モデル検査によるソフトウェア上流設計の品質向上技術

Software Quality Improvement at Early Design Stages Using Model Checking Techniques

池田　信之　　高田　沙都子　　藤原　聡子
■ IKEDA Nobuyuki　■ TAKADA Satoko　■ FUJIWARA Satoko

近年，ソフトウェアは社会インフラシステムや組込みシステムとして人々の生活により深くかかわるようになってきており，信頼性を確保するために必要な作業コストが増大している。モデル検査は設計仕様の機械的かつ網羅的な探索により，ソフトウェア開発の上流工程で不具合の検出を可能とし，修正のための後戻りコストを大きく低減できる。このため，モデル検査は品質向上とコスト低減を両立させる技術として期待が高まっているが，実際の開発に適用する際には，検査の作業コストを削減することや，多様な検査ニーズへ対応することが必要であり，技術的な課題となっている。

これらの課題への対策として東芝は，検査リードタイムを圧縮するための解析自動化ツール，及び組込みシステムでニーズの高い実時間仕様検査を可能とするモデル化手法を開発した。

As the role of software in industrial systems and embedded systems has continued to grow in recent years, the cost of achieving safety of software has also been increasing. Model checking enables automatic exhaustive exploration of design specifications, thereby realizing fault detection at the early stages of development and reduction of revision costs. Model checking techniques have consequently become an area of increasing interest for achieving both quality improvement and cost reduction. At the same time, reduction of the cost of model checking itself and accommodation of various needs for verification are significant issues in the successful use of these techniques in real-world applications.

Toshiba has developed two techniques to address these issues: (1) automated analysis of counter-examples, and (2) real-time extension of the model checking method.

1　まえがき

ソフトウェアは様々な場所に取り入れられ，人々の生活に大きくかかわってきており，信頼性や安全性を高めることが重要視されている。しかし，ソフトウェアが複雑化してきたことでこれまで以上に誤りが混入しやすくなっているうえ，仕様を網羅した検証が人手では困難になってきている。

このような課題への対策として東芝は，仕様に含まれる問題をツールにより網羅的に検出するモデル検査に着目し，実用化のための技術開発に取り組んでいる。ここではこの一環として開発した，設計に含まれる問題点を自動検出する解析自動化ツール，及び検査の適用範囲を実時間モデルに拡張する手法について述べる。

2　モデル検査の概要

2.1　モデル検査適用の意義

システムの設計段階で仕様の誤りが混入すると，試験工程で動作確認をするまで発見できず，仕様作成までさかのぼって対応する必要があるため，多大な修正コストがかかる（**図 1**（a））。

このような問題を解決する手段として，形式的手法が期待されている。特に，組込みシステムなど装置制御を主な目的とするソフトウェアでは，上流工程で不具合を早期発見する方法として，形式的手法の一つであるモデル検査が注目されている。組込みシステムの開発では，製品開発へのモデル検査の適用事例が徐々に報告され始めている。

モデル検査では仕様に対して検査を実施できるため，シス

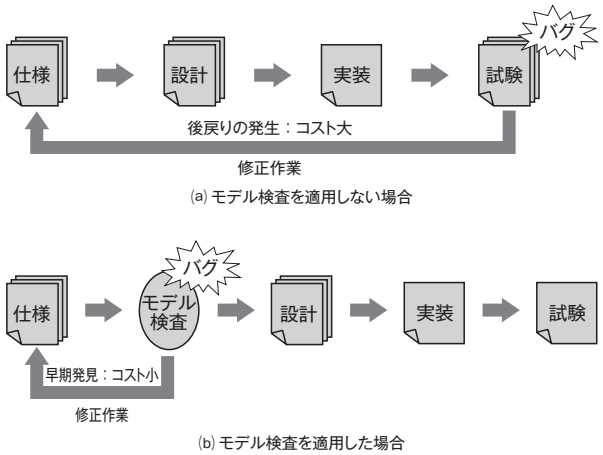

図 1. 従来工程とモデル検査を適用した工程の修正コスト比較 ― 従来工程では，試験工程でバグが発見され後戻りのコストが大きかったが，モデル検査を適用すると，上流設計でバグを修正できるため修正コストが小さい。
Comparison of modification costs of two development processes

