13. 手法の概要および選択法に関する情報

想定読者	(1)プロジェクト管理者 (2)開発者
目的	7 章「手法の選択方法」で対象としなかった手法以外の代表的な形式手法に関する概要を整理し、目的に即した形式手法を特定するための情報を提供する。また、モデル検査の性質検証に利用することの多い、LTL と CTL について整理する。
想定知識	ソフトウェア開発に関する基礎
得られる事	● 主な形式手法の概要

13.1. その他の代表的な形式手法

本ガイダンス本編の「第 7 章 手法の選択方法」では、実システムへの応用実績の多い形式手法について具体的な特徴や実績を示した。本節では、それらに準じる主要な形式手法について概要をまとめる。

手法名	年代	開発組織	説明	関係サイト
PVS	1980 年代	米国・スタンフォード研究所(SRI)	● PVS(Prototype Verification System)は形式仕様記述と検証のためのシステムである。 ● PVS は、 * 仕様記述言語(関数、集合、リストなど) * 事前定義された定理 * 型チェッカー * 対話的定理証明器 * シンボリックモデル検査器(BDD) などからなる。 ● 仕様記述言語は、高階型付き言語に基づいている。表現力のある言語と強力な自動推論により、大規模な形式化と証明に対応できる。 ● PVS は SRI における 25 年間の形式手法ツール開発の知見に基づいている。	PVS Specification and Verification System http://pvs.csl.sri.com/
FDR2	1990 年代	Formal Systems Europe Ltd.(Oxford University)	● FDR2(Failures/Divergence Refinement 2)は CSP(Hoare's Communicating Sequential Processes)で記述されたモデルのプロパティに対するモデル検査ツールである。 ● 状態遷移のリファインメント関係、デッドロックやライブロックを検証できる。	http://www.fsel.com/index.html
Alloy	1990 年代	MIT (Daniel Jackson)	● Alloy は形式仕様記述言語と自動解析ツールを含む形式手法である。言語は、ソフトウェア設計のための軽量なモデリング言語である。記述した仕様に対して、Alloy Analyzer を用いて完全に自動な分析ができる。また、可視化機能によって、解法と判例を把握しやすい。 ● Oxford 大の Z の表記法と、Pittsburgh 大の SMV の解析に啓発されて作られたとされる。 ● Alloy は関係代数に基づいており、ドット演算子による join 演算は RDB の join と類似している。 ● SAT に基づく自動解析を特徴とする。	Alloy Community http://alloy.mit.edu/community/

