# 実行ログ分析に基づくフォールトのパターン化による並行プログラムのデバッグ支援

M2019SE007 高木裕也
指導教員：沢田篤史

## 1　はじめに

並行プログラミングにおいて，システムの障害（failure）からプログラムのフォールト（fault）を特定するデバッグは困難な作業である．並行プログラムのデバッグでは，障害の状況から，障害に至る動作履歴を想定してプログラムのフォールトを特定する．並行プログラムではプロセスのコンテキストスイッチが非決定的に切り替わるので，検討すべき動作履歴は単一プロセスに比べて複雑になる．

McDowell らのサーベイ論文 [2] でまとめられている通り，並行プログラムのデバッグは古くから扱われている問題である．当時から現在に至るまでデバッグのための様々なツールやアプローチが継続的に提案されている [3, 4] ．これら様々なデバッグ支援技術に共通する特徴は，対象プログラムの実行時に観測されるログなどの情報を扱っていることである．実行時の情報を，それぞれの観点からのモデルに基づいて抽象化し，フォールト箇所の特定や修正に用いている．

本研究の目的は，並行プログラムの同期問題に関するデバッグを支援することである．並行プログラムにおける同期制御の基本構造は典型的な同期問題の解法にあると仮定して，同期問題の解法に対してプログラムの実行履歴から，プログラムのフォールトを特定する方法の提案を目指す．プログラムのフォールトとプログラム動作の軌跡の間の関係が解明されれば，障害発生時の軌跡の情報からプログラムのフォールトへの対応付けが可能になると期待できる．

本研究の基本的なアイデアは，プログラムのフォールトと軌跡の関係を定式化して，プログラムの動作の軌跡からプログラムのフォールトへの対応づけを「フォールトパターン」として提示することである．プログラムのフォールトと軌跡の間の関係を図 1 に示す．フォールトと軌跡の間の関係を表現するために，フォールトと軌跡はそれぞれ構造をもつと仮定した．フォールト構造と軌跡構造をそれぞれパターンとして表現して,「フォールト構造の集合」から「軌跡構造の集合」への写像を「誤り軌跡パターン」とし,「軌跡構造の集合」から「フォールト構造の集合」への写像を「フォールトパターン」と定義した．

本研究では，プログラムのフォールトと軌跡の間の関係を定式化するために，文献 [1] で示されている典型的な同期問題の解法を対象して，以下の方法で研究を進めた．

1. フォールトの分類
2. フォールトを特定する奇跡の構造とそのパターン表現
3. フォールトの構造とそのパターン表現

セマフォアを用いた同期問題の解法に対するフォールトを分類して，その分類を基に誤り軌跡を検討する．誤り軌跡を簡潔に表現するためのパターン表現を定義して，フォールトパターンについて検討する．

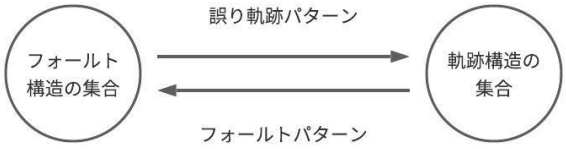


図 1　フォールトと軌跡の間の関係

## 2　並行プログラムにおける同期問題

並行プログラミングで肝心なことは，並行プロセス間での同期と通信であり，並行プロセス間における同期に関する典型的な同期問題（synchronization problem）がある．文献 [1] には，同期問題として「相互排除問題」「生産者-消費者問題」「眠り床問題」「読み書き問題」「五人の哲学者問題」「喫煙者問題」を取り上げ，セマフォアを用いた解法が示されている．

セマフォアは（semaphore）は値を持ち，P 命令もしくは V 命令を用いてプロセスの同期を制御する．セマフォアの値を 0 と 1（あるいは false と true）に限ったとき 2 値セマフォアという．整数の値としてとるセマフォアを整数セマフォアという．セマフォアは初期値を設定して利用する．

## 3　フォールトパターン

### 3.1　対象とする並行システム

#### 並行システム

並行システムは 3 項組 $(PS, V, E)$ として定義する．$PS$ はプロセスの集合である．並行システムは，インターリーブにより並行合成され，セマフォを用いて同期を制御するとみなす．$V$ はプロセス間の変数の集合である．変数は全てのプロセスから参照できる共有変数である．変数の内部構造はここでは規定していない．変数は代入により値が変化しその値は環境で保持される．環境 $E$ は変数の集合 $V$ から値の集合への写像である．値の内部構造はここでは規定していない．

