10年後の形式手法

北陸先端科学技術大学院大学 情報科学研究科 教授
二木　厚吉

形式手法への現場からの期待が高まっているように見える。形式手法は長い歴史を持ち、現場からの期待の度合いも時とともに変化してきたが、最近の期待の高まりは主に以下の２点に起因している。

(1) 自動車の車載システムに代表されるように、人命に直接かかわるようなソフトウェアシステムが複雑化し、その安全性や信頼性を確保するためには形式手法が必要となってきた。

(2) 世界規模の情報ネットワークが重要な社会基盤となり、ソフトウェアシステムが、工業製品だけでなく、我々の生活を支えるさまざまな社会システムやビジネスシステムに浸透するにつれて、問題領域・要求・設計のモデル化・記述・解析に形式手法を用いる必然性が高まってきた。

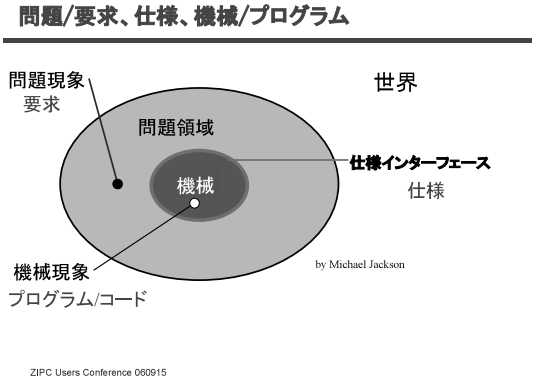
形式手法の現場への適用は新しい局面を迎えており、現場の技術者が形式手法の基本的な性質を理解し、その可能性と限界を自らの業務に基づき判断することが重要となりつつある。以下では、来るべき10年間の形式手法の進展を見極めるために有用と思われる、形式手法の基本事項について概説する。

１．形式手法とは？

「形式手法」はformal methodsの訳語である。この他にも、形式的方法、形式的手法といった訳が有り、訳さずにフォーマルメソッドとカタカナで綴ることもある。「形式的」という日本語が「内容が無く形式に堕している」というコノテーションが強いので、「形式化」の力を積極的に評価する立場のformal methodsの訳語としては形式手法またはフォーマルメソッドが適切であろう。

「形式（form）」という言葉・概念は、「形式は物事の本質を客観化した形」であり、また同時に、「形式は物事の本質を切り落とした形だけのもの」である、という二面性を持つ。この二面性は、理論と実践、科学と技術、基礎と応用などの間にある緊張関係と同様の性質をもち、時代や応用領域などに依存して色々な形で現れるが、時には深刻な対立関係に発展することもある。「形式手法は薫り高い研究成果を誇るが役には立たない」という批判は、システムの信頼性や安全性を確保する中核技術としての形式手法への期待が高まる現在でも、たびたび繰り返される。

形式手法の定義には、数学に基づくソフトウェア開発法、という非常に一般的であいまいなものから、仕様、設計、プログラムなどの正しさを数理論理（mathematical logic）に基づき厳密に検証できるソフトウェア開発法、といったより限定的なものまで、さまざまなものがある。形式手法の基本的な要件を何に求めるかについては、研究者の間にさえ共通の認識があるとはいえない。形式手法に関する最も包括的なウェッブサイトであるFormal Methods Virtual Library（http://www.afm.sbu.ac.uk/）には95に上る記法・方法・ツール（individual notations, methods, tools)が掲載されている[1]。形式手法は、ある人にとってはSPINシステムによるモデル検



1 以下の記述に現れる方法、言語、ツール名の詳細についてはこのウェッブページを参照されたい。

査であり、他の人にとってはCafeOBJ言語システムやRAISE言語を使った問題領域（domain）や要求（requirement）のモデル化と記述であるといったように、広範な領域をカバーしつつある。最近では、形式手法はソフトウェアシステムだけを対象とするのではなく、ハードウェアシステム、バイオシステム、社会システムなどの広範なシステムを対象とする横断的な汎用科学技術（generic science&technology）である、とする傾向が顕著になりつつある。

