拡張トレース表を用いたマルチスレッド動作の学習支援環境†

村田　匠*

A Learning Support Environment for Multi-threaded Programs Using Enhanced Trace Tables

Takumi Murata

1　はじめに

大学におけるプログラミング教育ではアルゴリズムやプログラミング言語の文法学習に重点が置かれている. 一方で, デバッグはプログミング学習カリキュラムでは一般的に強調されていなかった分野である[1].

Web アプリケーションなどで用いられる形式にマルチスレッドプログラムがある. これはタスクを複数のスレッドに分割して処理を行うものである. シングルスレッドプログラムと比較して, 応答性の改善などが期待できる. 現代のコンピュータサイエンスのカリキュラムにおけるアルゴリズムやプログラミングのコースのほとんどは、伝統的なシーケンシャルアプローチやシリアルアプローチに焦点を当てており[2], マルチスレッド学習の機会が限定的であるという問題がある.

マルチスレッドに起因するバグは, 再現性が低いことがあり[3], プログラムの過去の状態を調査できないデバッガを用いた修正は困難である. レースコンディションはマルチスレッドに起因するバグの 1 つである. これは複数のスレッドが 1 つの共有資源に同時にアクセスした際に発生する. これらは複数スレッドの相互作用によって発生するため, 修正にはマルチスレッド特有の概念やスレッドの実行順序の理解が必要である. 一方で, マルチスレッドプログラムのデバッグを行う場合, ほとんどの生徒は, 警告したにもかかわらず, シングルスレッドプログラミングで成功した解析やデバッグのテクニックに頼る傾向があることが指摘されている[4].

2　目的

大学におけるプログラミング教育ではデバッグ学習とマルチスレッド学習の機会が限定的であるという問題がある. このことから, 本研究では拡張トレース表を用いた初学者向けのマルチスレッドプログラム動作学習ツールを作成した. 提案ツールは拡張トレース表を用いた課題を解くことで, マルチスレッドに起因するバグを事例を用いて学習する. これにより, シングルスレッドでプログラミングを学習した者のマルチスレッドプログラム動作理解を促進することを目的としている.

3　提案手法

本研究では, マルチスレッド動作学習のための拡張トレース表を提案する.

3.1　トレース

近年のデバッグ作業で広く用いられるデバッガは, プログラムをブレークポイントで停止させ, 変数等の状況を調査するものである. デバッガを利用したプログラムは通常逆戻りできないため, 過去の状態を調査できない. したがって, マルチスレッドプログラムのデバッグには不向きである. 一方で, プログラムのデバッグ手法の 1 つにトレースを用いる方法がある. トレースとはプログラム実行中のイベントを逐次的に記録したものである. これを不具合が観測された地点から遡及するように検分することで開発者はプログラムの制御の流れや変数の変化を読み取って障害の箇所を特定することができる. この作業はプログラム実行後に実施するため, デバッガと比較してマルチスレッドプログラムのデバッグには有効であるといえる.

本研究のトレースは, イベント(メソッド実行, 変数代入, 条件分岐)にスレッド情報を付与したものとする.

3.2　トレース表

デバッグを行うには, プログラムの各行で変数の値がどのように変化したかを正確に読み解く必要がある.

この方法の 1 つに, 学習者のプログラミングにおけるデバッグ支援にトレース表を用いる方法がある. トレース表とはプログラム実行の各ステップにおける手続き番号, 変数の値, 条件式の結果などを逐次記述するものである. 学習者がプログラムの制御フローや各行の処理内容, 変数の

---

†本研究の一部は以下において発表した
・情報処理学会 コンピュータと教育研究会 175 回研究発表会
*電子情報メディア工学専攻　2228017　橋浦研究室

値の変化などを正確に理解していない場合，作成したトレース表にその誤りが反映される．

3.3 拡張トレース表

本研究では，各スレッドの命令実行順序を視覚的に理解しやすくするために，トレースと変数値入力欄を組み合わせた拡張トレース表を用いる．リスト 1 は複数のスレッドを用いてカウンターを増加させるプログラム(カウンタープログラム)である．このカウンタープログラムは 2 つのスレッドを作成し，それぞれのスレッドで変数 c を 1 つずつ加算する．したがって，リスト 2 のように最終的な変数 c の値は 2 になることが予想される．しかし，レースコンディションが起こるとリスト 3 のように変数 c の値が 1 度しか加算されないことがある．このときのトレースがリスト 4 である．

