並行システム記述に対するパスの照合の研究

M2011MM080 山内宏也
指導教員：野呂昌満

1　はじめに

並行システムを開発する上で，システムの振舞いを検証する方法にモデル検査がある．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上で重要な情報である．しかし，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．本研究室では，並行システム記述に対するパターンを用いたフォールト検出に関する研究を行なうことにより，検証コストの削減を目指している [7].

本研究の目的は，並行システム記述に対するフォールトパターン検出ツールの設計・実現および，その有効性を確認することである．本研究では，状態遷移機械に基づいた図式表現による並行システム記述をプロセス代数 CSP[2] に変換して検証することを前提としている．CSP 記述に対するイベントのパターン検出ツールを作成することにより，フォールトの検出を自動化することを目指す．

本稿では，UML から CSP に変換するツールとパス照合を行なうツールとして，CSP からグラフに変換するツール，正規表現からオートマトンに変換するツール，パス照合エンジンを開発して，CSP のモデル検査器 FDR[3] よりフォールト検出に有効であることを議論する．パス照合にかかる計算量を考え，開発したツールで計算量が削減されているかを考察した．また，自動販売機の事例に適用することで，イベントの隠蔽が計算量の削減に妥当であることを確認した．本研究で作成したツールを利用する事で検証コストの削減に寄与することが期待できる．

2　背景技術

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 (State Transition Machine，以下 STM と呼ぶ) の集合として捉える．各 STM は外部に一つずつ有限長のキューを持ち，STM はキューを介した非同期通信により動作するものとする．

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，イベント受理の三つがある．イベント送信とは，アクション実行時に STM を指定して，相手のキューが満杯でなければイベントを送信することである．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，キューを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．本計算モデルでは，イベント送信とイベント受信が同時に起こるものとして捉え，システムの振舞いはイベント送信とイベント受理のみに着目する．

2.2　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 CSP で表し，CSP の代表的なモデル検査器 FDR を用いる．検証の枠組みを図 1 に示す．まず，STM と環境，仕様を，それぞれ図式表現から CSP に変換し，変換した STM，環境と計算モデルの CSP を一つに合成して検査対象の CSP を作成する．次に，作成した検査対象と仕様を FDR に入力することで仕様を満たすかどうか検証する．

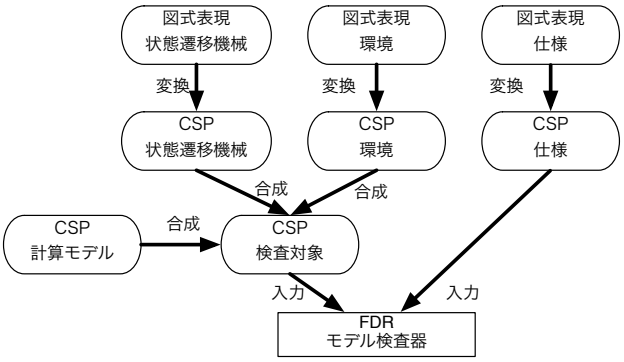


図 1　検証の枠組み

2.3　フォールト検出の枠組み

本研究では，フォールトパターンを正規表現で記述する事により，フォールトパターンによるフォールトの検出を根付き有向グラフにおけるパス照合問題に帰着させる．フォールト検出の枠組みを図 2 に示す．検査対象の根付き有向グラフとフォールトパターンのオートマトンをパス照合エンジンに入力し，照合を行なうことでフォールトの検出を行なう．有向グラフは並行システム記述の CSP から生成し，オートマトンは正規表現から生成する．本研究では，これらの変換と照合を行なう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，UML から

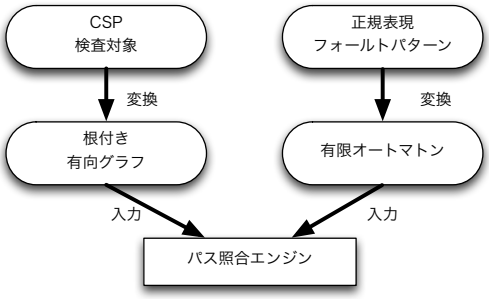


図 2　フォールトパターンによるフォールト検出の枠組み

CSP へ変換する CSP 生成器，CSP 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，パス照合エンジンの四つのツールから成る．

