形式手法と組込みソフトウェア

北陸先端科学技術大学院大学
安心電子社会研究センター　特任助教授
青木　利晃

今日の社会システムでは、あらゆる部分にソフトウェアが組み込まれているため、その誤りは時間と金銭の莫大な損失や生命の損失を引き起こすことがある。ソフトウェア工学の主要な目的の１つは、このような複雑さに対処し正しくソフトウェアを構築することである。この目的を達成する１つのアプローチは、ソフトウェア開発において、数学を基礎とした言語やツールを用いて、対象となるシステムを記述し検証すること、すなわち、形式手法を用いることである。

１．形式手法の概要

形式手法には様々なものがあり、古くは、1960年代に提案されたFloyd法、Hoare 論理などのプログラムの検証手法がある。その後、厳密に仕様を記述するための形式的仕様記述（Formal Specification）、検証技法であるモデル検査手法（Model Checking）や定理証明手法（Theorem Proving）が提案されてきた。

１.１．形式的仕様記述

形式的仕様記述では、数学的に裏付けのある概念に基づいた言語を用いて厳密に仕様を記述する。Z, VDM, Bなどが代表的な手法である。Zは集合論に基づいて仕様を記述するための言語であり、集合、関係、関数、および、 その上の制約を書くための記法がそろえられている。また、仕様をモジュール化するためのスキーマと呼ばれる概念も導入されている。このような言語を用いると、対象の仕様を厳密に解釈する必要が生じ、結果として仕様の理解や品質が向上することになる。また、記述した仕様からプログラムやテストケースを生成する手法、および、仕様の正しさをモデル検査手法や定理証明手法を用いて検証する手法なども提案されている。

１.２．モデル検査手法

モデル検査手法は、有限状態で特徴づけられる振るまいを網羅的に探索し、不具合を発見するものである。不具合が発見された場合、反例、すなわち、そのような状況へ導く処理列を出力する。モデル検査手法では、自動的に検査できる反面、網羅的に探索するため、非常に多くの状態の数を探索しなければならなくなる状態爆発問題が生じる可能性がある。そのため、多くのモデル検査手法の実装では、効率的なデータ構造や探索最適化手法を用いている。モデル検査手法を実装した様々なツールがあり、Spin, SMV, LTSAが代表的なものである。

１.３．定理証明手法

定理証明手法では、述語論理などの形式的体系を用いた証明を行う。そのため、本質的に無限の状態を含むものを取り扱うことができるが、自動化が困難である。ゲーデルの不完全性定理では、算術計算を含む述語論理では、完全な体系（すべての正しい事実が証明できる体系）が無いことが証明されている。一方で、Presburger Arithmeticのように、算術計算の範囲を限定することにより完全で決定可能（自動的に計算可能）なアルゴリズムが提案されている。定理証明手法を実装したシステムには、大別して、なるべく証明を自動化することを目指したものと、対話的な証明を支援するものがある。代表的なものとして、前者には、ACL2, Eves, LP, 後者には、Isabell, HOL, Coqなどがある。

以上のような形式手法が適用された事例は数多く報告されている。10年以上前の事例の報告であるが、文献［１］でサーベイされているので参考にしてほしい。現在では、検証ツールを動作させるコンピュータの性能の向上、ツールの完成度の向上、理論の発展など、当時とは異なる状況である。また、最近の国際会議などの

報告では、モデル検査の適用事例が数多く報告されており、その中には家電、汎用アプリケーションなども含まれている。これらのことを考慮すると、欧米諸国では、形式手法の適用が広まりつつあると考えられる。

２．組込みソフトウェア開発への応用

組込みソフトウェアでは処理のタイミング、共有リソース管理、並行性といった概念が重要である。これらにまつわる性質の検証は、モデル検査手法が得意とするところである。また、前提となる知識の量が少なく、短期間の訓練で使えるようになるため、形式手法の導入のファーストステップとして適している。

一方で、モデル検査手法には状態爆発問題がつきまとう。たとえば、8bitの変数を考えてほしい。この変数では０から255までの数字を表現できるので、最大で256個の状態を持つ。8bitの変数を２つ用いると、0から255までの数字の組み合わせなので、最大で、256の２乗個の状態を持つ。すなわち、変数の個数の指数乗で状態が増えるのである。３～４個の変数で、現在のPCのメモリを全部食い尽くすくらいの状態数になるであろう。実際のCなどのプログラムに変数が何個あるかを考えると、モデル検査ツールだけでは直接的に扱うことができないことがわかる。よって、Cなどのプログラムに書かれているくらいの情報を取り扱うためには、より高度な定理証明手法などが必要となり、すべての計算を扱う検証手法は自動化ができないことがわかっている。それくらい、プログラムの計算や振る舞いを検証することは困難な問題なのである。

