未来に向かって

不確かさを考慮したソフトウェア・テスティングおよび形式検証

石川冬樹（国立情報学研究所） 來間啓伸（(株)日立製作所） 中島 震（国立情報学研究所）

ソフトウェアにおける不確かさ

ビッグデータ分析や機械学習のコンピュータ・プログラムは，未知の相関関係や予測データを計算結果とするので，正解がない，あるいは結果の正しさ確認に相当な労力を要する．テスト不可能[1]といえる．また，近似解を効率良く求めるために乱択アルゴリズムを用いると，計算結果やその収束性が確率分布の影響を受けて変動し，テスト実行の成否が一目瞭然とはならない．

昨今 IoT（Internet of Things）が話題になっているが，これらのソフトウェアは従来にない性質[2]を持つ．たとえば，自身の制御下にない構成要素や他者が提供する構成要素と動的に連携したり，統計的な揺らぎを伴う物理現象に起因するデータを入力したりする．このとき，「特定の想定環境でシステムが要求を満たす」ことを形式検証により確認しても，期待される信頼性を保証することが難しい．

上記はいずれも，ソフトウェアのディペンダビリティ達成において，不確かさを扱うことの必要性を示す．本稿では，テスティングと形式検証それぞれに関して，不確かさを扱う研究の動向を紹介する．

不確かさとテスティング

➡ テスティングにおけるオラクル

ソフトウェア・テスティングは，プログラムが欠陥を含むことの確認を目的とする．検査対象プログラムへの入力を $\{x_1, x_2, \ldots\}$ と与えたときに，各実行結果が $\{y_1, y_2, \ldots\}$ であったとする．各入力に対する正解 $\{c_1, c_2, \ldots\}$ が既知ならば，実行結果と一致するか（$y_i=c_i$ であるか）照らし合わせる．一致しなければプログラムに不具合がある．一致したとしても，膨大な入力の可能性における一部しか扱えず，不具合がないとは結論できない．入力の定め方を工夫することが重要になる．なお，定めたテストの妥当性を判断する基準，上記であれば既知の正解を得た根拠（それらをどういう情報からどう導いたか）をオラクルと呼ぶ．オラクルは，同等の実行結果に対して常に同じ判断を導くべきである（そうではなくまるで偶然でテスト成否が決まるかのようなテストは Flaky といわれる）．

➡ 疑似オラクル

各入力に対する絶対的な正解が不明のとき，疑似オラクルとして，検査対象プログラムとは別の手段で得られた解を，相対的な正解 $\{r_1, r_2, \ldots\}$ として用いる．現実的に実現可能で，妥当な正解を決めたものといえる．プログラム改版に際して実施する回帰テストにおいて，ある入力に対して，改版前と同じ結果が得られれば，改版後のプログラムに新たな欠陥が混入していない，と考えることに似ている．

相対的な正解を生成する方法は，いくつかある．N バージョン・プログラミングは，同じ機能仕様のプログラムを複数開発する手法である．複数の開発チームが異なるプログラミング言語を用いるなどの工夫によりプログラム開発の独立性を高める．これらのプログラムの計算結果は互いに疑似オラクルとなる．

メタモルフィック・テスティングは，検査対象プログラムの特徴を踏まえ，相対的な正解を生成する方法である．ある入力 x_i に対する出力が $y_i=\mathrm{f}(x_i)$ であるとする．このとき，「入力 x_i に対し変換 T_I を施した $T_I(x_i)$ をプログラムへの入力としたときに得られた出力 $\mathrm{f}(T_I(x_i))$ は，元の出力 y_i に変換 T_O を施した $T_O(y_i)$ に一致する」といえるような T_I と T_O を理論的に定める．たと

えば，ある行列計算では「入力となる行列の1列目と2列目を入れ替えると，出力となる行列も1列目と2列目が入れ替わるはず」と言えたとする．このとき，ある入力に対して出力を得た後，その入力行列の列を入れ替えたものに対する出力を得て，元の出力結果の列を入れ替えた行列になるかを確認する．そうならない場合，プログラムに不具合があると見なせる．メタモルフィック・テスティングの方法は，教師あり分類学習のサポート・ベクタ・マシンなどに適用されている（たとえば文献3)）．