2．形式手法の源流

形式手法の最も一般的な定義は「数学的な手法に基づく科学的・分析的なシステムの構築法」とすることが出来る。計算機の誕生が「数学的な計算（数値計算）を機械で行う」ことに動機付けられていることを考えれば、計算機とそれを動かすソフトウェアの発展は数学的な手法に裏打ちされたものであった、と言うことが出来る。しかし、情報ネットワークの急速な進展に伴い、ソフトウェアシステムとして実現される機能は多様化し、数値計算に代表される「計算」はそのごく一部になってしまい、ソフトウェアシステムは我々の生活に必要なあらゆる機能を実現することが期待されるようになっている。これに伴い、ソフトウェアシステムの科学的・分析的な構築を支援する数学的な手法も、純粋な計算を支援するものから、複雑なシステムの構築に必要な論理的な推論や検証を支援するものへと変容を遂げてきた。以下では、現状の形式手法への発展を可能とした基盤技術を概観する。

2.1．プログラム検証技術

プログラム検証のアイディアは電子計算機の誕生当初の早い時期からあったとされる。電子計算機上で動作し人間が望む仕事を自動的に行うプログラムは、人類にとってまったく未知の新しいものであり、その動作の正しさを納得するためには数学的な検証が必要である、と当事者が考えたであろう事は容易に推測される。単一プロセッサから他プロセッサへ、単一処理から多重分散ネットワーク処理へと計算機システムが複雑化するのに対応して、プログラム検証技術はソフトウェア技術の中核技術として発展を遂げている。プログラム検証技術はいまだに活発な研究活動が展開されており、その可能性を結論的に述べるのは難しいが、以下の点を指摘しておく。

プログラム検証の完全な自動化は出来ない：プログラム検証の完全な自動化の問題は、原理的に機械には解けない問題（決定不能問題；unsolvable problem）を含むので、すべてのプログラムを自動的に検証するプログラムを開発することは不可能であると認識すべきである。応用領域を限定したり、プログラムの形を制限したりすることである程度の自動化が期待できるが、プログラムの自動検証は決定不能問題と遭遇する可能性を常にはらんでいる、という事実を前提にすることが大切である。

プログラム検証は人間が適切な仕様を与えることで意味を持つ：プログラム検証はプログラムが仕様の通り動作していることを示すことであり、検証の自動化の可能性は、ユーザーがどのような種類の仕様をどのように表明するかに依存して決まる。検証の自動化を考える以上に、適切な仕様の与え方を定めることが重要である場合が多い。

正しいプログラムを仕様から自動生成する「生成による検証（proof by generation）」の技術が有望な分野が多い：プログラムの開発に先立ち仕様を適切に与えることが、システム開発全体の効率を高めるためには重要である。コーディングの前に仕様が開発されているのであれば、仕様から正しいプログラムを自動的に生成するほうが、コーディングの後でプログラム検証を行うよりは効率的であり、より高い信頼性を獲得できる可能性が高い。MDA/MDDなどは、こうした洞察に基づく開発方法論である。

2.2．形式仕様言語（formal specification languages）

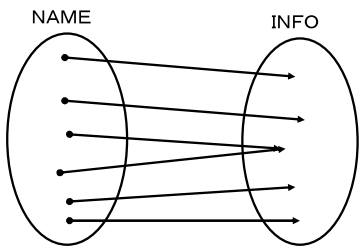
モデルや仕様を記述するための言語としては、実務家の間ではUML（Unified Modeling Language）がデファクトとして受け入れられつつある。しかし、UMLに形式手法を適用し検証を行おうとすると、UMLに厳密な意味を付加す

る必要があり、UMLの形式化（またはUMLの数学的な意味定義）の研究が盛んに行われるようになってきた。

形式手法の前提となる数学的な意味定義を伴った仕様言語の研究開発は、UMLなどの図式言語の研究開発とは独立に行われてきた。以下がその代表的なものである。

VDM, RAISE, Z, B：集合、写像、関数といった数学的な概念にもとづき、システムの仕様を記述する形式仕様言語。数学的な意味は型付けされた一階述語論理（typed first order predicate logic）を用いて与えられる。モデルベース形式仕様言語として分類されるが、これは集合、写像、関数などの数学モデルを用いてシステムの仕様を定義する立場から来ている。仕様記述のための数学記法（mathematical notation）であり、計算機上で実行可能であるように設計されていない。VDMとZは、ソフトウェアシステムの仕様を数学的に記述するための言語として、もっとも早い時期に設計開発された言語である。仕様記述者とプログラム開発者の意思疎通の記法として、いくつかの実プロジェクトで使用された実績を持つ。最近では、検証システムとの連携により仕様の検証を図ろうとする研究が盛んに行われるようになった。