リスト 1：カウンタープログラム

```
public class Main {
  public static void main(String args[]) {
    Counter counter = new Counter();
    Thread t1 = new Thread(counter);
    Thread t2 = new Thread(counter);
    t1.start();
    t2.start();
  }
  class Counter implements Runnable {
    private int c = 0;
    public int getNum() { return c; }
    public void setNum(int n) { this.c = n; }
    @Override
    public void run() {
      int n = getNum();
      n++;
      setNum(n);
    }
  }
}
```

リスト 2：正しい変数の値の遷移

```
変数 c : 0
変数 c : 1
変数 c : 1
変数 c : 2
```

リスト 3：レースコンディションが発生した際の遷移

```
変数 c : 0
変数 c : 0
変数 c : 1
変数 c : 1
```

リスト 4：レースコンディション発生時トレースの一部

```
[thread-1] int n = getNum()
[thread-1] return c
[thread-2] int n = getNum()
[thread-2] return c
[thread-1] n++
[thread-1] setNum(n)
[thread-1] c = n
[thread-2] n++
[thread-2] setNum(n)
[thread-2] c = n
```

図 1 はレースコンディション発生時の拡張トレース表である．左側には変数値入力欄，右側にはカウンタープログラムがレースコンディションを起こした際のトレース(リスト 4)をグラフ化したものが表示されている．変数値入力欄の上部には複数スレッドからアクセスされ，出力に関わる変数(共有変数)が指定されている．変数値入力欄には同じ行のトレース記録時点の共有変数の値を入力する．グラフ化したトレースでは，同じスレッド情報をもつトレースは同じ色の長方形ノードとして表現される．記録されたスレッドの数だけ独立したグラフを描画し，複数のグラフが横並びに配置される．各グラフの左端に配置されているノードはスレッド情報を記載するノードである．記録されたイベントはスレッド情報ノードの右側に配置されており，記録されたイベント名が記述されている．イベントは上から記録された順番に配置されており，メソッド内で別のイベントが記録された場合は，そのメソッド実行イベントの右側にイベントが配置される．また，空欄はスレッドが動いていない状態を表している．

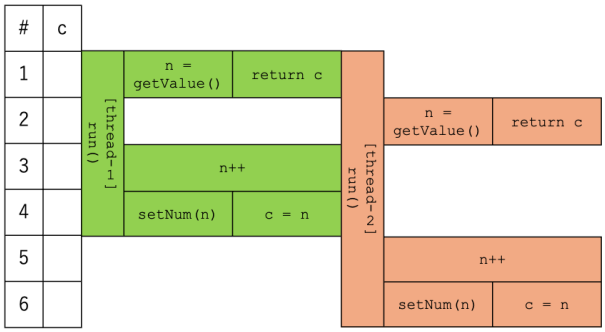

図 1：レースコンディションが発生時の拡張トレース表

4 実装

本研究では，マルチスレッド動作学習のためのツールを実装した．本ツールでは，トレース選択問題と変数値穴埋め問題を行うことで，マルチスレッドプログラムの各スレッドにおける命令実行順序，複数のスレッドからアクセスされる変数の値の変化を学習する．これにより，レースコンディションが起こる仕組みを学習する狙いがある．

4.1 トレース選択問題

マルチスレッドプログラムはスレッドの実行順序次第で様々な出力が行われる可能性がある．トレース選択問題では，出力と順番の異なる 4 つのトレースが提示され，出力に対応するトレースを選択する．これにより，出力からスレッドの実行順序を予測する．回答を送信することで答え合わせ画面に遷移する．

4.2 トレース選択問題答え合わせ画面

選択問題答え合わせ画面(図 2)では，正誤判定，選択したトレース通りのステップ実行の再現を行う．ステップ実行を確認することで，スレッド切り替えによるプログラム動作を学習する．

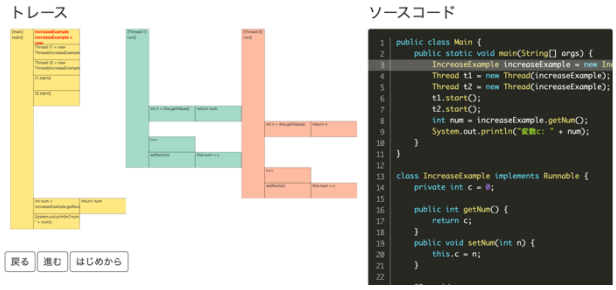

図 2：トレース選択問題答え合わせ画面

4.3 変数値穴埋め問題

穴埋め問題画面では，拡張トレース表に共有変数の値の変化を入力する．これにより，共有変数の値の変化を予測する．回答を送信することで答え合わせ画面に遷移する．

4.4 変数値穴埋め問題答え合わせ画面

穴埋め問題画面答え合わせ画面(図 3)では，誤っている回答と空欄を赤く強調表示する．また，ヒントとして共有変数に変更を加えるイベントを赤く強調表示する．学習者は，強調されたイベントに対応する入力欄以外では共有変数の値が変化しないことを前提に再回答する．これにより，共有変数の値の変化から，レースコンディションが観測されるまでの共有資源の変化を学習する．