SAL	2000年代	米国・スタンフォード研究所(SRI)	● SAL(Symbolic Analysis Laboratory)は、抽象化、プログラム解析、定理証明、シンボリックモデル検査などのツールを組み合わせたフレームワークである。 ● SAL の重要な部分は、遷移システムを記述する中間言語である。この中間言語は、他のモデリング言語やプログラミング言語への変換や、他の解析ツールへの入力となることを意図している。もともとは、状態遷移図と Murφ や SMV といったモデル検査ツール、さらには不変条件の生成ツールとの間の中間言語を意図していた。実際のところ、SMV への変換の負担から独自のモデル検査器を作っている。 ● モデル検査ツールは、BDD を用いたシンボリックモデル検査と、SAT を用いた有界モデル検査、および試験的な"Witness"モデル検査、さらには SMT ベースの"Infinite"有界モデル検査ツールである。検証性質は LTL で表現する。 ● シミュレータ、デッドロック検査ツール、自動テスト生成ツールも具備している。	http://sal.csl.sri.com/
CBMC	2000年代	Carnegie Mellon University	● CBMC は ANSI-C と C++プログラムの有界モデル検査ツールである。 ● 配列の境界（バッファオーバフロー）、ポインタの安全性チェック、例外とユーザの指定したアサーションについて検証できる。検証は、プログラム中のループ展開と論理式への変換によりなされる。	http://www.cprover.org/cbmc/
JavaPathfinder	2000年代	NASA Ames Research Center	● JPF(Java Pathfinder)は Java のバイトコードを対象としたモデル検査ツールである。 ● デッドロック、アサーション違反、処理されない例外などを検出する。検査は JPF 独自の JVM により解釈、実行される。 ● NASA で開発されたオープンソースソフトウェアであり、火星探査機の制御システムの検証などにも使われた実績を持つ。	http://babelfish.arc.nasa.gov/trac/jpf/wiki/intro/start
ESC/JAVA2	1990年代	School of Computer Science and Informatics at University College Dublin	● ESC/Java2(Extended Static Checker for Java version 2)は JML(Java Modeling Language)のアノテーションを含む Java プログラムの静的コード解析を行い、ランタイムエラーを検出する。 ● JMLは契約に基づく設計(DbC: Design by Contract[163])を念頭に置いた言語であり、事前条件、事後条件などを記述する。 ● ESC/Java2 は、JML 付きのプログラムを論理式で表現し、プログラムのアノテーションで記述された仕様に対する妥当性を検証する。証明器は、Simplify と呼ばれる。 ● 基になった DEC/SRC ESC/Java は DEC/Compaq/HP で開発された。	http://secure.ucd.ie/products/opensource/ESCJava2/
Coq	1980年代	INRIA-Rocquencourt	● Coq は型理論に基づく、定理証明系である。Coq は、数学的証明、形式仕様、プログラム、プログラムの仕様に対する正しさの検証用途に設計されている。 ● Coq は Gallina という名の形式仕様記述言語を提供する。Gallina の用語は、プログラムと、そのプログラムの性質、およびそれらの性質の証明を表現できる。	http://coq.inria.fr/what-is-coq

[163]プログラムコードの中にプログラムが満たすべき仕様についての記述を埋め込むことで設計の安全性を高める方法。

Bandera	2000年代	Kansas State University, University of Nebraska (Lincoln).	● Bandera は Java のソースコードに対するモデル検査するためのツールセットである。 ● プログラムのスライシングと抽象解釈を行うことで、モデル検査を行う。ソフトウェアモデル検査であり、モデルを作る難しさと性質を記述する難しさを緩和することを目標としている。性質の記述は LTL で行う。	http://bandera.projects.cis.ksu.edu/
CafeOBJ	2000年代	北陸先端大学院大学	● CafeOBJ 言語は OBJ 言語を拡張した代数仕様言語である。,振舞仕様、書き換え仕様、パラメータ化仕様などが記述できる。 ● CafeOBJ 言語システムは,等式を書き換え規則として実行することで等式推論を健全にシミュレートすることができ,対話型検証システムとして利用できる。 ● CafeOBJ はメタ言語（Meta Language）としての機能も備えており、形式言語設計にも使える。	http://www.ldl.jaist.ac.jp/cafeobj/
LOTOS	1980年代	University of Twent など	● LOTS は(Language Of Temporal Ordering Specification)分散システム、OSI (Open Systems Interconnection)サービス・プロトコルの設計に利用する形式仕様記述言語である。 ● CCS と CSP に基づくプロセス代数と、抽象データ型言語（ACT ONE)に基づくデータ代数とからなる。並行、非決定的、同期的、非同期的なコミュニケーションの記述に適している。ISO 標準にもなっている。	http://www.cs.stir.ac.uk/~kjt/research/well/
Agda		Chalmers 工科大学、産業技術総合研究所 システム検証研究センター	● Agda は依存型関数プログラミング言語である。値に依存して変化するデータ型を持つことで、ML や Haskell よりも複雑な型を扱うことができる。 ● また、構成的型理論に基づく対話型定理証明支援系でもある。依存型に基づく定理証明型であり、Coq にも類似する。	http://wiki.portal.chalmers.se/agda/pmwiki.php
Z	1970年代	Programming Research Group (PRG) at the Oxford University Computing Laboratory (OUCL)	● Z 記法は、計算機システムを表現するための形式仕様記述言語である。抽象的な高水準で複雑な振る舞いを正確かつ簡潔に書くことを目的としている。 ● Z の意味は Zermelo-Fraenkel 集合論と一階述語論理に基づいており、Z による数式表現は、代数的、論理的に扱うことができる。 ● Z の特徴的な考え方には「スキーマ」がある。スキーマは、いくつかの公理(axiom)により仕様記述された名前付きオブジェクトの集合であり、Z はスキーマをまとめて大きな仕様を後段で作成できるような仕組みを提供する。 ● Z は ISO 標準にもなっている。	Z User Group http://www.zuser.org/