3　パターン検出ツール

3.1　UML の CSP 生成器

状態遷移機械に基づいた図式表現に UML を用いて CSP 記述へ変換を行なう．プロセス代数 CSP(Communicating Sequential Processes，以下 CSP と呼ぶ) とは，並行システムのプロセスが通信によって互いに相互作用することを記述するための表記法のことである [1]．UML から CSP 記述へ変換するツールは既存研究で作成されており，自動変換が可能となっている [6]．本研究では，2.1 節で示した計算モデルを表わすために必要最低限な CSP 演算子を用いる．必要最低限の CSP 演算子で構成された CSP へ変換できるように既存のツールに変更を加えた．

3.1.1　対象となる記述

図式表現から CSP 記述へ変換する対象には 2.2 節で説明した STM と環境と仕様がある．STM は状態遷移図およびシーケンス図，環境と仕様はアクティビティ図を使って記述し，CSP へ変換する．

3.2　CSP のグラフ生成器

3.2.1　対象となる CSP 演算子

CSP からグラフへの変換では，計算モデルを表わすために必要最低限な演算子を変換対象とする．対象となる演算子 [3] を表 1 に示す．

表 1　変換対象となる演算子

演算子	CSP 記述
接頭辞	a -> P
外部選択	P_1 [] P_2
インターリーブ	P_1 ||| P_2
シェアリング (同期)	P_1 [||] P_2
内部選択	P_1 |~| P_2
逐次処理	P_1 ; P_2
正常終了	SKIP
停止	STOP

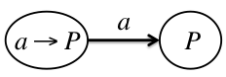
3.2.2　演算子の操作的意味

変換対象となる演算子の操作的意味 [4] と操作的意味から作成するグラフを示す．ここでは接頭辞とシェアリングの演算子を取り上げて説明する．

- 接頭辞

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

グラフ



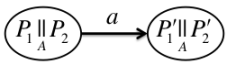
$(a \to P)$ という接頭辞のプロセスに対して外部イベント'a' が与えられた場合，プロセス $(a \to P)$ はプロセス P に遷移する．

- シェアリング

操作的意味 (同期あり)

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

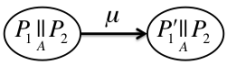
グラフ 1



操作的意味 (同期なし)

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

グラフ 2



プロセス P_1 と P_2 が同じイベントで遷移する場合かつ，そのイベントが同期イベントの集合に含まれる場合に限り，両方のプロセスが同時に遷移をすることができる．また，イベント μ で遷移する場合は，μ が同期イベントの集合に含まれない場合かつ，正常終了イベント $tick$ でない場合に限り，μ を受けたプロセスだけが遷移することができる．

3.3　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 2 に示す．

表 2　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　パス照合の定義

パスの照合とは，入力として以下の四つが与えられたときに，

- 根付き有向グラフ (検査対象)
- 有限オートマトン (パターン)
- パターンとして着目するイベントの集合
- 表示するイベントの集合

「有限オートマトンが受理する根付き有向グラフのイベントのパス」を出力することである．

4.2　辿り方

パス照合にはパスを辿る対象として，検査対象とパターンがある．ここでは，検査対象から辿る場合について示す．検査対象から辿る場合のアルゴリズムの概要を図 3 に示す．アルゴリズムは，検査対象を探索して，探索時の

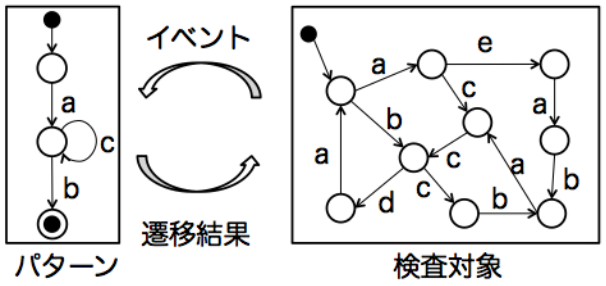


図 3 検査対象から辿る照合アルゴリズムの概要

イベントをパターンへの入力として，パターンが遷移可能か調べる．パターンのオートマトンの遷移結果は，(1) 受理状態に遷移，(2) 遷移成功，(3) 遷移失敗とし，受理状態に遷移する場合がパスの照合の成功となる．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．