このようにモデル検査手法は組込みソフトウェアの特徴である並行性などの概念の取り扱いが得意であるが、Cのプログラムのような膨大な状態を持つものについては直接的には扱えない。ここで２つの方法が考えられる。１つはCなどのプログラムではなく、その設計モデルを対象とすることである。設計の目的によっては、状態爆発を回避でき、かつ、有効な検証を行えるものがある。モデル検査手法のこのような適用法は、古典モデル検査と呼ばれている。作成した設計モデルを形式化して、検証可能なくらい厳密に記述されている検証モデルを作成する。そして、それをモデル検査ツールにより検証し、意図通りの性質が成立することを確認した後、検証モデルに基づいてプログラムを実装するのである。

もう１つの方法は、プログラムを状態爆発が生じないくらいに抽象化を行うことである。このような適用法はモダンモデル検査と呼ばれている。モダンモデル検査では、作成したプログラムを自動抽象化し、検証モデルを抽出する。そして、それをモデル検査ツールで検証する。すなわち、モダンモデル検査では、直接的にプログラムを検証することができるのである。このようなツールには、Bandera, BLAST, SLAMなどがあり、定理証明システムと連携してプログラムの自動抽象化を行ってる。抽象化手法には、個々の変数を抽象的な値に対応づけるデータマッピング法、変数に関する述語を用いて変数のまとまりで柔軟な抽象化を行う述語抽象化法などが提案されている。しかしながら、このような抽象化の手続きは、一般的には決定不能問題であるため、制限された範囲で自動化を行っている。筆者は、プログラムの抽象化は、必ずしも自動化する必要は無いと考えている。最初は人手によりアドホックな抽象化をしても良いと思う。ノウハウがたまれば系統化したり、自動化すればよいであろう。

まとめると、組込みソフトウェアに適用する形式手法であるが、まずは、並行性やタイミングにまつわる性質に絞ってモデル検査手法を試してみるのがよいであろう。適用対象としては、設計モデル、および、抽象化したプログラムである。そして、モデル検査手法には限界があるので、より広い範囲の性質や記述の取り扱いを望むのであれば、定理証明手法などその他の形式手法に段階的に手を出していくのがよいであろう。

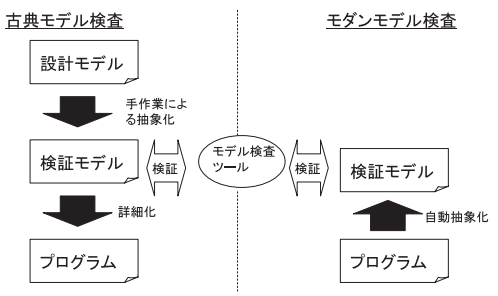

図１　古典モデル検査とモダンモデル検査

3．我々の取り組み

組込みソフトウェア検証に関する我々の取り組みを紹介する。我々は、定理証明手法とモデル検査手法を、組込みソフトウェアの正しさの検証に応用する研究を行っている。それらのうちモデル検査手法を用いた研究について紹介する。

3.1．RTOS上のソフトウェアの振る舞いの検証法

組込みソフトウェアではμITRON［2］のような優先度付きマルチタスクが扱えるRTOS（Real-Time Operating Systems）を使用する場合が多い。この場合、並行処理を直接的にプログラムできるが、その反面、動作やタイミングなどの検証が困難になる。並行動作するタスクの実行タイミングを考慮しなければならないからである。このような振る舞いの検証はモデル検査手法が得意とするところである。そこで、我々は、RTOS上のソフトウェアをモデル検査ツールを用いて検証する手法を提案している［4］。

本研究では、μITRON4.0に準拠したRTOS上で動作するソフトウェアを対象とする。また、モデル検査ツールとしてSpin［3］を用いる。Spinでは、非同期的に並行動作するプロセスを扱うことができ、μITRONのタスクの概念に近いからである。しかしながら、タスクのスケジューリングや優先度、その他のサービスコールを直接的には扱うことができず、それらによる振る舞いを無視して、単に並行に動作するタスクとして検証を行うのが通常である。たしかに、それにより検証された性質は、スケジューリングを導入しても成立する。スケジューリングを導入した振る舞いは、単に並行動作するタスクの振る舞いより小さいからである。しかしながら、検証時に、実装すると排除されるような実行列によるエラーが検出される場合が多く、本来検証したい性質が扱えないことがよくある。そこで、μITRONの振る舞いに基づいて検証が行えるライブラリを提案した。このライブラリを用いると、Spinの入力言語であるPromelaで優先度やサービスコールを直接的に記述ができ、シミュレーション、および、モデル検査による検証の際、より実装に近い振る舞いで検証を行うことが可能になる。現在、周期や処理速度などを扱う手法、RTOS自体の検証についても研究を行っている。