テムの不具合を早期に発見することができる。詳細設計や実装を行う前に不具合の修正ができ，後戻りも最小限に抑えることができるため，コスト削減の効果が期待できる（図1(b)）。近年，開発期間は短期化する傾向にあり，このような技術は今後より有用となることが予想される。

2.2 モデル検査の位置づけ

モデル検査は，前述した形式的手法と呼ばれる技術の一つである（**図2**）。形式的手法とは，論理学や離散数学をシステム開発に応用し品質向上を図る技術の総称である。形式的手法には，形式的仕様記述及び形式的検証という二つの大きな技術カテゴリーが存在する。形式的仕様記述は，システムの満たすべき仕様を数式により厳密に記述する技術であり，形式的検証はシステムの性質を数学的に証明する技術である。形式的検証には，数式で表現した仕様モデルから推論を積み重ねて証明する定理証明手法と，状態モデルなどのふるまいモデルを用いて性質を検査するモデル検査がある。

定理証明手法は厳密解が得られるが，適用可能な対象の範囲が狭く，証明に時間が掛かる。一方，モデル検査は検査結果の完全性は保証されないものの実用上十分な検査ができ，適用可能な範囲が比較的広く，検査時間が掛からないという利点がある。

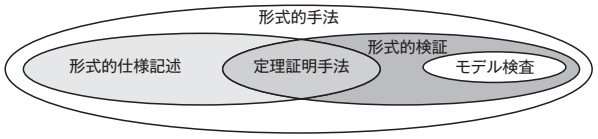

図2. モデル検査の位置づけ — モデル検査は，形式的手法のうち形式的検証に含まれる技術である。
Positioning of model checking

2.3 モデル検査の作業フロー

モデル検査を実施する作業フローを**図3**に示す。モデル検査の入力は，システムのふるまいやシステムが動作する環境を記述した検査用モデルと，システムが満たすべき性質を定義した検査条件の二つである。なお，検査ツールにより，状態モデルからプログラム形式の検査モデルを作成して，代わりに入力とする場合もある。モデル検査を実行した結果，状態モデルに問題があり検査条件を満たさない場合，モデル検査ツールから“反例”として検査条件を満たさないシステムの動作シーケンス（処理の実行順序）の一例が出力される。反例は不具合に至るまでの処理ステップを順次並べたテキストなどで表現される。

反例解析では，出力された動作シーケンスを吟味し，設計上の問題点を特定する。特定された問題を状態モデル上で修正し，反例が出なくなるまで検査を繰り返すことで最終的に正しい状態モデルが得られる。

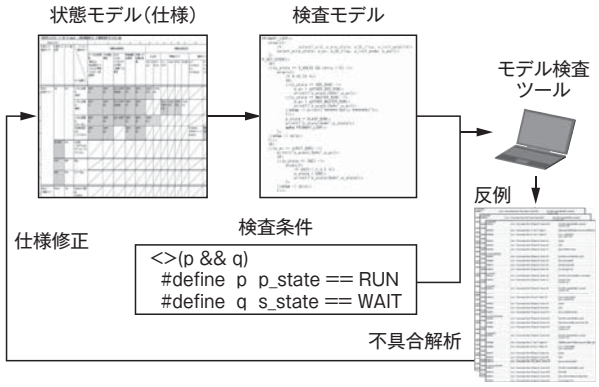

図3. モデル検査の作業フロー — 検査条件と検査モデルを作成した後，モデル検査を実施する。
Workflow of model checking

3 モデル検査を適用する際の課題と取組み

モデル検査を実際の製品開発に適用する際，二つの課題に着目し，対策に取り組んでいる。

第1の課題は，検査リードタイムの圧縮である。モデル検査作業は本来の開発工程を遅らせることなく，タイムリーに実施できることが必要とされる。このため，検査を効率的に実施し，検査工数を圧縮することが重要である。