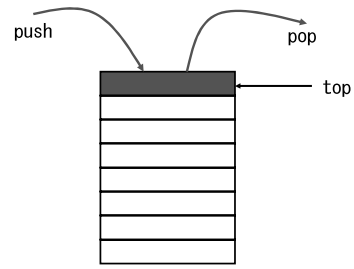
形式仕様の例（1）
Zを用いたデータ辞書の仕様 － 図



ZIPC Users Conference 060915

OBJ, CafeOBJ, Maude, CASL：システムの仕様を、幾種類かのものの集まり（ソート（sort）と呼ばれる）とその上に定義されたいくつかのオペレータ・演算・関数（operator, function）を基本単位として記述する形式仕様言語。ものを多種類に分けて分類することで型付けが導入され、多ソート代数（many sorted algebra）が仕様の数学的なモデルとなることから、代数仕様言語（algebraic specification languages）と分類される。数学的な意味は仕様が定義する代数を規定することにより与えられる。オブジェクト指向の先導概念である抽象データ型（abstract data type）の形式仕様言語としてスタートし現在では動的システムをも含む一般的なシステム用の形式仕様言語に発展している。OBJ, CafeOBJ, Maudeは仕様の意味を限定するために等式（equation）だけをつかう立場を基本とし、その等式を書換システムと解釈することで実行可能仕様を実現している点が、VDM, Zなどの実行可能性を前提としない形式仕様言語と異なる。実行可能性を有することで検証システムを自然に内包した仕様言語システムが実現できるという大きな特徴がある。

形式仕様の例（2）
CafeOBJを用いたスタックの仕様 － 図



ZIPC Users Conference 060915

2.3．システム検証技術

検証技術はシステム技術一般に広く普及し多様化している。ハザード（hazard）や失敗の可能性または事後の解析技術として発展してきた失敗木解析（fault tree analysis）の技術、統計的・確率的な信頼性解析技術などのいわゆる信頼性工学の手法には、分野に特化したものも含めて、多くの手法・方法が存在する。ソフトウェアが最終的には論理的・数学的な構造物であるという前提のもとに、主にソフトウェア技術の分野で発展してきた検証技術として、定理証明（theorem proving）とモデル検査（model checking）がある。

定理証明技術：定理証明という名前が示すよう

に、「数学の証明を計算機上で実行する」という動機からスタートした技術である。数学の証明の過程を形式化しそれを計算機で支援させるシステムがいくつも開発されている。こうした研究の成果として、100年以上のあいだ未解決で1976年に初めて解決された４色問題が、定理証明システムCoqを使って検証されたというニュースが2004年末に話題になった。数学的に厳密な検証を計算機で支援させ、出来る限り自動化を図るという研究は、計算機科学におけるもっとも挑戦的なテーマの一つである。定理証明システムは、対話型検証システム、証明支援システムなどと呼ばれることもあり、ケンブリッジ大学・ミュンヘン工科大学のIsabelle/HOL、フランス国立研究所・パリ大学のCoq、スタンフォード研究所のPVSといったシステムが有名である。数学的な検証過程を計算機で支援するので、検証すべきシステムを有限状態システムとしてモデル化する必要がある、といった制限は無い。システム検証技術において定理証明技術を役立たせるためには、(1) システム開発に役立つ十分に厳密なモデル化・仕様記述技術、(2) 数学の定理証明ではなくシステム検証向きの証明過程の定式化とシステム化、(3) システム検証に向けてユーザーインターフェースの開発、などの問題を解決する必要がある。こうした課題の克服を目指して、CafeOBJやMaudeなどの実行可能形式仕様言語システムを発展させて、システム検証向きの対話型検証システムが研究開発されている。