3.2．UMLで書かれた設計モデルの検証

ソフトウェアの仕様書や設計書を書くための標準的な記法としてUML（Unified Modelling Language）がある。UMLはソフトウェア開発で一般的に用いられるダイアグラムを集めて標準化したものであり、組込みソフトウェア開発でよく用いられる状態遷移図などの記法も含まれている。また、最近では、スケジューリングやパフォーマンスに関する要件を記述するための拡張記法UML Profile for Schedulability, Performance, and Time Specificationも標準化されている。我々は、文部科学省e-Societyプロジェクトの「高信頼性組込みソフトウェア構築技術」において、UMLで書かれた組込みソフトウェアの設計書をモデル検査技術で検証する手法を提案、および、その手法の支援ツールを開発している［5］。このツールでは、UMLのクラス図、および、ステートチャート図で書かれた設計書をモデル検査ツールSpinで取り扱い可能な形式に変換し、指定した性質が成立するかしないか検証する。変換と検証は自動的に行われる。検証した結果、指定した性質が成立しない場合に

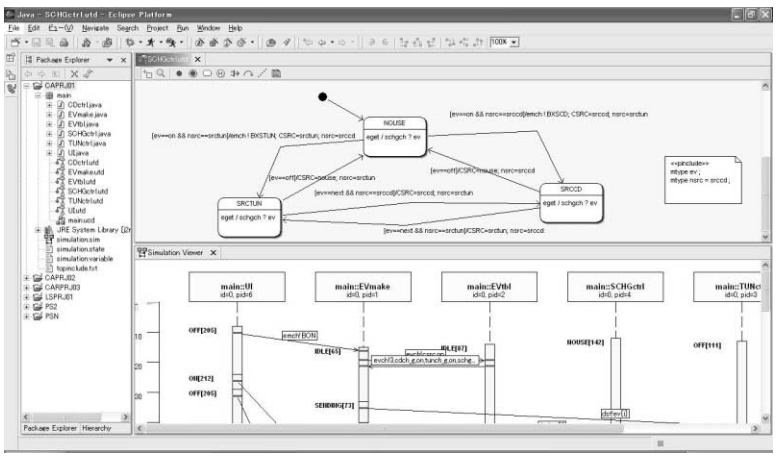

図2　検証支援ツール

は、Spinから出力された反例に基づいて、成立しない状況が発生するシーケンス図を出力する。この支援ツールは、特に、検証の工学的側面に焦点を当てている。組込みソフトウェア開発でのモデル検査技術による検証の役割や適用方法について明確にして、それを支援する機能を盛り込んでいる。現段階では、プロダクトライン開発や繰り返し開発プロセスを支援する機能などが実装されている。また、企業においてツールを使ってもらったり、事例に適用することによりさらなる検討を行っている。

４．まとめ

最近、組込みソフトウェアの品質が問題になることがよくあり、形式手法への期待が高まっている。一言に形式手法といっても様々なものがあり、それぞれ扱える記述、得意なこと、限界などの特性をもっている。これらの特性を見極めて、どの技術をどのように適用するかを決めるべきである。モデル検査手法は習得するのが容易で、かつ、使い初めてすぐにメリットが感じられ、比較的導入がしやすい技術であると考えている。まず、モデル検査手法から試してみるのも１つの手である。また、形式手法で取り扱う記述と現在の実際の組込みシステムで用いられる記述との間にはギャップがある。開発で用いられる記述を形式手法のものに置き換えるだけでなく、形式手法の記述や使い方を組込みソフトウェア向けにカスタマイズしたり、組込みソフトウェア開発で用いられる記述を形式化する必要がある。さらには、実際に適用してノウハウの蓄積も必要であろう。ノウハウは企業の武器であり、努力の賜であるから、公開はされないと思う。地道に企業内でノウハウの蓄積活動する以外には無いであろう。ひょっとすると海外の企業でそのような活動がされているかもしれない、という危機感を最近持っている。日本の国際競争力を維持、向上させていくためには、日本でも積極的にこのような技術に挑戦していくことが必要ではないだろうか。

参考文献

［１］E.M.Clarke and J.M.Wing: Formal Methods: State of the Art and Future Directions, ACM Computing Surveys, 1996.

［２］坂村健: μITRON4.0標準ガイドブック, パーソナルメディア, 2001.

［３］G.Holzmann: The SPIN Model Checker, Primer and Reference Manual,Addison-Wesley, 2004.

［４］青木利晃、片山卓也：RTOSに基づいたソフトウェアのためのモデル検査ライブラリ、組込みソフトウェアシンポジウム2005, pp.56-63, 2005.

［５］モデル検査技術によるUML設計検証, 情報処理学会学会誌, 47巻5号,pp.498-505, 2006.