#### プロセス

プロセスにはプログラムが対応づいている．プログラムは，アクションの列として定義する．

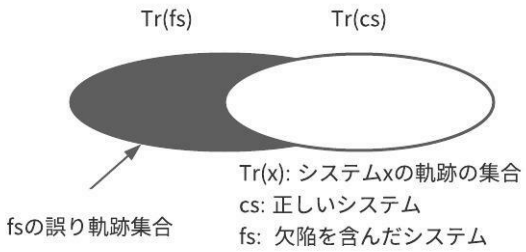


図 2 誤り軌跡集合

**アクション**

アクションはプログラムの基本構成要素の集合である．基本構成要素は，命令型プログラムの基本構成要素にセマフォアの同期命令を加えたものである．命令型プログラムの基本構成要素は，代入，条件，繰り返しとする．セマフォアの同期命令は $P(x)$ と $V(x)$ とする．$x$ はセマフォアを区別するための名前である．

### 3.2 誤り軌跡

**軌跡集合**

軌跡集合 $Tr(S)$ は並行システム $S$ の軌跡の集合として定義する．軌跡は $S$ がある時点まで実行した動作履歴である．

**軌跡**

軌跡は，実行の有限列として定義する．実行は 3 項組 $(P, A, E)$ である．$P$ はプロセス，$A$ はアクション，$E$ は環境である．

**誤り軌跡**

誤り軌跡は，正しいシステムに対して，フォールトの存在を判定できる軌跡とする．

**誤り軌跡集合**

誤り軌跡集合は，正しいシステムに対して，フォールトの存在が判定できる軌跡の集合である．正しいシステム $cs$ に対する誤り軌跡集合は，以下の写像 $ETr_{cs}$ で定義される．

$$ETr_{cs}(fs) = Tr(fs) - Tr(cs)$$

$fs$ はフォールトを含んだシステムである．誤り軌跡集合は，図 2 で示された灰色の部分で，フォールトを含んだシステム $fs$ の軌跡集合から正しいシステム $cs$ の軌跡集合の差である．

**誤り軌跡パターン**

誤り軌跡パターンは，フォールトの構造から誤り軌跡の構造への写像である．

**フォールトパターン**

フォールトパターンは，誤り軌跡の構造からフォールトの構造への写像である．

## 4 フォールトに対する軌跡の分析

### 4.1 フォールトの分類

同期問題のプログラムは，P 命令と V 命令の使い方と，共有変数の使い方の誤りの 2 種類に分類される．それぞれの単一のフォールトを検討した後で，複数のフォールトについての誤り軌跡を検討する．

#### 4.1.1 セマフォアの誤用

セマフォアの同期命令の間違った使い方について検討する．同期命令には P 命令と V 命令がある．セマフォアは複数存在する場合があるので，セマフォアを識別するために名前がある．同期命令は，名前を含めて P(名前)，V(名前）と表現する．セマフォアの誤った使い方として以下のものが考えられる．

(1) セマフォア初期値誤り

セマフォアの初期値の値を間違えた誤りを「セマフォアの初期値誤り」とする．セマフォアを利用する場合，その初期値の設定が必要である．

(2) セマフォア抜け

何らかの要因でセマフォアが抜け落ちている誤りを「セマフォア抜け」とする．

(3) セマフォア名前誤り

セマフォアの名前の指定を間違えた誤りを「セマフォアの名前誤り」とする．同期問題の解法には，複数のセマフォアが用いられる場合がある．セマフォア名前誤りは，正しいセマフォアの名前を間違えた場合の誤りである．

(4)P-V 命令取り違い

セマフォアの P 命令と V 命令を間違えた誤りを「P-V 命令取り違い」とする．例えば，セマフォア s の P 命令である P(s) とすべきところを，V(s) と間違えた場合である．

#### 4.1.2 共有変数の誤用

同期問題の制御に用いられる共有変数に関する誤りについて検討する．共有変数の誤った使い方として以下のものが考えられる．

(1) 変数初期値誤り

変数の初期値を間違えた誤りを「変数の初期値誤り」とする．共有変数で用いられる変数は，通常，初期値が設定されている．変数の初期値誤りは，この変数の初期値を間違えた場合である．

(2) 変数更新誤り

共有変数の代入時における更新する値の式の誤りを「変数更新誤り」とする．変数更新誤りは，同期問題に関わる変数の更新値を間違えた場合である．

(3) 変数名誤り

共有変数の変数名を間違えた誤りを「変数名誤り」とする．同期問題の解法で複数の変数が用いられた場合，例

```
読みのプロセス (R)      書きのプロセス (W)
01:P(m);                01:P(w);
02:   rc = rc + 1;      02:  write;
03:  if(rc == 1)        03:V(w);
04:     P(w);
05:V(m);
06:  read  ;            共有変数
07:P(m);                int rc = 0;
08:   rc = rc -1;
09:   if(rc == 0)       セマフォア
10:      V(w);          m, w: 初期値 1
11:V(m);
```