第2の課題は，製品開発で必要となる多様な検査を実現することである。特に，組込みシステム開発では実時間仕様検査のニーズが高いが，SPIN（Simple Promela Interpreter）(注1)のようなモデル検査ツールでは対応していない[1]。

これらの課題に対し，それぞれ，反例解析を自動化するツールの開発，及び実時間仕様検査を実施する技術を開発し，実際の開発でのモデル検査で利用している。

4 反例解析の自動化による検査リードタイムの圧縮

4.1 反例解析を自動化する意義

モデル検査ツールから出力される反例は非常に量が多く，反例解析の作業が検査リードタイムを長くする要因となっている。実際に代表的なモデル検査ツールを製品仕様に適用した際，反例として出力される動作シーケンスは，テキストで数万行から数百万行の規模となることがある。このような場合，手作業で反例から設計上の問題点を解析する作業には，半日から数日の時間が必要となる。

一方で，反例を解析して得られる設計上の問題には，経験的にいくつかのパターンが存在することがわかってきた。例えば，並行システムでは相互のやり取りのタイミングがずれること

（注1） G. J. Holzmannによる代表的なモデル検査ツール。並行処理システムの動作仕様検査に適している。

によって問題が発生しているものが多い。

このようなことから，反例から問題のパターンを検出するツールにより作業を自動化し，モデル検査作業の効率化を図ることとした。

4.2　反例解析の自動化方法

ここでは反例解析の自動化方法を説明するための代表的な例として，並行処理モデルにおける相互状態参照で発生しうる設計上の問題について示す。

図4(a)に示す並行動作する二つのプロセスP1，P2があり，片方のプロセスが相手プロセスの状態Cを検出すると状態Bに遷移する仕様を想定する。また，状態Bへの遷移は，相手のプロセスが状態Cである場合だけ許されると仮定する。設計者が期待する動作は，プロセスP2の状態Cを検出したプロセスP1が状態Bに遷移したとき，相手プロセスP2は状態Cのままといった状況である。しかし，プロセスP2が状態Cからすぐ状態Dに遷移するような場合，タイミングによってはプロセスP2が状態Cである短い間に，プロセスP1がプロセスP2の状態Cを検出し，状態Bへと遷移する場合がある。プロセスP2はプロセスP1の挙動にかかわらず状態Dへ遷移するため，プロセスP1が状態B，プロセスP2が状態Dという仕様上望ましくない状況が発生する。このような想定していなかった状況の発生が原因となり，反例が出力されるということが多く見られる。

前述の例では相手の状況を検出後，自分の状態の遷移が終わる前に相手の状況が変化しているといった現象が発生しており，この状況は自他の状態と発生しているイベントの並びによりパターン化することができる。このようなパターンをデータとして入力し，反例の中から問題の原因となる箇所を自動検出して出力する反例解析ツールを開発した（**図5**）。

