SS2009「形式手法適用」WG ポジション・ペーパ

「形式手法」の「適用」について

中島 震
国立情報学研究所
nkjm@nii.ac.jp

1 はじめに

日常生活を支える社会基盤にソフトウェア技術が浸透するに伴って、ソフトウェアの信頼性や安全性への関心が高まっている。産業界では新たな技術の登場に期待する声が大きくなり、形式手法に注目しはじめた [18]。実際、モデル検査 [5] を使う試みが増えている。

ソフトウェア開発の技術発展を考えると、「形式手法」への期待は当然のことであろう。一方、商業誌でも「形式手法」が取り上げられる [17] に至り、もう一度、「形式手法」とは何であって、その「適用」とは何であるのかを考えたい。

本稿では、産学連携プロジェクト、大学院[1]の講義、産業界向けセミナー講演などの経験をもとに、いくつかの論点を示す。

2 モデル検査の適用

モデル検査 [5] は、時相論理、オートマトン、グラフアルゴリズムといった基礎技術からなる並行システムの自動検証アルゴリズムである。E. Clarke 博士ら創始者の 3 名が 2007 年度 ACM チューリング賞を受賞した。実用的なモデル検査ツールとして、SMV（あるいは NuSMV）、SPIN[8][19] などが知られている。

[1]総研大情報学専攻、JAIST/JJREX プログラム

モデル検査法は有限状態空間の網羅的な探索に基づく検証法である。基本的な考え方はきわめてわかりやすい。SMV も SPIN も複雑な言語構文を持たない。SPIN を学部生向けに教えるための教科書 [4] も出版されているくらいである。

一方、原理はわかった気がするが、うまく使えないという声も多い。理由は以下のようなところであろう [20]。すなわち、ソフトウェア開発過程でモデル検査ツールをどのように用いるかは自明ではない。適用工程を明確にすることが第一である。また、作業の入力となる情報の種類や検証の目的に応じて進め方が異なる。これらを明確にしないで、つまり、曖昧な目的のまま、とにかく使ってみる、という中途半端な姿勢が原因である。

モデル検査法の基本的な考え方は難しくはないが、通常は、検査対象システムを抽象化する必要があり、これが大切である。「抽象化支援検証（Abstraction Aided Verification）」という言葉が成立するくらい、検証と抽象化は密接な関係にある。C 言語や Java プログラムを対象とするモダン・モデル検査では、対象プログラムや検査性質を限定することで抽象化を自動的に行う方法の研究が中心になっている。検査したいプログラムが自動検証ツールの適用対象と合致すれば利用者はモデル検査の仕組み等を知らなくても良い。モデル検査ツールはバックエンドの検証エンジンにすぎない。

一方、ソフトウェア開発上流工程の生成物である分析や設計を表現したデザインを対象とする場合、

人手作業が主体となる。適用工程によって、作業の入力となる設計情報が異なり、モデル検査法を用いる目的も多様である。以下、作業を実際に行う観点からモデル検査の適用法を整理する。

最初に、モデル検査法は、いわゆる「検証」以外の目的で使われることを思い返したい。

- 検証（verification）：システムが検査性質を満たすことで正しさを保証
- 確認（validation）：システムが期待通りの振舞いを示すこと（あるいは、示さないこと）を確認
- 不具合発見（falsification）：システムが検査性質を満たさない不具合状況を発見

通常、「確認」と「不具合発見」はプログラムのテスティングで行う作業である。モデル検査の場合、基本探索アルゴリズムの網羅性を利用することで、個々のテストデータを与えるテスティングよりも系統性に優れた形で「確認」や「不具合発見」を行うと考える。

どのような目的でモデル検査法を用いるかによって作成するシステム記述の内容を考えなければならない。SPIN を用いる場合であれば、入力仕様言語 Promela の記述を作成する作業である。Promela 言語の使い方、典型的なイディオム、さまざまな抽象化法の具体的な実現法、等を知ることが大切である。さらに、モデル検査適用の目的や与えられた情報の種類によって作業手順が大きく異なる、という点に注意する。Promela の場合、仕様の全体を並行性の単位である Promela プロセスに分割して表現する。対象システムの何をプロセスに対応させるか、というモデリングも重要である。

本稿では、主として、作業の入力となる情報の種類（所与の情報）の違いから分類する。(1) 検証性質駆動、(2) コンポーネント駆動、(3) アルゴリズム記述駆動、に分け、(3) をさらに 3 つに詳細化した。