13.2. LTL と CTL

モデル検査における性質の検証には、時相論理が一般に利用される。SPIN(7.3.1.1)では LTL、NuSMV(7.3.1.2)では LTL と CTL、UPPAAL(7.3.1.3)では CTL のサブセットを利用している。

以下では、LTL と CTL により表現される性質について整理し、モデル検査による性質記述を把握するための情報を提示する。

図 13-1 は状態遷移モデルを LTL と CTL がどのようにとらえるかを概念的に示している。前者は実行パスによる直線的な解釈をし、後者は木構造による解釈をする。 13.2.1.1 では LTL による表現について、13.2.1.2 では CTL による表現について、13.2.1.3 では LTL と CTL の表現パターンライブラリについて記載する。

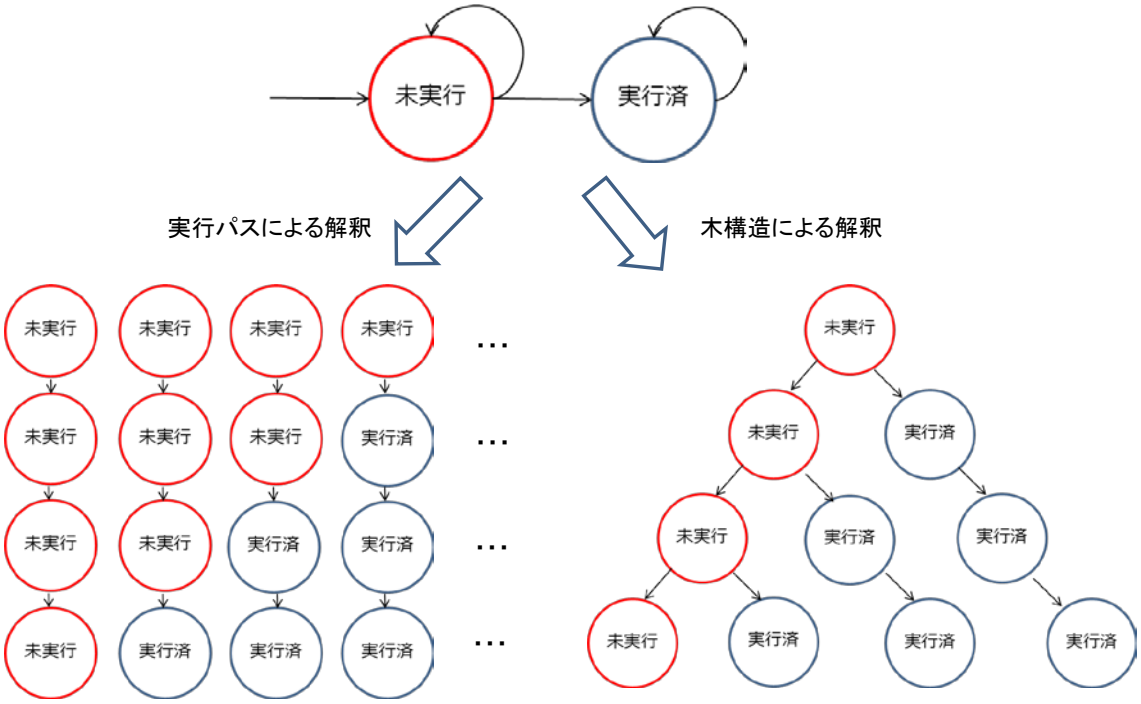

図 13-1：LTLとCTLの簡約な表現[164]