4.3 探索方法

パスを辿る際の探索方法として，深さ優先探索と幅優先探索が考えられる．ここでは検査対象に対して深さ優先探索を適用した際の C 言語風の照合アルゴリズムを図 4 に示す．

```
#define N 辺を辿る回数の上限
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 4 深さ優先探索の照合アルゴリズム

(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．(b) ではバックトラックを行なっており，trace に辿った辺を格納している．(c) では，二回以上同じ辺を辿った後に受理可能となるパターンを考慮して，辿る辺の回数の上限を判定している．

4.4 状態数の削減

本研究の特徴として，「フォールトパターンは着目するイベントの数が検査対象に対してとても少ない」という点がある．本研究ではこの特徴に注目し，「着目しないイベントの隠蔽が状態数削減に効果的である」と捉え，状態数の削減を行なう．

隠蔽を利用した状態数削減の例を示す．まず，パターンで着目しているイベント以外を隠蔽して内部遷移 τ に変換する例を図 5 に示す．ここでは，検査対象のイベン

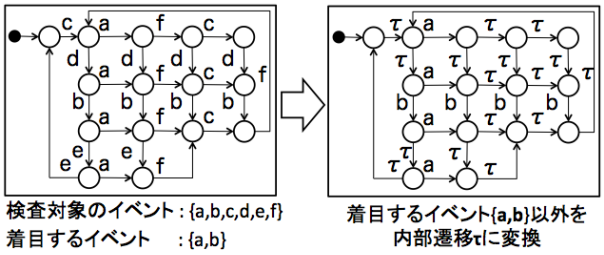


図 5 イベントの隠蔽

トの中から，パターンで着目するイベント a, b 以外のイベントを全て τ 遷移に変換している．次に，変換した τ 遷移を全て縮合する例を図 6 に示す．ここでは，NFA を

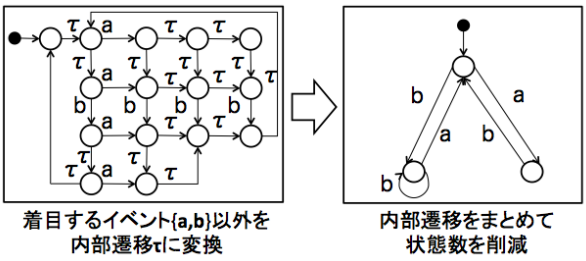


図 6 状態数の削減

DFA へ変換するアルゴリズムを用いて，内部遷移 τ を縮合し，状態数の削減を行なっている．今回の例では，隠蔽を利用した状態数削減により，状態数が 15 個から 3 個へ削減できることから，状態数削減は効率的であると言える．

4.5 表示するイベントの指定

フォールトを特定する際には，パターンとして着目するイベントに含まれない，何かしらのイベントの振舞いを調べたいという要求がある．例としては，以下のようなものが挙げられる．

- パターンで着目している STM に関係するイベント
- あるキューに対するイベント

本研究では，このように関心のあるイベントを任意に指定できる機能を作成することで，フォールト特定の支援を行なう．

5 考察

5.1 FDR とパス照合エンジンの比較

FDR の詳細化関係 (refinement) を用いたフォールト検出について考え，本研究で作成したパス照合エンジンを使ったパス照合の有用性を示す．

FDR では，失敗モデル (failure model) による活性検証を行なうことで，仕様で記述されたイベント列が必ず起こることを検証することができる．しかし，活性検証で

フォールトを検出するためにはパターンで記述された振舞いを CSP を用いて厳密に記述する必要があり非常に困難である．また，FDR を用いた検証では，フォールトを検出した場合の結果が真となり反例の出力がされないという欠点がある．この欠点を解消するには，あらかじめ仕様にフォールト特定に必要となるイベントの振舞いを全て加えて記述しなければならず，現実的に不可能である．

上記の理由から，FDR を使ったフォールト検出は困難であると考えられるので，フォールトの検出は，パス照合エンジンを使った検出が有効だと考えた．

5.2 パス照合エンジンの計算量

5.2.1 パス照合の計算量のクラス

パス照合における最悪の計算量は，全てのパスをしらみつぶしに探索したときにパスが一つも検出できない場合である．このときの計算量は以下のように n の指数関数となる．

$$O(n^m)$$