(1) 検証性質駆動作業　システムが持つべき性質の概要がわかっていて、その性質を満たす（あるいは満たさない）ためのシステム条件を方式検討することを目的とする。システムの概略像を具体化し検証性質が成り立つか検査する。概略からのシステム構築という設計作業を補助する手段として使う。一般に、システムが満たすべき性質を宣言的に書き表すことは難しく、熟練を要する。また、本質的に設計作業であることから、ここで分類した作業の中では最も難易度が高い。

(2) コンポーネント駆動作業　自然言語などの従来法で作成された「設計書」から出発する場合などに見られる作業形態である。設計書に書かれている非形式的な表現をもとに、システム記述を形式化（formalization）する。ついで、やはり設計書に書かれている種々の要件を時相論理式などで具体的に表すことでモデル検査（analysis）を行う。形式化過程で曖昧さ、矛盾点などを解消できる。デザイン段階のシステム記述を取り扱うので、システム構成要素の振舞い、要素間の情報連携など、いわゆるアーキテクチャデザインが関心事であることが多い。「確認」が主な目的であるが、デッドロックの有無など並行システムの安全性については「検証」も重要である。

(3) アルゴリズム記述駆動作業　前 2 者では、所与の情報が非形式的な記述であったが、より形式性の高いデザイン記法による記述から出発することがある。作業の観点から 3 つに細分類した。

(3A) デザイン記法　状態遷移ダイアグラム、フローチャート、などを表現したデザインであって、その処理手順が明確な意味を持つデザイン言語で書かれた場合にあたる。デザイン言語の意味定義が厳密であれば、形式的に定義された抽象化の方法を適用し、モデル検査ツールの入力仕様言語に変換すれば良い。抽象化をうまく行えば、モデル検査法を、「検証」、「確認」、「不具合発見」のいずれの目的にも利用することができる。抽象化を行わないで、デザイン記法を直接 Promela に変換する素朴な方法では、状態爆発によってモデル検査の作業がうまくいかないことが多い。

(3B) リバース　開発済のプログラムにモデル検査法を用いた事後検証（a posteriori verification）を行う場合がある。デザイン検証を行ったとしてもプログラミング過程で不具合が混入する可能性が残るのでプログラム検査が必要である。プログラムのテストや実行時に発生した不具合の原因を調べる目的でモデル検査法を用いるような場合がある。

一般に、プログラムをモデル検査しようとする場合、取り扱うデータ値やプログラム規模の大きさから逐次実行プログラムであっても状態数が膨大になる。プログラム開発時に作成した設計情報を参考にソースプログラムをリバースしてモデル検査に必要な情報を得る。リバース作業のなかで、アドホックな抽象化（近似）をせざるを得ない。典型的には、繰り返しループの回数を小さな固定値にする、大きな配列は代表要素だけにする、等の近似を行う。不具合状況を再現できれば作業は成功であるが、そうでない場合、確実な結論を導き出すことができない。デバッグの補完として行う場合、発生した不具合状況を再現するまで頑張ることになる。

(3C) プログラム検査　事後検証のひとつであって、特定のプログラムならびに検査性質に特化した自動解析ツールを使う場合である。C プログラムを対象とする SLAM、Terminator、VARVEL などがある。

検査したいプログラムが自動解析ツールの適用対象と合致すれば利用者はモデル検査の仕組み等を知らなくても良い。ツールによっては、取り扱うプログラム規模に制限があるなどの理由によって、事前処理を必要とする場合がある。また、自動解析ツールが用いている抽象化技法あるいは近似法によっては、見かけの不具合であるかの判断を行うために、解析結果に吟味が必要となる。一般にプログラムの自動検証は原理的に不可能である。注意して使う心がけが大切なことは言うまでもない。

まとめ　モデル検査法はソフトウェア科学の基礎技術が工学的な実用性まで結び付いた分野である。良いツールや解説書が入手でき、ソフトウェア工学の道具として使うことが可能になっている。一方、開発の現場での必須技術として広まるためには、まだ多くの関門が待ち受けている。本稿では、作業分類の観点から、モデル検査の適用について整理した。現在、ある産学連携プロジェクトにおいて、産業界の具体的な事例経験をもとに、モデル検査の適用ガイドラインを整理する活動と行っている。