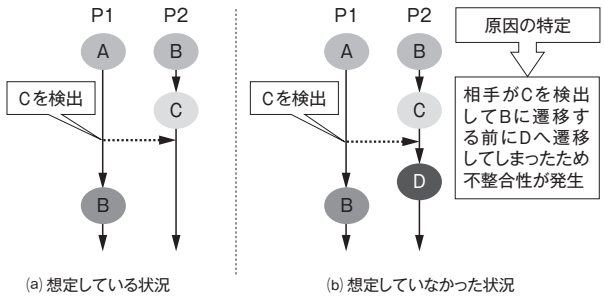

図4．並行処理モデルにおける典型的な反例 — 一時的に発生する想定外の状況が，反例出力の原因となることが多い。

Typical counter-example of parallel processing model

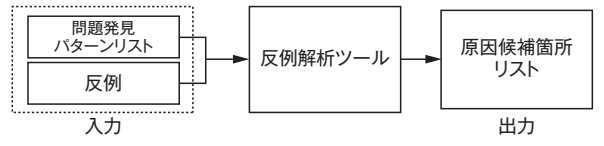

図5．反例解析ツール — 反例解析ツールは，あらかじめ定義した問題発見パターンを用いて，反例の中から問題の原因となる箇所を特定する。

Counter-example analysis tool

4.3　反例解析自動化の効果

反例解析ツールを利用しない場合，出力された反例の最後尾から1ステップずつ問題箇所がないかどうかの確認を行う手順を取るが，ツールを利用した場合，解析を行う対象を絞り込めるため反例解析作業の効率を向上させることができる。ある社会インフラ系の制御システムでの適用事例で計測した結果，従来は反例1件当たり平均で約3時間必要であった解析作業が約30分に短縮できた。また，出力された反例の約80 %についてこの方式を適用した結果，設計上の問題点を特定できた。

5　実時間仕様の検査手法

5.1　実時間仕様を検査する手法の意義

組込みシステムで一般に利用される並行処理では，動作時間を数値で指定する実時間仕様が多く用いられ，これに対する検査ニーズが存在する。しかし，SPINのようなモデル検査ツールでは，実時間仕様を直接表現する機能がない。実時間検査を直接実施可能なツールも存在するが，扱えるモデルのサイズが一般的なモデル検査ツールより小さいという問題がある。

一つの対策として実時間仕様から時間間隔に関する仕様を無視し，順序にかかわる仕様だけを抽出して状態モデルを作成することもできる。しかし，このようなモデルを用いると，時間制約の充足性に対する検査ができないほか，順序にかかわる検査においても，無効な反例を多く検出し，検査作業の効率が著しく低下する。ここで，無効な反例とは設計上発生しえないとみなせる反例を指す。例えば**図6**に示す仕様で，それぞれの処理時間が実際の計測により十分保証されている場合でも，時間間隔を無視した状態モデルでは無効な反例が検出

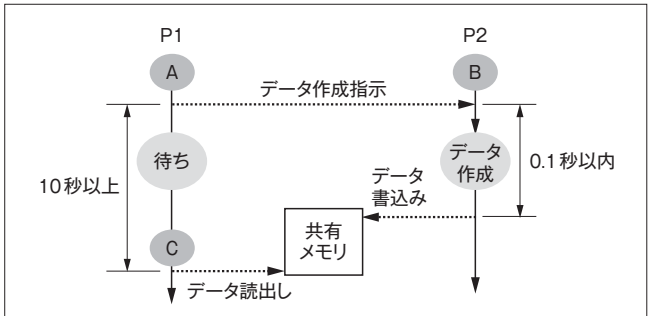

図6．無効な反例を検出する実時間仕様の例 — 時間間隔を無視したモデルでは仕様上，起こりえない反例が検出される。

Example of real-time specification with detection of useless counter-examples by conventional model checking

される可能性がある。例えば，データ書込みとデータ読出しがこの順に処理されることを検査した場合，処理が逆転する反例が検出されるが，実時間仕様で規定された動作では起こりえない。

5.2　実時間仕様のモデル化手法

5.1節で述べた問題に対する対策として，一般的なモデル検査ツールで周期タスクの実時間仕様検査を可能とするモデル化手法を開発した（**図7**）。主な手順を以下に示す。

⑴　個々のタスクの状態を“待ち”，“実行可能”，“実行”の三つの状態に単純化する（①）。

⑵　タスクごとに，優先度，処理時間，待ち時間を割り当てる（②）。

⑶　経過時間に相当する値を実行状態にあるタスクの処理時間，待ち状態にあるタスクの待ち時間からそれぞれ減算する（③）。

⑷　処理時間が0になったタスクを待ち状態に，待ち時間が0になったタスクを実行可能状態に移行させる。なお，待ち状態，実行可能状態へ移行する際，それぞれ待ち時間，処理時間を初期化する（④）。