n はある状態から遷移可能な辺の数を表わしており，m はグラフ (木) の深さを表わしている．全ての状態から遷移可能な辺が n 本ある場合に，深さ m までの全ての組合わせでパスを行なう際に最悪となる．本研究では，グラフの木の深さを減らす方法としてバックトラックを用い，遷移可能な辺を減らすために状態数削減を用いて計算量の削減を試みている．次節では，バックトラックによる計算量の削減を自明のものとし，状態数削減による計算量削減について考察する．

5.2.2 状態数削減による計算量の削減

本研究で扱っている状態数削減で考察すべき計算量は次の２点である．

- τ 遷移をまとめる
- 状態数削減後の探索にかかる計算量

本研究では，τ 遷移をまとめる方法として，NFA を DFA に変換するアルゴリズムを利用している．この変換にかかる計算量は最悪の場合で，正規表現の長さ r の指数関数程度になることが分かっている [5]．しかし，そのような大きさになるオートマトンは極端な場合であり，本研究で扱う計算モデルではこのような計算量がかかることは少ないと考えた．

次に，状態数削減後の探索にかかる計算量について考察する．最悪の場合は，削減前と同様にパスが一つも検出できないときだと考えられ，以下のようになる．

$$O(k^m) \quad (k = n)$$

k はパスを検出した際に表示するイベントの数を表わしており，最悪の場合は全てのイベントを表示するときである．しかしながら，通常全てのイベントを表示することは考えられず，表示するイベントの数は元のイベントの数に対して非常に少なくなると考えられ，計算量は以下のようになる．

$$O(k^m) \quad (k \ll n)$$

この計算量は k の指数関数であるが，k は状態数削減前の n に対して非常に少なく，一桁程度となる．また，前述のバックトラック法を用いることにより，指数 m も削減できることから本研究で提案した方法により計算量は削減できたと考えた．

5.3 パターン検出ツールへの事例適用

自動販売機システムにパターン検出ツールを適用し，4.4 節で示したイベントの隠蔽を用いた状態数の削減の妥当性について考察する．適用した事例は STM が 14 個あり，グラフ化したときの状態数が 200 万の自動販売機システム三つを対象する．システムが，$(a;b)$ の繰り返しを満たすかどうか，適当なイベントを 2〜5 個残した際の状態削減後のシステムの状態数と，状態削減とパス照合にかかる合計の計算時間を記録し，FDR と状態数削減の点において比較を行なった．

記録したデータの平均値をとったところ，状態削減後の状態数はイベントを二つ残した場合から順番に，$\frac{1}{160,000}$，$\frac{1}{35,000}$，$\frac{1}{13,000}$，$\frac{1}{8,000}$ となり，十分な状態削減が行なわれていることが分かった．また，表示するイベントの個数によって状態削減とパス照合における計算時間は変化し，表示するイベントが少ないほど計算時間が短いことが分かった．一方，FDR を用いた場合では，イベントを残す数に限らず状態数は一定となり，計算時間の変化はなく一定となった．これらの結果より，イベントの隠蔽を用いることにより，状態数の削減および，計算時間を削減できるので，イベントの隠蔽は妥当だと考えられる．

6 おわりに

本研究では，UML から CSP へ変換するツールとパスを照合するツールを開発し，自動販売機の事例に適用してフォールト検出に有効であることを示した．今後の課題として，最短のパターンを検出するアルゴリズムや，作成するグラフを制限するアルゴリズムなど，フォールトパターンの検出に特化したパスの照合アルゴリズムを開発することによる計算量の削減が考えられる．

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

[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.