3　形式仕様の適用

形式手法の源流は、1960 年代、IBM ウィーン研究所での VDL、VDM といわれている [18]。その後、多くの研究者が形式手法の研究に関わっているが、J.-R. Abrial 氏が果たした役割 [1] はきわめて大きい。「集合論を仕様記述に使う」こと、「産業界での適用なくして形式手法はあり得ない」ことを基本的な考え方とする。実際、Z 記法の考え方、B メソッドの推進、において多大な貢献をみることができる。上記の 2 つに VDM を加えて、形式手法黎明期から続く「モデル規範型（Model-oriented）」の形式手法と呼ぶこともある。本稿でも、これを採用し、以下、3 つをまとめて考察する。

VDM、Z 記法、B メソッドに共通する考え方は、「状態ベースの仕様（State-based Specification）」スタイルを用いた手続きの形式仕様を中心に考えることである。さらに、段階的な詳細化の考え方による「構築からの正しさ（Correct by Construction）」を採用する。正しい初期仕様から出発し、リファインメント関係を検証しながら、段階的に具体的な、すなわち、実行可能なプログラムとして表現可能な記述に詳細化していく。各手法の証明条件を見ればわかるのであるが、記述に対する正しさの基準は弱く、充足可能性（Satisfiability）と不変量の保存（Invariant Preservation）くらいといってもよい。もともと、リファインメントによって、正しさの基準を与えようとしていたからであろう。すなわち、仕様として与えられた抽象記述に対して、具体記述の正しさをリファインメント関係の形式検証によって保証する。正しさを保証する方法としてリファインメントが大切であり、これを抜きに形式仕様ならびに検証の技術を用いたとしても、何か中途半端な印象を残す。な

お、Z 記法は性質表現も自身の枠組で可能なので検証作業者が必要に応じた「正しさの基準」を追加することができる。

ところが、現在、研究者コミュニティから一歩外に出ると、リファインメントの側面を大きく取り上げることが少ない。形式仕様を作成することの重要性で留まっている。数学的な記法に慣れさえすれば、形式仕様の作成はそんなに難しくはないという声も聞く。本来はリファインメントこそが、ここで論じる形式仕様の本質ではなかったのか。これを抜きにした適用に意味があるのか。以下、リファインメントの視点から考察する。

基本的な考え方は、Hoare によるデータ・リファインメント関係 [7] である。3 つの手法は、いずれもフォワード・シミュレーションを採用しているが、各手法の意味定義の違いによって、証明条件は異なる。さらに、何を持ってリファインメントが正しいとするかも手法によって異なる。ただし、Back や Morgan のリファインメント計算（Refinement Calculus）と混同してはならない。リファインメント関係の定義に共通性はあるが、リファインメント計算では、抽象記述から具体記述を得る生成規則（Transformation Rules）として用いる。これに対して、ここで議論する手法では、リファインメント関係検査（Refinement Checking）に用いる。問題が簡単化されてはいるものの、リファインメント関係の検証はややこしい。自動検証を含めて簡便な方法が適用できるのは、関数的なフォワード・シミュレーションである。B メソッド、Z 記法、VDM の発展をリファインメントから眺めてみたい。いずれも、ある時代に「変身」していることがわかる。

B メソッドから Event-B へ　B メソッド [2] はリファインメントをベースとして実用的なプログラムの導出に成功した [14][16]。産業界で「構築からの正しさ」の技術を実証した唯一の形式手法といっても良い。最近、Event-B が提案され、リファインメントについて新しい世代を迎えている。

B メソッドが手続きの機能仕様を対象としていたのに対して、Event-B は要求モデリング等の上流工程での表現に用いるためにイベント概念を中心とした。並行性や非決定性の表現に力点をおく。B メソッド流の数学記法による式表現を用いるガードコマンド言語（Guarded Commands）といっても良い[2]。言語仕様を簡単化すると共に、リファインメントについては柔軟さを増した。B メソッドでは、VDM や Z 記法と同様に、抽象記述と具体記述は 1 対 1 に対応する。一方、Event-B では、1 対多の対応、新たなイベント導入が可能である。当然、リファインメントの証明条件は増え、また複雑になったが、フォワード・シミュレーション関係であることには変わりはない。

Event-B が柔軟さを取り入れた理由は、対象をオペレーションからシステムに広げたこと、ならびに開発上流工程の作業、すなわち、要求モデリングに用いることを意図したからである。B メソッドよりも柔軟なリファインメント技法を用いることで、「リファインメントに基づく要求モデリング（Refinent-based Requirement Modeling）」を達成した。すなわち、自明な初期記述から出発し、リファインメントによって段階的に要求モデルの記述内容を豊富にしていく。このリファインメントは、何も、実行可能なプログラムに向かう必要はない。リファインメントの最終結果だけではなく、リファインメント過程そのもの、すなわち、その過程で得た記述の全体と記述間のリファインメント関係の全体が、要求モデルを形成すると考える。記述の構成要素であるイベントは必ずしも実行可能なプログラムに対応するとは限らないことはリファインメントの観点からも重要な決定事項である。