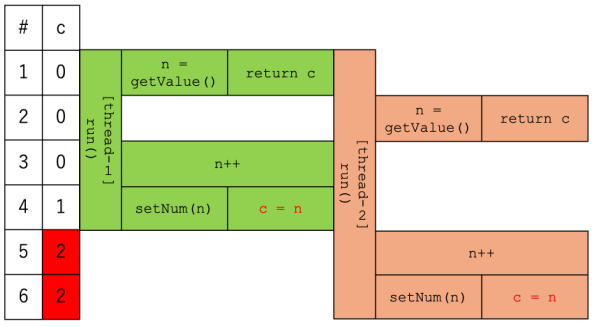

図 3：変数値穴埋め問題画面

4 評価

本研究では，マルチスレッド動作学習における拡張トレース表の有効性を確認するための実験を行った．評価の明確化のため，以下の RQ を設定した．

1. ツールを用いた学習の前後でテスト結果に差があるか
2. テキスト形式トレースと拡張トレースグラフでは学習効果に差があるか
3. テキスト形式トレースと拡張トレースグラフでは学習にどのような差があるか

実験はオブジェクト指向プログラミングを受講した大学生と大学院生 10 名を対象に行った．被験者は拡張トレース表を用いて学習する実験群とテキスト形式トレースを用いて学習する統制群に分けられた．実験の手順は以下の通りである．

1. 実験全体とテストの説明
2. 事前テスト
3. ツールの説明
4. ツールを用いた学習(2 回)
5. 事後テスト

5 実験結果と考察

5.1 RQ1：ツールを用いた学習の前後でテスト結果に差があるか

図 4 は実験群の事前テストと事後テストの正答率を比較したものである．事前テストと事後テストの結果に差があるか確認するために対応ありの t 検定を行った．その結果，$p>0.05$ より，有意差は認められなかった($\alpha=0.05$, $p=0.7169$).

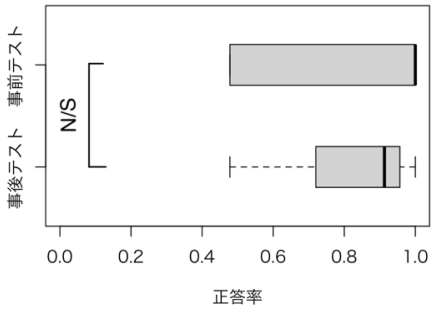

図 4：実験郡の事前テストと事後テストの正答率比較

図 5 は統制群の事前テストと事後テストの正答率を比較したものである．事前テストと事後テストの結果に差があるか確認するために対応ありの t 検定を行った．その結果，$p>0.05$ より，有意差は認めらなかった($\alpha=0.05$, $p=0.1044$)．

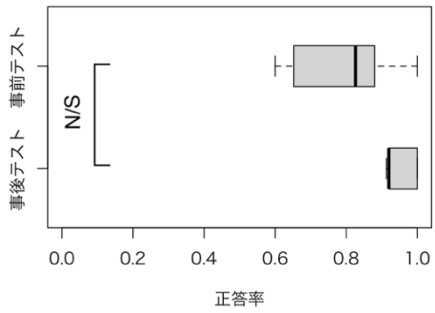

図 5：統制群の事前テストと事後テストの正答率比較

5.2 RQ2：テキスト形式トレースと拡張トレースグラフでは学習効果に差があるか

図 6 は統制群の事前テストと事後テストの正答率を比較したものである．事前テストと事後テストの結果に差があるか確認するために対応ありのt検定を行った. その結果, p>0.05 より, 有意差は認められなかった(α=0.05, p=0.1044).