➡ 統計的オラクル

プログラム実行が確率分布に従う変数に依存する場合，正解が一意に決まらないので，計算結果を統計的に解釈する．同一テスト条件下で繰り返し実行して得た計算結果の集まりを標本と見なし，統計的な仮説検定を用いれば，与えられた確信度で欠陥の存在を推定できる．ある計算結果の母平均 μ_0 を正解値，標本平均μを検査値とする．帰無仮説を「不具合がない（$\mu_0 = \mu$)」，対立仮説を「不具合がある（$\mu_0 \neq \mu$)」として，背理法を用いる．帰無仮説の仮定下で計算した標本統計値がきわめて稀にしか現れないことを示し，不具合があると推定する．統計的オラクルは乱択アルゴリズムを用いたプログラムや確率離散シミュレーションに適用されている．

➡ 偏りのある入力データセット

不具合を顕在化させやすい「扱いが難しい入力データ」を用いると，テスティングの効率が向上する．たとえば文献3) では，教師あり分類学習の特徴を利用して，正しく分類することが難しいデータ点を系統的に生成している．統計計算を行うプログラムに対しては，不具合の検出効果が大きい「外れ値」の自動生成が重要だろう．

不確かさと形式検証

形式検証では正しさが既知であることを前提とする．不確かさを表すために，確率概念を設計仕様に導入し，統計的な観点から正しさを論じる方法が関心を集めている．

➡ 構築からの正しさと確率概念

形式手法の目的は，正しさの証明とプログラム構築を同時に行う「構築からの正しさ（correct by construction)」と呼ばれる開発法の提供である．具体的な形式手法として，リファインメントに基づくBメソッドやEvent-B[4] が提案されてきた．Bメソッドは，機能仕様から出発し，段階的詳細化に基づいて，検証済みプログラムを得る．パリ地下鉄システムの開発をはじめとして，欧州を中心に産業界で実適用されている．

Event-BはBメソッドの発展であり，開発上流工程のシステムモデルを段階的に構築する手法である．ソフトウェア自身あるいは制御の対象となる機器だけでなく，外部環境を含めて，非決定的に発生するイベントによってシステム全体がどう振る舞うかを記述し検証する．現在，形式検証の技術は成熟段階にあり，Event-Bにおいても，仮定や既知の性質から新たな性質を示す定理証明，可能なイベント実行列を網羅的に調べるモデル検査，いずれについても強力なツールが利用可能になっている．

Event-Bに限らず形式手法では，開発上流工程の抽象的な仕様を表現するとき，非決定的な振舞いと捉える．「じゃんけん」における3つの手のように，可能な選択肢をすべて考え，その上で安全性などが保証できるかを調べる．現実世界ではいずれ「あいこ」が停止して勝敗がつくことを期待できる．ところが，出す手を非決定的に選択するモデルでは「あいこ」が無限に繰り返される可能性を排除できない．このような現実とのギャップを埋めるため，非決定的選択に確率を与える拡張法が提案されている．たとえば，文献5) は，Event-Bを拡張し，各選択肢が選ばれる確率を与えるようにすることで，「ほぼ確信できる停止性（almost-certain termination)」を検証する方法を与えた．選択確率の追加を非決定的選択のリファインメントとして扱う．これによって，Event-Bの標準的な方法の枠組みの中で形式検証が可能になった．

➡ 確率系を対象とするモデル検査

一般的に，何かの事象の発生確率を考え，望ましい事象の発生確率が一定値以上であるといった性質を検証したいことがある．このような検証問題を，従来のモデル検査の発展形と捉え実現した技術として確率的モデル検査があり，PRISM などのツールが知られている．確率的モデル検査では，システムの振舞いを表す状態遷移を，マルコフ連鎖やマルコフ決定過程を用いて表現する．通常のモデル検査と同様に時相論理を用いて「常に」「いつか」「までは」といった検証性質を表現し，その性質が満たされる確率が一定値以上かを検査する．