B メソッドは Event-B に「変身」した。

Z 記法から Z 記法へ　Z 記法 [3][15] でのリファインメントは Spivey の方法 [24] がもとになっており、フォワード・シミュレーション関係である。さらに、例題の多くは、関数的なフォワード・シミュレーション（Functional Forward Simulation）で取り扱うことが可能である。一方、Woodcock たちは、産業界の

[2]Blocking Interpretation

問題への適用経験から、バックワード・シミュレーション導入の必要性を感じた [25]。さらに、Mondex の事例では、複数のリファインメントの組み合わせによって、はじめて可能になるような場合を示している。問題となった例は、システムの大域的な状態を明示する仕様からプログラムと同等のインプリメンテーションへのリファインメントである。

このような対象の広がりは、B メソッドから Event-B への変化と似た面がある。すなわち、Z 記法も B メソッドも、その誕生時点では、機能設計の方法がソフトウェア開発の中心であった。そこで、Z 記法のスキーマは、手続きの機能仕様やデータ構造の記述を想定したものであった。最近は、上流工程のデザインとして、Event-B と同様に、システム全体、あるいは要求モデルに関心が移ってきている、Z 記法もこれに対応する新たな方法を導入したということであろう。Event-B との違いは、バックワード・シミュレーション関係も併用すること、ならびに、プログラムの導出までカバーする意欲があること、の 2 点にあると思われる。

Z 記法はリファインメントの方法について「変身」した。

VDM から VDM++へ　歴史的には最初であった VDM を最後に説明するには、それなりの理由がある。上記の 2 つと発展の方向が少し異なるからである。VDM [10] は、状態ベース仕様の考え方に忠実な事前・事後条件による宣言的な関係を表現する書き方（implicit style）と、実行可能な言語要素に限定した書き方（explicit style）を提供する [3][6]。リファインメント証明条件[3]が定義されているのは前者であり、フォワード・シミュレーション関係を標準とする。

一方、Jones を中心として、1990 年代半ばより、産業界への展開と共に、後者の実行可能仕様に軸足を移し、VDM++ [23] が考案された。いわゆる、Formal Methods Light [11] である。リファインメントに基づく正しさの保証が、実行可能性 [4]を拠り所とする確認（Validation）に置き換わった。現在では、VDM++ が、少なくとも国内では、良く使われているようである [13]。実行可能デザインの記述言語として使うのでとっつきやすい。C や Java 等のプログラミング言語よりは記述の抽象度が高いものの、実行可能であることから、プログラムに近い詳細デザイン記述として使われる。有用な使い方は方式設計、アルゴリズムの確認であろう。

VDM は実行可能デザイン言語に「変身」した。

まとめ　Event-B と Z 記法が、プログラムに近いオペレーション機能仕様からシステム全体あるいは要求モデルに、記述の対象を変えていった。これに対して、VDM++は、従来からのプログラムに近い詳細設計の抽象レベルに留まることで実用化を目指しているように思える。VDM++をシステム全体あるいは要求モデルの記述に用いる方法を考えることも興味深い。現在、ある具体的な事例を対象として、VDM++を、このような目的で用いる実験を行うことを計画している。

余談であるが、自動解析可能なモデル規範型形式仕様言語として登場した Alloy [9][21] でリファインメント関係を導入する場合、関数的なフォワード・シミュレーションを用いる。また、Alloy は有界解析を行うので、これを使うには一工夫が必要である。何よりも自動解析できることが面白い。Alloy によって検証対象システムを記述して解析することも有用である。一方、Alloy を論理系と考えて、他手法の証明条件をエンコードして自動解析することも興味深い。有界解析[5]であることから、解析結果の吟味が難しい場合もある。しかし、自動解析の結果を示せるので、証明条件の検証を実感することができる。学習ツールとして有効である。詳細については別稿 [22] に譲る。

[3] Non-blocking Interpretation
[4] Blocking Interpretation
[5] Sound でも Complete でもない。