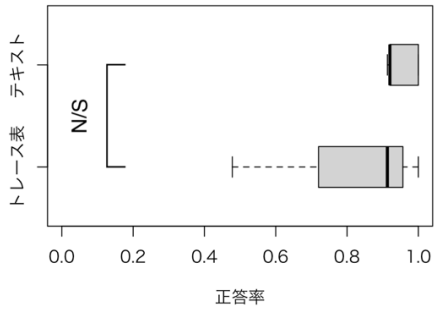

図 6：実験郡と統制群の事後テスト正答率比較

5.3 RQ3：テキスト形式トレースと拡張トレースグラフでは学習にどのような差があるか

実験群と統制群で選択問題回答数に差があるか確認するためにマン･ホイットニーの U 検定を行った. 結果, p>0.05 より，有意差は認められなかった(α=0.05,$p=0.3731).
実験群と統制群で穴埋め回答数に差があるか確認するためにマン･ホイットニーのU検定を行った．その結果, p>0.05 より，有意差は認められなかった(α=0.05, p=0.9002).

図7は実験群と統制群の学習時間を比較したものである．実験群と統制群で学習時間に差があるか確認するために対応なしのt検定を行った．その結果，p<0.05 より，有意差が認められた(α=0.05, p=0.002122)．実験群の学習時間の平均は 341.3 秒，統制群の学習時間の平均は 690.3 秒であった.

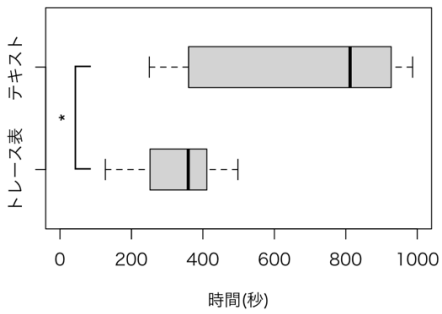

図 7：実験郡と統制群の学習時間比較

5.4 考察

実験結果から，拡張トレース表はマルチスレッド動作学習時間の短縮に寄与する可能性が示唆された.しかし，実験群の学習前後のテスト結果には有意な差が見られず，実験群と統制群の事後テストの結果にも優位な差は見られなかった．これは答え合わせ画面に遷移せずに問題画面で熟考する被験者が多く，用意したヒント機能を活用できていなかったことが原因と考えられる.

6 関連研究

トレース可視化については，様々なものが提案されている．Malnati ら[5]は，学生のマルチスレッドの理解を改善するために，スレッドごとのトレース記録・可視化ツール JThreadSpy を提案している．JThreadSpy は動的インスツルメンテーションによりメソッドフローを記録している．UML を拡張した図を用いることで, スレッドごとに実行フローを表示, メソッド呼び出しとオブジェクトの関連付け, クリティカルセクションへの同時アクセスを表現している．工学科の学生にオブジェクト指向プログラミングのコースで並行処理を教えるために使用し，並行プログラミングに関連する抽象的な概念を視覚化する貴重なツールであると述べている．その一方で，プログラムがカプセル化の原則に従っていることを前提としており，イベントはメソッド呼び出しと `return` のみしか扱っていない.

7 結論

本研究では，トレースを用いたマルチスレッド学習ツールを提案した．実験結果から拡張トレース表はマルチスレッド動作学習時間の短縮に寄与する可能性が示唆された．今後の課題として，答え合わせ機能の改善，更に規模の大きな実験の実施が挙げられる.

参 考 文 献

1) Jacqueline Whalley, Amber Settle, and A. Luxton-Reilly, “Novice reflections on debugging,” Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (online), pp.73–79, Apr. 2019.
2) D. M. Rao, "A Spiral Bilingual-Programming Approach for Teaching Concurrency and Parallelism," 2020 IEEE Frontiers in Education Conference (FIE), Uppsala, Sweden, pp.1-9, Oct. 2020.
3) Chen Li, Emily Chan, Paul Denny, A. Luxton-Reilly, and Ewan Tempero, “Towards a framework for teaching debugging,” Proceedings of the Twenty-First Australasian Computing Education Conference (online), pp. 79–86, Jan. 2019.
4) Oracle Corporation, “マルチスレッドプログラムのデバッグ,” https://docs.oracle.com/cd/E19253-01/819-0390/compile-19263/index.html, Jan. 2010 (Accessed on 2022/12/22).
5) Giovanni Malnati, Caterina Maria Cuva, and Claudia Barberis, “JThreadSpy: teaching multi-threading programming by analyzing execution traces” Proceedings of the 2007 ACM workshop on Parallel and distributed systems: testing and debugging (online), pp.3–13, Jul. 2007.

指導教授	審査委員（主査）	准教授	橋浦　弘明
	審査委員（副査）	教授	粂野　文洋
	審査委員（副査）	准教授	加藤　利康