K-019

モデルチェッキングを用いたアセンブリ並行プログラミング学習支援システム
A Learning System for the Concurrent Programming with Model Checking

吉田 英輔† 角川 裕次†
Eisuke Yoshida Hirotsugu Kakugawa

1. はじめに

並行プログラムとは、プログラムをプロセス(あるいはスレッド)に分割し、互いに協調しながら並行動作するものである。近年、CPUレベルでスレッドがサポートされるなど、計算機システムの並列、分散化が進んでいる。そのため、並行プログラムの需要が増加し、プログラマが並行プログラムにふれる機会が高まっている。しかし、一般に並行プログラムの開発は、逐次プログラムの開発に較べ格段に難しく、並行プログラムの動作やしくみを正しく理解しておかないと思わぬバグに悩まされ、たちまち手に負えなくなってしまう。したがって、並行プログラムの基本的な動作や原理、しくみを分かりやすく学習できる支援システムが求められているといえる。

2. 並行プログラム

並行プログラムは、複数のプロセスが切り替わりながら実行される。このプロセスの切り替わりは、CPU命令単位で行われ、切り替わるタイミングはスケジューラに依存している。並行プログラムを正しく動作させるためには、全てのスケジューリングで正しく動作させる必要がある。しかし、全てのスケジューリングに対して正しく動作するかをチェックするのは容易ではない。なぜなら、スケジューリングの組み合せの数は爆発的に増加するため、実行によるテストを行うことができないからである。ところが、モデルチェッキングと呼ばれる技術を用いることで並行プログラムを検証することが可能である。

3. 関連研究

モデルチェッキングを用いて検証を行うことで並行プログラミングの学習を支援するシステムが開発されている[1]。ところが、このシステムでは並行プログラムの機械語レベルの細かな動作やしくみについての学習を意図して作成されていないため、高級言語レベルの並行プログラムに対して検証が行われる。つまり、各ステートメントがアトミックに実行されるという仮定の下で検証されている。しかし、実際の計算機では、コンパイルされた機械語が実行される。そこで、本研究では並行プログラムの機械語レベルの細かな動作やしくみを学べるようにアセンブリレベルの並行プログラムを扱うことにした。これにより、全てのスケジューリングに対してモデルチェッキングを用いた検証が可能となり、正しく動作するかどうかが判定可能になった。しかし、アセンブリ言語を対象にするため、初心者には分かりにくいと考えられるため、様々な視覚化により分かりやすいシステムになるよう考慮した。

†広島大学大学院工学研究科情報工学専攻

4. モデルチェッキング

モデルチェッキングとは、システムの有限状態モデルと論理プロパティを与えると、与えられた初期状態において、そのプロパティがモデルを満たすかどうかを検証する技術である。モデルチェッキングでは、形式的な検証が効率的に行われるように工夫されている。したがって、状態が非常に多く、検証の困難な並行、並列、分散システムに対して効率的な検証が可能であるという特徴をもつ。

4.1 モデルチェッカー

モデルチェッキングでは、論理プロパティをCTL(Computation Tree Logic)やLTL(Linear time Temporal Logic)で記述する。本研究では、LTLによるモデルチェッカーとして知られているSPIN(Simple Promela INterpreter)[2]を使用した。SPINは、C言語に似た検証言語PROMELA(PROcess MEta LAnguage)により並行システムのモデルを記述して検証を行なう。したがって、アセンブリプログラムを直接検証することはできない。そこで、本研究では、アセンブリプログラムを同等な動作をするPROMELAに変換することで安全性と生存性の検証を行った。

5. 学習支援システム

5.1 システムの概要

図1は、システムの全体構成を表した概要図である。本システムでは、MIPS系アセンブリ言語により並行プログラミングを行い、システムが並行プログラムをシミュレートすることで実行動作を学習できるようになっている。また、外部プログラムのモデルチェッカーSPINを用いてモデルチェッキングによる検証を行う。これにより、正しく動作するか判定することができ、正しく動作しない場合は、そのエラーパスを見つけだして実行することができる。仕様を満たさない実行の実例を図2に示す。また、システムの特徴について以下にまとめる。

- システムはJava言語によるプログラムとして実装
- アセンブリ並行プログラムをシミュレート
- MIPS R2000の命令セットを参考に実装
- レジスタ、メモリ等のリソース値を視覚化
- アセンブリ命令単位で実行でき、Undo/Redoが可能
- モデルチェッキングによる安全性と生存性の検証
- エラーパスの視覚化と自動実行
- 危険区域問題サンプルプログラムの提供

5.2 学習の流れ

本システムを用いた学習の流れについて説明する。ここでは、予め用意したサンプルプログラムを用いた学習について紹介する。まず、サンプルプログラムを本システム上のエディタに読み込み、本システム上に仮想的なプロセスを作成する。同様に協調動作するもうひとつの

プロセスを作成する。このとき、プロセスを作成した時点で自動的にアセンブリソースコードからPromelaソースコードが生成されるようになっている。続いて、生成されたPromelaソースコードをモデルチェッカーSPINを用いて検証し、安全性と生存性を満たしているかを調べる。満たしていなかった場合は、満たしていないというメッセージを表示される。学習者は、エラーパスの再実行ができ、命令実行のUndo/Redo機能を用いて、エラーの原因を探りながら動作を学習できる。HTMLファイルにより実行順序を確認することもできる。以上がサンプルを用いた学習の流れである。この例では、サンプルを用いた学習を示したが、本システムにおいてはプログラムの検証機能を用いることで学習者自身が並行プログラミングを行うことが可能である。