他方，マルコフ連鎖などのモデルを構築して事象の発生確率を求めるのは計算コストが高い．これに対し，高速に行える離散シミュレーションを多数回繰り返すことで，事象の発生確率が一定値以上であることを与えられた確度で示す方法がある．すなわち，仮説検定を行うもので，この方法を統計的モデル検査という．

➡ 環境の不確かさへの対応

不確かさを考慮すべき場合として，ソフトウェアが外部環境と強く関係する状況がある．従来は，開発時に，ソフトウェアに影響を与える外部環境に関する想定あるいは仮定を明確にし，その下で検証を実施した．しかし，外部環境が実世界のさまざまな要素を扱う状況では，あらゆる想定を尽くすことが難しく，また外部環境の変化も激しい．開発時に特定条件下で検証を行うだけでは不十分である．

実行時検証（Runtime Verification）と呼ばれる技術では，ある時点までの実行経路を基に，検証対象の性質が満たされたか，破られたか，その時点ではどちらであるか断言できないか，といった判断を下す．この方法は，定理証明，モデル検査，テスティングに対する第 4 の検証手段と位置づけられている[6]．実行列の監視機構に加えて，検証判断を行うモニタを検証性質から自動生成することが技術的な肝となる．

外部環境の変化が検出された際に，外部環境を表すモデルを更新し，このままシステムを作動させてよいかを確認する考え方もある．この場合，現時点の状況よりも未来に起こるさまざまな可能性をモデル検査により網羅的に調べる．従来開発時に構築し検証していたモデルを実行時にも更新し活用する Models@run.time と呼ばれるアプローチである．モデル検査をシステム実行時に高速実行することが重要な課題となる．

モデルによるディペンダビリティ達成

本稿では，不確かさのあるソフトウェアのディペンダビリティ達成に向けて，テスティングと形式検証それぞれにおける動向を概観した．いずれにおいても，「正しさ」の基準や，その確認方法がこれまでとは変わっていることが見てとれる．従来の数理論理やオートマトンに基づく基礎理論だけでなく，確率や統計に基づいたアプローチが重要になっている．いずれにしても，これまで以上に，「システムをいろいろ動かしてみる」だけでディペンダビリティを確保することは到底できず，しっかりした仮説やモデルに基づいたアプローチが重要となる．

参考文献

1) Weyuker, E. J. : On Testing Non-testable Programs, Computer Journal, Vol.25, No.4, pp.465-470 (1982).
2) 中島　震：共通理解フレームワークとしてのサイバー・フィジカル・システム，情報処理学会ソフトウェア工学研究会第 190 回（2015）.
3) Nakajima, S. and Bui, H. N. : Dataset Coverage for Testing Machine Learning Computer Programs, APSEC 2016, pp.297-304 (2016).
4) 中島　震，來間啓伸：Event-B ―リファインメントに基づく形式手法，近代科学社（2015）.
5) Hallerstede, S. and Hoang, T. S. : Qualitative Probabilistic Modelling in Event-B, iFM2007, pp.293-312, Springer (2007).
6) Leucker, M. and Schallhart, C. : A Brief Account of Runtime Verification, The Journal of Logic and Algebraic Programming, Volume 78, Issue 5, pp.293-303 (2009).

（2017 年 4 月 29 日受付）

石川冬樹（正会員） f-ishikawa@nii.ac.jp

国立情報学研究所 コンテンツ科学研究系 准教授．電気通信大学情報理工学研究科 客員准教授．博士（東京大学，情報理工学，2007 年）．形式手法や最適化技術を中心に，ソフトウェア工学およびその先端システムへの適合に関する研究に従事．

來間啓伸（正会員） hironobu.kuruma.zg@hitachi.com

（株）日立製作所 研究開発グループ 研究員．2006 年総合研究大学院大学複合科学研究科情報学専攻博士課程修了．博士（学術）．主としてソフトウェア工学と形式手法の研究に従事．

中島　震（正会員） nkjm@nii.ac.jp

国立情報学研究所 情報社会相関研究系 教授．総合研究大学院大学教授（併任）．博士（東京大学，学術，2000 年）．ソフトウェアのディペンダビリティ，CPS に関する研究に従事．2001・2015 年度本会山下記念研究賞受賞．