図 3 第一種の読み書き問題の解法

えば，変数として a と b が用いられた場合，a の変数を使用するところを誤って b の変数を使用した場合である．
(4) 比較誤り
条件の記述にを間違えた誤りを「比較誤り」とする．C 言語などのプログラミング言語では，条件式で代入を使用することができる．条件式に代入が用いられた場合の誤りは，比較誤りの例である．

### 4.2 誤り軌跡の検討

前節で示したフォールトの分類に基づいて誤り軌跡を検討する．文献 [1] で紹介されている第一種の読み書き問題の解法を図 3 に示す．

#### 4.2.1 セマフォアの誤用

(1) セマフォアの初期値誤り
セマフォアの初期値が誤っていた場合は，最初の P 命令もしくは V 命令実行後のセマフォアの値が正しいプログラムと違った値になる．
(2) セマフォア抜け
セマフォアの命令が何らかの原因で実行されないことは，その命令の前後の実行履歴から特徴づけることができる．例えば，プロセス R の位置 04 の P(w) の抜けは，実行の履歴が

- 位置 03 における rc の値が 1,
- 位置 05 の V(m)

から特定することができる．
(3) セマフォア名前誤り
セマフォアの名前の誤った使用は，その誤りの実行が軌跡に含まれる．例えば，プロセス R の位置 04 の P(w) から P(m) への間違いは，実行の履歴が

- 位置 03 で rc の値が 1,
- 位置 04 の P(m)

から特定することができる．
(4)P-V 命令取り違い
P 命令と V 命令の使用を取り違えた誤りは，セマフォア名前誤りと同様の誤り軌跡となる．例えば，プロセス R の位置 04 の P(w) を V(w) に間違えた場合は，実行の履歴が

- 位置 03 で rc の値が 1,
- 位置 04 の V(w)

から特定することができる．

#### 4.2.2 共有変数の誤用

共有変数の誤用については，(1) 変数初期値誤りと (2) 変数更新誤りについて説明する．(3) 変数名誤りと (4) 比較誤りは (2) と同様の誤り軌跡となる．
(1) 変数初期値誤り
変数の初期値が誤っていた場合は，最初に変数を利用する実行で誤りを検出することができる．図 3 の解法では変数が rc である．誤り軌跡は，変数の最初の使用で rc の値が正しい初期値以外の値となる．
(2) 変数更新誤り
変数の更新値を誤った場合は，誤った更新の最初の実行でフォールトを検出することができる．プロセス R の位置 02 の更新値を間違えた場合，その間違えたアクションが軌跡に記録される．

#### 4.2.3 複数フォールト

(1) 同一プロセス内における複数フォールト
同一プロセス内における複数のフォールトがある場合は，複数のフォールトの中でフォールトの位置が最初の位置のフォールトを検出することができる．ただし，フォールトの特定がフォールトの位置以降の実行に依存する場合，別のフォールトによる履歴の影響を考慮する必要がある．次の位置のフォールトは，最初の位置のフォールトを取り除くことにより検出することができる．
(2) 複数プロセスにおける複数フォールト
複数プロセスにおいて複数のフォールトがある場合には，各プロセス毎に上記の同一プロセス内における複数のフォールトを適用して，プロセス毎に最初のフォールトを検出することができる．同一プロセス内における複数フォールトと同様に，最初のフォールトを取り除くことにより，次のフォールトを検出することが可能になる．

### 4.3 軌跡のパターン表現

#### 4.3.1 実行の表現

軌跡は実行 $(P, A, E)$ の列で定義されている．$P$ はプロセス，$A$ はアクション，$V$ は環境である．実行を以下のように表現することにする．

(プロセス名，アクション, (変数名と値の組) のリスト)

プロセスはプロセス名，アクションは基本構成要素，変数名と値の組はアクション実行後の環境の変数とその値である．例えば，$(R, rc = rc+1, (rc, 1))$ は，プロセス $R$ がアクション $rc = rc + 1$ を実行した後で，変数 rc の値が 1 であることを表している．

#### 4.3.2 誤り軌跡のパターン表現

**誤り軌跡の検討**

(1) 着目するプロセスに制限した軌跡

誤り軌跡は，ある着目したプロセスの軌跡に対して特徴付けられる．システムの軌跡 $tr$ に対して，あるプロセス P に制限した軌跡を P↑ $tr$ で表現する．この場合，実行の表現を $(P, A, E)$ からプロセス $P$ を省略することが可能となる．

(2) 実行位置による軌跡の表現

誤り軌跡は，正しい軌跡の後で複数の実行により特徴付けられる．軌跡の表現を，誤り軌跡を特徴づける最初の実行位置から表現することにする．実行の項目に，軌跡の位置を追加することで，その実行の位置を表現する．