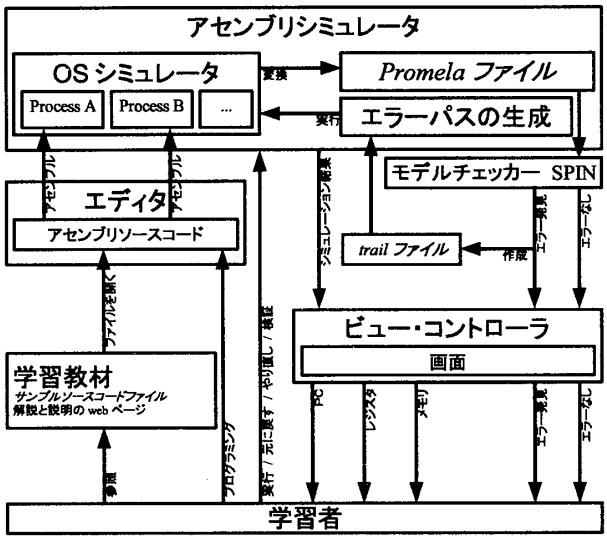

図 1: システム概要図

6. システム評価

6.1 評価実験

広島大学工学部のコンピュータ・アーキテクチャ学研究室の学部４年生６名にお願いし、本システムの評価実験を行った。評価実験の方法は、本システムを用いて基本的な危険区域問題についての学習を行い、アンケートにより評価を行った。具体的には、４つのサンプルプログラムをそれぞれ実行してもらい、レース状態や安全性と生存性、そして正しく動作することについて理解できたかどうか質問した。システムの使用法や簡単な説明はWebページとして用意し、学習者にそれらを参照してもらい、好きなペースで学習してもらった。

6.2 結果と考察

アンケート結果によると、全ての被験者は既に講義でMIPS系アセンブリ言語を用いたプログラミングの経験があることが分かった。また、一人を除いては、基本的な危険区域問題について既に講義で学んでいたことが分かった。理解度を問う質問に対しては、６人中４人がレース状態、安全性、生存性について全て理解できたと答えている。その他は安全性のみ、あるいはレース状態のみ理解できなかったという答えがそれぞれ１人だった。そのため、本システムでこれらの学習が可能であり、より深い理解を促しているといえる。講義と支援システムのどちらが理解しやすいかったかという質問に対しては、半数ずつに分かれた。おそらく、本システムのみを用いた学習ではなく、講義などで本システムを補助的に用いることが一番効果的であると考えられる。システム以外の情報(Webページやテキスト)が必要であるかという質問に対しては、全ての人が必要であると答えた。より分かりやすいWebページやサンプル、そして学習教材が求められているといえる。

また、被験者から以下のような意見が得られた。

- 正しく動作しない場合にどのように実行されるかをもっと分かりやすくしたほうが良いと思う
- 動作が目に見えて良かった、レジスタの視覚化が分かりやすかった
- 危険区域問題は高級言語レベルの方が理解しやすい
- 実行ログが直観的に分かりづらかった
- HTMLの実行例でレジスタ値が表示されないので詳細まで確認するのが困難だった
- システムを使いこなすのに時間がかかりそう

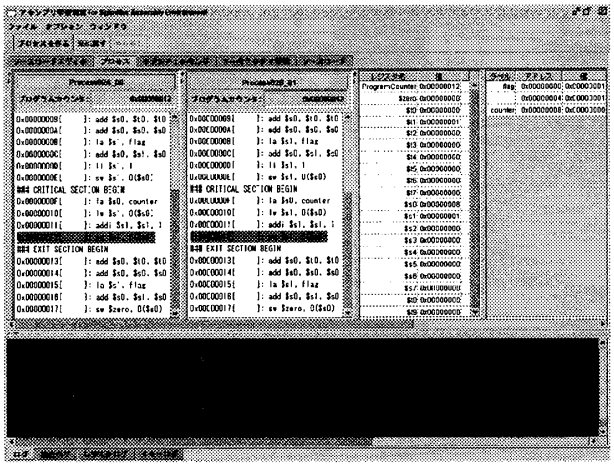

図 2: システムの実行画面

7. おわりに

以上、複雑で難しい並行プログラミングを計算機を用いて支援する学習支援システムについて述べてきた。システム評価について総合的に判断すると、従来の学習方法と較べ、具体的な実行動作が学べるため動作やしくみが分かりやすくなっている。とはいえ、一般に並行プログラミングは複雑で難しいため、より分かりやすく工夫する必要があるといえる。今後の課題としては、アンケートで得られた意見を参考にして、より分かりやすいシステムにすることである。

参考文献

[1] Seigo Tsurumi. Implementation of a learning system with model checking for concurrent programming. 修士論文, 広島大学大学院工学研究科, 2003.

[2] Gerard J. Holzmann. The Model Checker SPIN. *Software Engineering*, Vol. 23, No. 5, pp. 279–295, 1997.