13.2.1.1. LTLによる表現

LTL では、無限長の実行パスによる表現により、性質が成り立つかどうかを記述する。LTL では、図 13-2 に示す性質を記述できる。

① 「いつか」は、現在の状態か、それ以降の状態で P が成り立つことを意味する。
② 「つねに」は、現在の状態と、それ以降のすべての状態で P が成り立つことを意味する。
③ 「つぎに」は、現在の状態の次の状態に P が成り立つことを意味する。
④ 「～まで」は、現在の状態か、それ以降の状態で Q が成り立つまで、P が成り立つことを意味する。

[164]出典:"Model Checking A Hands-On Introduction", A. Cimatti, M. Pistore, and M. Roveri, 2003 に基づき MRI が作成

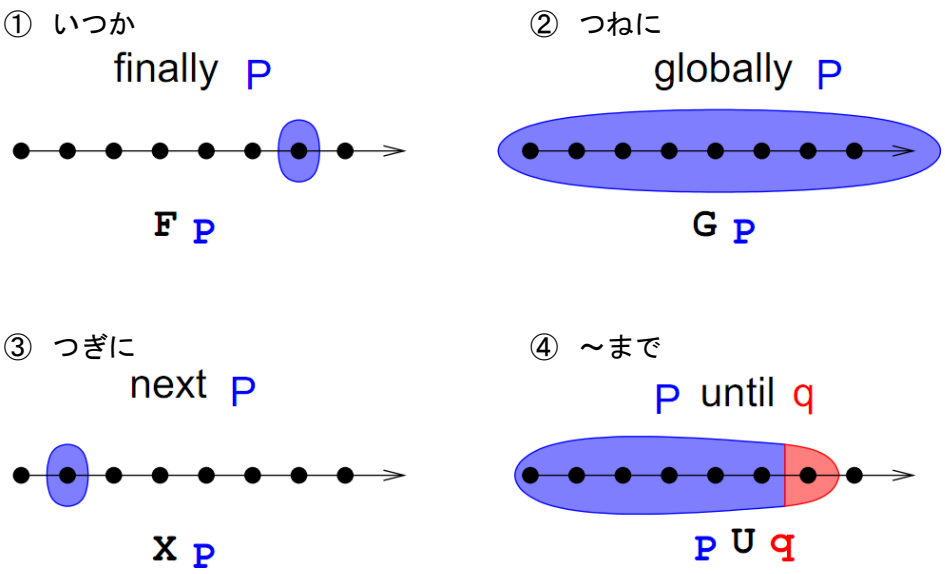

図 13-2：LTLによる表現[165]

13.2.1.2. CTLによる表現

CTL では、木構造の表現により、実行の分岐の可能性について性質が成り立つかどうかを記述する。CTL では、「どのパスでも(A)」「あるパスでは(E)」を LTL の演算子に付加して記述する。図 13-3 に示す性質を記述できる。①～④では LTL の表現に「どのパスでも」という条件が加わり、⑤～⑧では「あるパスでは」という条件が加わる。