⑸　実行可能状態であるタスクのうち，もっとも優先度が高いタスクを実行状態に移行させる（⑤）。

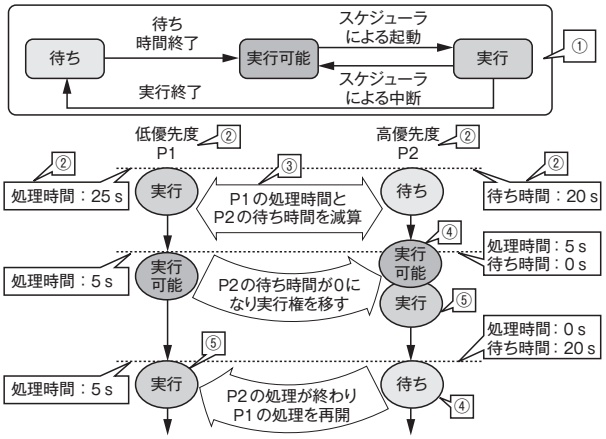

図7．周期タスクの実時間仕様を検査するためのモデル化手法 ― 処理時間と待ち時間の変化をモデル中に明示的に記述することで，一般的なモデル検査ツールでも実時間仕様検査ができる。

Modeling method for real-time model checking of periodic tasks

5.3　実時間仕様のモデル化手法の適用例

5.2節で述べた実時間仕様の検査手法を社会インフラ系の制御システムなどの開発で利用し，効果を得ている。以下の適用例に示すように，様々な観点での検査に適用でき，それぞれの検査で検出した問題は設計仕様の初期検討段階で対策できることから，設計品質向上の効果が得られた。

⑴　データ生成から参照までの時間差の計測　　あるタスクで生成したデータをほかのタスクが参照する際に，もっとも遅れて参照するタスクではどれだけ時間差が生じるかを予測した。

⑵　CPU使用率の計測　　CPUの使用率は，ある単位時間にいずれかのタスクがCPUを占有している時間の割合である。CPUを占有している時間は，いずれかのタスクが“実行”状態である時間を積算することで検出できる。これを利用し，CPUの使用率が高い状況が一定時間以上継続しないことを検査した。

⑶　優先度逆転の発生と処理遅延時間の計測　　優先度の異なるタスクが混在し，共有資源を排他的に利用する場合，低優先度のタスクが占有している資源の解放を高優先度のタスクが待つことで処理が遅れる問題が知られている。このような優先度逆転問題の発生と，発生した場合の処理時間の遅れを予測した。

6　あとがき

モデル検査をソフトウェア開発現場へ適用するための取組みとして，当社が開発した，反例解析を自動化し検査リードタイムを圧縮するためのツール，及び実時間仕様検査を可能とするモデル化手法について述べた。

今後は，これらの技術を更に洗練するとともに，関連する技術の整備と実績の拡大を図る。

文　献

⑴　G. J. Holzmann. THE SPIN MODEL CHECKER. Addison-Wesley, 2004, p.596.

⑵　青木利晃，ほか．“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．東京，2005-10，情報処理学会ソフトウェア工学研究会．2005，p.56－63.

⑶　池田信之，ほか．実用化に向けたモデル検査適用手法の開発．東芝レビュー．**62**，9，2007，p.46－49.

池田　信之　IKEDA Nobuyuki

ソフトウェア技術センター　ソフトウェア設計技術開発担当参事。ソフトウェア上流設計の技術開発に従事。情報処理学会会員。

Software Design Technology Group

高田　沙都子　TAKADA Satoko

ソフトウェア技術センター　ソフトウェア設計技術開発担当。ソフトウェア上流設計の技術開発，及びソフトウェア開発プロジェクトの支援業務に従事。

Software Design Technology Group

藤原　聡子　FUJIWARA Satoko

ソフトウェア技術センター　ソフトウェア設計技術開発担当。ソフトウェア上流設計の技術開発，及びソフトウェア開発プロジェクトの支援業務に従事。

Software Design Technology Group