モデル検査技術：モデル検査の「モデル」は、通信ソフトウェアや論理回路（半導体回路）の検証技術から発展してきた「有限状態モデル」を意味する。モデルという言葉は万能の意味合いが強いので、モデル検査におけるモデルが有限状態モデルであるという事実は注意を要する。モデル検査技術は、現場で使われ多くの現実の問題を解決しているという意味において、現時点で最も成功した検証技術である。有限状態モデルに限定することで、一般的な検証では困難なLTL（Linear Temporal Logic）やCTL（Computation Tree Logic）といった時制論理式（temporal logical formula）で記述された性質の検証も自動化できるという特徴を持つ。またBDD（Binary Decision Diagram）などの巧妙なデータ構造を採用することで、大きな状態集合を持つシステムを合理的な時間内で検査できるようにした点も、大きな技術的なブレークスルーであった。モデル検査がもたらした技術的な成果は大きいが、(1) 有限状態モデルでは対応できないシステム検証問題は多く存在する、(2) 現状のモデル検査システムでも対応できないほど大きな有限状態をもつシステムも多く存在する、(3) モデル検査は反例の発見には有効であるが検証・証明には限界がある、といった諸点は認識しておいたほうが良い。モデル検査と対話型検証を統合しより強力な検証技術を開発することが、当面の最重要の技術課題である。

３．形式手法の展望

形式手法を、個別の手法やツールの集合体としてではなく、統一的な分野として捉えるためには、VDM, Z, OBJ, CafeOBJといった形式仕様言語（formal specification languages）による形式仕様の開発が形式手法の中心にあるとする見方が適切と思われる。こうした立場からは、形式手法の最大の効用がシステムの信頼性や安全性のできる限り客観的な解析と検証であるし、形式手法の基本要件を以下の2点とすることが多い。

(1) 意味が数学的に定義された形式言語によりシステムのモデルが記述される。
(2) その記述が数学的に解析・検証される（数理論理に基づく論理的な解析・検証法がある）。

形式手法は、数学的なモデル化法と解析法に基づく手法であるが、それはまた計算機科学とくにプログラムとソフトウェアに関する理論や技術から発展したものである。すなわち、以下のような理論や技術は、従来の数学の対象ではなかったが計算機科学の中で発展し、それらに基づきシステムの汎用的なモデル化、記述、解析・検証を目的とする形式手法が整備されてきたといえる。

- **プログラムとプログラム言語：**手順や手続きを厳密に記述し、その記述を解析・検証するための理論・言語・方法論。
- **並行分散システム：**複数の主体から構成されるシステムを記述し解析するための

理論・言語・方法論。

■ **形式仕様と汎用システムモデル：**システムの要件・仕様を論理式として記述しそれを解析・検証するための理論・言語・方法論、ならびにオブジェクトモデルやUMLに代表される汎用的なシステムのモデル化法・記述法。

今後５年、10年のスパンで形式手法の現場への適用は、以下のような二つの異なった領域に二極化していくことが予想される。

(1) プログラムコードを直接の対象として、その正しさを解析・検証する分野。
(2) プログラムコードを作成する以前に、またはプログラムコードを作成することを前提とせず、問題、要求、設計の仕様を記述しその正しさを解析・検証する分野。

3.1. プログラムコードの検証

プログラムコードの正しさの検証は計算機の誕生以来の研究テーマである。先に述べたとおり、あらゆる種類のプログラムの正しさを統一的に検証することが不可能なことは、理論的にも技術的にも証明されている。しかし、ある特徴を共有するプログラムのクラスを対象とする検証システムや、人間と検証システムが対話しながらプログラムを検証する対話型検証システムは着実に進歩してきた。とくに、プログラムを有限状態機械としてモデル化し到達可能な状態をすべて検索することで、プログラムが望みの性質を持つことを検証するモデル検査技術は、プログラムの検証技術に大きな進歩をもたらし、その適用領域を拡大させている。以下にプログラムコードの検証技術の今後を予測する上でのいくつかのポイントを掲げる。