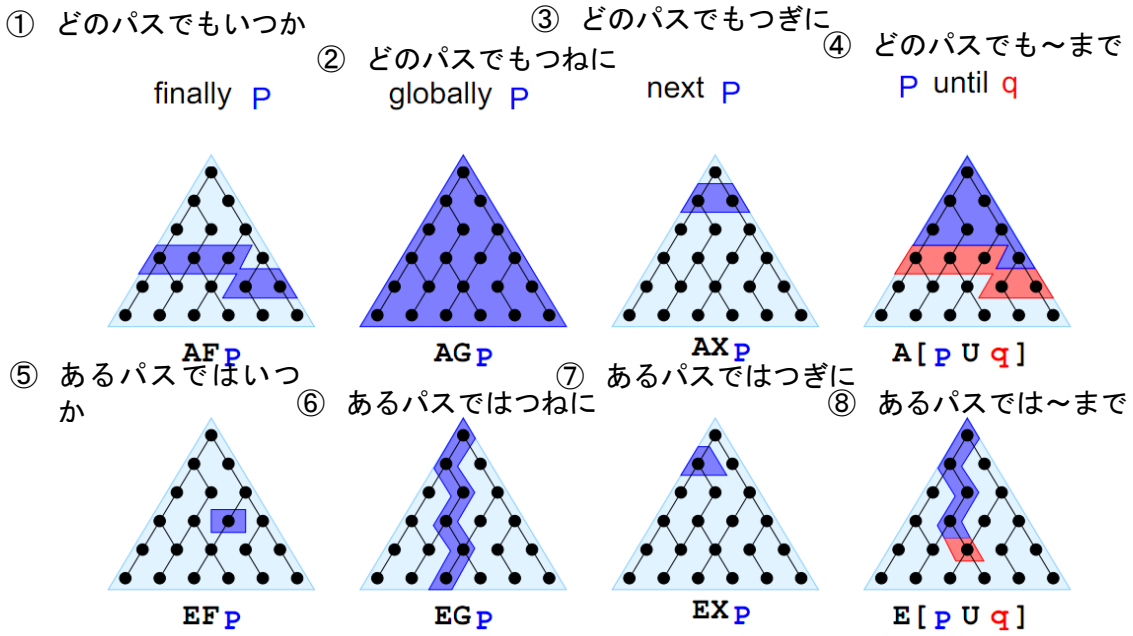

図 13-3：CTLによる表現[166]

[165]出典:"Model Checking A Hands-On Introduction", A. Cimatti, M. Pistore, and M. Roveri, 2003 に基づき MRI が作成

[166]出典:"Model Checking A Hands-On Introduction", A. Cimatti, M. Pistore, and M. Roveri, 2003 に基づき MRI が作成

13.2.1.3. 検証性質の記述パターンライブラリ

検証性質は時相論理式で書かれるので、専門知識が必要となる。しかし、並行・リアクティブシステムの仕様において有限状態の検証で利用される検証性質にはいくつかのパターンがあると考えられ、そのパターンを整理してリポジトリ化する動きがある[167]（以下、パターンライブラリと記す）。

パターンライブラリには LTL や CTL の性質記述パターンも含まれており、代表的な検証性質が表 13-1 に示す階層で分類されている。また、パターンライブラリとは別に、8.2.4 小節にも典型的な記述例を示している。

表 13-1：性質記述パターンライブラリの概要[168]

Occurrence パターン	
absence	ある状態やイベントが、指定された対象範囲内で決して生じないことを表す。
universality	ある状態やイベントが、指定された対象範囲内で常に生じることを表す。
existence	ある状態やイベントが、指定された対象範囲内で生じる(ことがある)ことを表す。
bounded existence	ある状態やイベントが、指定された対象範囲内で k 回生じる(ことがある)ことを表す。バリエーションとして、少なくとも k 回や高々 k 回などがある。
Order パターン	
precedence	状態またはイベント P, S に対して、指定された対象範囲内で、P の前に必ず S が先行することを表す。
response	状態またはイベント P, S に対して、指定された対象範囲内で、P の前に必ず S が応答することを表す。
precedence chain	状態またはイベントの列 P1,…,Pn,、および S1,…,Sm に対して、指定された対象範囲内で、P1,…,Pn の前に必ず S1,…,Sm が先行することを表す。
response chain	状態またはイベントの列 P1,…,Pn,、および S1,…,Sm に対して、指定された対象範囲内で、P1,…,Pn の前に必ず S1,…,Sm が応答することを表す。

[167] 出典：Spec Patterns, SAnToS laboratory, http://patterns.projects.cis.ksu.edu/

[168] 出典："Spec Patterns", SAnToS laboratory, http://patterns.projects.cis.ksu.edu/に基づき MRI が作成