE,* pp.74-83.2010.

[5] 石畑清，アルゴリズムとデータ構造，岩波書店，1989.

[6] 小栗 達也，山内 宏也，“アーキテクチャ記述の振舞い検証支援ツールに関する研究，” 南山大学 2010 年度卒業論文，2011.

[7] 神谷 浩翔，“フォールトパターンを利用した実行前検査の研究，” 南山大学大学院 2011 年度修士論文，2012.

[8] 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.