■ マイクロソフトがプログラムコードの自動検証技術に大きな研究投資をしていることからも推測できるとおり、この技術は適切に適用することで、多くのソフトウェア開発現場において大きな効果を発揮することが期待される。

■ プログラムコードの自動検証の中核技術は、モデル検査と静的解析（static analysis）技術である。モデルチェッキングシステム（たとえばSPINなど）の使い方を習得するだけではなく、こうした技術の背景にある理論も含め、それらの対する深い理解を持った技術者を養うことが大切である。

■ PVS, HOL, Coqといった定理証明システム（theorem proving system）を使った対話型のプログラムコード検証技術を通常のソフトウェアの開発現場で使うのは現実的でない。しかし、こうした技術を使って初めて検証できるプログラムコードも多くある、ということは認識すべきである。

■ プログラムの全自動検証は魅力的であるが、完全な全自動検証は原理的に不可能であり、何らかの条件をつけて限定的にしか実現できないと再認識しておくことが重要である。

■ 領域を特定したプログラム生成による検証（verification by generation）の手法は、適切に適用することで効果を発揮する可能性が高く、適用可能性を常に検討すべきである。

3.2. 問題・要求・設計の記述と検証

システム開発の上流工程の問題仕様、要求仕様、設計仕様に誤りがあると、それがコード開発、テスト、運用といった下流工程において甚大な被害を及ぼす。システム開発においては、なるべく上流で仕様を体系的かつ客観的に記述し、それを解析・検証する技術が重要である。また電子社会の進展に伴い、電子社会として実現される可能性のある社会行政システムやビジネスシステムの客観性の高い記述を体系的に開発し、その記述を通してシステムの健全性や安全性の解析や検証をする技術がますます重要になりつつある。このように、形式手法は要求や設計そのものを記述し解析・記述する汎用技術としての重要性を増しつつある。以下に要求仕様や設計仕様の記述と検証に関連した分野での形式手法の展開を予測する上でのポイントを掲げる。

■ UMLなどのオブジェクトモデリングを支援する言語やシステムは、システムの要求や設計の記述を支援する技術として発展し、Modeling Languageがシステム開発において重要であるという認識を高め

たという功績は大きい。しかし、UMLで記述されたシステムの性質の解析や検証には、形式手法を導入することが不可欠である。

- 要求や設計の検証技術が実用化されれば、システム開発におけるその効果はコードレベルの検証よりもむしろ大きい。しかし、この分野でモデルチェッキング技術により期待できる自動化は限定的であり、対話型の検証技術が必要であろう。特に、要求仕様の開発においては、モデル化→記述→検証→モデル化→記述…というサイクルの中で現実の問題の文書化・定式化を進めるためには、対話型のモデル化・検証技術が不可欠であろう。
- 要求や設計の検証は、本質的に困難な問題を多く含むので、この分野における形式手法の現場への適用は、モデル化と記述から始まり、検証に関しては特に信頼性と安全性への要求が高い分野に限定的となる可能性が高い。
- 問題・要求の分析と記述は問題領域そのものの分析と記述であり、必然的に多くの分野にまたがる活動であるので、経営学、経済学、社会学などの関連する分野との連携が重要となる。

4．形式手法の教育と人材養成

形式手法はコストの高い技術であるといわれる。これは、この技術を使うために、

(1) データ構造とアルゴリズムの理論、
(2) 述語論理・等式論理・時制論理などの数理論理学の素養、
(3) 関数プログラミングなどの現代的なプログラミング技術、
(4) コンパイラ技術などのプログラム言語の基本技術、
(5) オブジェクトモデリングなどのモデル化技術や形式仕様言語を含む仕様開発の基本技術、
(6) 対話型検証システムやモデル検査システムの基礎理論やそうしたシステムを使う能力、

といった基礎基盤的な知識や能力を持った人材を必要とするからである。すべてのソフトウェア技術者にそうした能力を要求するのは現実的ではないにしても、適用領域ごとに将来必要となる技術の動向を見極めるためには、上に掲げたような基礎的な素養を有し、各々の問題領域ごとに形式手法の適用可能性とその限界を見極めることができる人材を育てることが大切である。