4 おわりに

形式手法として、モデル検査とモデル規範形式仕様の 2 つを取り上げ、適用について、いくつかの論点を示した。形式手法適用は従来のソフトウェア開発での信頼性確保の技術と違いが大きい。形式手法が数学的な基礎を持つこと以上に、形式手法が想定するパラダイムの違いが大きい。正しさを保証する基本的なスタンスが異なる。すなわち、プログラム実行に基づくテスティング技術に対して、形式手法では、抽象化や模倣関係（シミュレーション関係）といった一般のソフトウェア技術者に馴染みの薄い技術を基本とする。この違いが従来の、あるいは、現状の実践とかけ離れていることが、形式手法の実践を難しくしている要因と思われる。一方、パラダイムは、個別の技術要素として実現されるばかりではない。たとえば、オブジェクト指向パラダイムは手続き型プログラミング言語である C を使う場合でも工夫することで実践が可能である。この実践によって、良質の C プログラムを作成する経験は誰しも持つ。形式手法のパラダイムも、これと同様に、ソフトウェア技術者への示唆とすることが考えられる。形式手法への拒否反応の源である「数学らしさ」を軽減した説明も可能である。

今、形式手法適用を進めるために必要なことは 2 つある。すなわち、(A) 形式手法パラダイムの真髄に触れる良質の教育、(B) 個別の有用な事例、さらにそこから整理したパターンやガイドラインを中心とする知識集積、である。

参考文献

[1] J.-R. Abrial, S.A. Schuman, and B. Meyer. Specification Language. In *On the Construction of Programs, McKeag and McNaghten (eds.)*. Cambridge University Press 1980.

[2] J.-R. Abrial. *The B Book*. Cambridge University Press 1996.

[3] 荒木 啓二郎, 張 漢明. プログラム仕様記述論. オーム社 2002.

[4] M. Ben-Ari. *Principles of the SPIN Model Checker*. Springer 2008. (翻訳出版準備中と聞く)

[5] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press 1999.

[6] J. Fitzgerald, P.G. Larsen. *Modeling Systems*. CUP 1998. 荒木啓二郎他訳、「ソフトウェア開発のモデル化技法」、岩波書店 2003.

[7] C.A.R. Hoare. Prrof of Correctness of Data Representation. *Acta Inf.*, Vol.1, pages 271–281, 1972.

[8] G.J. Holzmann. *The SPIN Model Checker*. Addison-Wesley 2004.

[9] D. Jackson. *Software Abstractions*. The MIT Press 2006.

[10] C.B. Jones. *Systematic Software Development using VDM (2ed.)*. Prentice Hall.

[11] C.B. Jones. A Rigorous Approach to Formal Methods. *IEEE Computer*, Vol.29, No.4, pages 20–21, 1996.

[12] C.B. Jones. Scientific Decisions which Characterize VDM. In *Proc. FM'99*. pages 28–47, 1999.

[13] 栗田 太郎. 仕様書の記述力を鍛える. 日経エレクトロニクス, 2007.2.12 号, pages 133-152, 2007.

[14] 来間 啓伸. B メソッドと支援ツール. コンピュータソフトウェア, Vol.24, No.2, page 8-13, 2007.

[15] 来間 啓伸, B. Wolff, D. Basin, 中島 震. 仕様記述言語 Z と証明環境 Isabelle/HOL-Z. コンピュータソフトウェア, Vol.24, No.2, page 21-26, 2007.

[16] 来間 啓伸. B メソッドによる形式仕様記述. 近代科学社 2007.

[17] 中島 震. 形式手法の実像を知る. 日経エレクトロニクス, 2006.8.26 号, pages 123-142, 2006.

[18] 中島 震. ソフトウェア工学の道具としての形式手法. ソフトウェアエンジニアリング最前線 2007. pages 27–48, 2007. NII-2007-007J, (July 2007). `http://www.nii.ac.jp/`からダウンロード可能.

[19] 中島 震. SPIN モデル検査. 近代科学社 2008.

[20] 中島 震. ソフトウェア工学からみたモデル検査法. 第 22 回回路とシステム軽井沢シンポジウム, 2009.

[21] 中島 震, 鵜林 尚靖. Alloy: 自動解析可能なモデル規範形式仕様言語. コンピュータソフトウェア, Vol.26, No.3, 2009.

[22] 中島 震. モデリング記法解析における自動検査ツールの利用. (to appear) 2009.

[23] 佐原 伸, 荒木 啓二郎. オブジェクト指向形式仕様記述言語 VDM++支援ツール VDMTools. コンピュータソフトウェア, Vol.24, No.2, page 14-20, 2007.

[24] J.M. Spivey. *The Z Notation: A Reference Manual (2ed.)*. Prentice Hall 1992.

[25] J. Woodcock, J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice Hall, 1996.