**パターン表現**

誤り軌跡を以下のパターンで表現する．

パターン=(実行番号，アクション，(変数名，値) の列) の列

実行番号は，軌跡上のアクションが実行された順番を表す．誤り軌跡は，着目するプロセスに制限した軌跡に対して，上記のパターンを用いて記述する．軌跡 `tr` においてプロセス `R` に着目した軌跡を `R ↑ tr` とする．

上記の誤り軌跡の表現に加えて，正規表現の拡張として以下のメタ文字を導入する．

- `.`：任意の行，アクション，値などにマッチ
- `^X`：X 以外にマッチ

### 4.4 第一種の読み書き問題の誤り軌跡

#### 4.4.1 誤り軌跡のパターン表現の例

(1)R のセマフォア `m` の初期値誤り

- `R ↑ tr(1,P(m),(m,^0))`

上記の表現は，プロセス R に制限した軌跡上で，実行位置 1 のアクションが `P(m)` で，実行後の変数 `m` の値が 0 以外の軌跡の集合を表す．

(2)R の位置 04 のセマフォア名前誤り

- `R ↑ tr(4,P(^w),.)`

上記の表現は，プロセス R に制限した軌跡上で，実行位置 4 のアクションが P(w) 以外である軌跡の集合を表す．

(3)R の位置 10 のセマフォア `V(w)` 名前誤り

- `R ↑ tr(.,if(rc==0),(rc,0))(.,V(^w),.)`

この誤りは，位置 09 での rc の値が 0 のときに検出することができる．上記の表現は，プロセス R に制限した軌跡上で，`if(rc==1)` での変数 `rc` の値が 0 で，その次の実行が `V(w)` 以外のアクションである軌跡の集合を表す．

#### 4.4.2 誤り軌跡集合とフォールト集合

第一種の読み書き問題における単一フォールトに対する誤り軌跡集合のパターン表現が定義できた．セマフォアの誤用で 27 パターン，共有変数の誤用で 7 パターンの誤り軌跡を定義した．個々のフォールトに対する誤り軌跡集合に重なりはなかった．軌跡からフォールトへの対応であるフォールト写像は逆写像として定義することができる．つまり，軌跡のパターンを発見すれば，フォールトを一意に特定することができる．

## 5 考察

### 5.1 検出されないフォールト

(1) 誤りの未実行

条件文の本体に誤りがある場合，プログラムの実行で条件文の本体が実行されない場合がある．例えば，図 3 の解法の R の位置 10 のセマフォア V(w) の名前誤りについて考える．位置 10 は，位置 09 において変数 `rc` の値が 0 のときに実行される．もし，読みプロセスが読み続けて書きプロセスが実行されない状況のとき，位置 09 で `rc` が 0 になることがない．このとき，誤りが実行されないので軌跡にも誤りが記録されない．しかし，これは正しい解法でも起こりうる状況である．

(2) フォールト分類の漏れ

誤り軌跡集合の定義は，フォールトの分類を基準としてフォールトを作成したので，フォールトの分類以外のフォールトを軌跡から検出することができない．例えば，共有変数の誤用では，共有変数抜け，という誤りがない．誤り分類は恣意的なので，誤りの分類を完全に行うことはできない．第一種の読み書き問題以外の問題を解析することにより，より網羅的な誤り分類を作成することが期待できる．

### 5.2 既存研究との比較

文献 [3, 4] などで扱われているデバッグ支援技術に共通する特徴は，実行時の情報を，それぞれの観点からのモデルに基づいて抽象化し，フォールト箇所の特定や修正に用いていることである．本研究では，正しいプログラムの解法が存在することを前提として，プログラムの実行軌跡からプログラムの詳細な欠陥を特定することを目指していることが，上記の既存研究との相違点である．

## 6 おわりに

本稿では，並行プログラムにおける同期問題を対象として，軌跡からフォールトを特定するために，フォールトの分類と，フォールトを特定するための誤り軌跡のパターン表現を提示した．P-V 命令を用いた第一種の読み書き問題の解法に対するフォールト集合を示し，フォールトと軌跡の間の関係の定式化の見込みを示した．今後の課題として，他の同期問題に適用とフォールト表現の構造化があげられる．

## 参考文献


[1] 土居範久, 相互排除問題, 岩波書店, 2011

[2] Charles E. McDowell and David P. Helmbold, “Debugging Concurrent Programs”, ACM Computing Surveys, Vol. 21, No. 4, pp. 593–622, 1989.

[3] Rachel Tzoref, Sumuel Ur and Elad Yom-Tov, “Instrumenting Where it Hurts— An Automatic Concurrent Debugging Technique”, Proc. ISSTA’07, ACM, pp. 27–37, 2007.

[4] Scott D. Fleming, Eileen Kraemer, R. E. K. Stirewalt, Shaohua Xie and Laura K. Dillon, “A Study of Student Strategies for the Corrective Maintenance of Concurrent Software”, Proc. ICSE’08, ACM, pp. 759–768, 2008.