ソフトウェアテストシンポジウム 2006
JaSST'06: Japan Symposium on Software Testing 2006

モデル検査のデバッグへの適用

篠崎　孝一[†] 太田　弘[†] 早水　公二[††]　星野　光勇[††]

[†]関西電力㈱電力技術研究所　〒661-0974 兵庫県尼崎市若王寺 3-11-20
[††]メルコ・パワー・システムズ㈱技術統括部　〒652-8555 兵庫県神戸市兵庫区和田崎町 1-1-2
E-mail: [†]{shinozaki.koichi@c3, ota.hiroshi@b5}.kepco.co.jp, [††]{khayamiz, hoshino}@pic.melco.co.jp

あらまし　ソフトウェア開発においてプログラムのバグを避けることは困難であり，デバッグ工程においてバグの発見と改修が行われる．バグに起因する不具合の発生頻度が低い場合，デバッグは非常に困難な作業となる．我々は，デバッグの際に明示的ではないがモデル化によるチェックが行われていることに着目し，モデル検査を利用するスキームを開発してデバッグの効率化を行っている．

キーワード　ソフトウェア，デバッグ，モデル検査，不具合，テスト

Model checking application to debug

Koichi SHINOZAKI[†]　Hiroshi OTA[†]　Kouji HAYAMIZU[††]　and Mitsuo HOSHINO[††]

[†]Power Engineering R&D Center, Kansai Electric Power Co., Inc.
3-11-20 Nakoji, Amagasaki-shi, Hyogo, 661-0974 Japan
[††]Engineering Department, Melco Power Systems Corporation.
1-1-2 Wadasaki-cho, Hyogo-ku. Kobe-shi, Hyogo, 652-8555 Japan
E-mail: [†]{shinozaki.koichi@c3, ota.hiroshi@b5}.kepco.co.jp, [††]{khayamiz, hoshino}@pic.melco.co.jp

Abstract　It is difficult to avoid program bugs in Software development, so bug search and fixing is done in the debug process. When program trouble caused by bug hardly reappear in the test process, debug become very hard. Taking notice that model checking is performed in the debug process thought it is not explicit, we have developed the debug scheme using the formal model checking in order to increase the efficiency of debug process.

Keyword　Software, Debug, Model Checking, Program trouble, Test

1. はじめに

ソフトウェア開発においては，要求定義，設計，コーディングの各工程で,バグが混入する可能性がある．

バグを減らすために様々な開発手法が用いられるが，現実にはバグを完全に無くすことができず，デバッグにより取り除く必要がある．

テスト工程で発見された不具合の原因が予測可能で再現性が高い場合のデバッグは比較的容易であるが，原因の予測が難しく再現性が低い場合のデバッグは，非常に困難な作業となる．実際，テスト者が連続監視をしている時には症状が発生せず，たまたま監視していないタイミングで発生することから，勘違いや錯覚と混同されて，存在が疑問視されるような頻度の低い不具合が発生した例もある．希頻度ではあっても，安全性やシステム全体への影響が懸念される場合には原因究明が必須であるが，このような場合には通常，原因の予測が難しく，動作試験の繰り返しにより発見できる可能性も低い．そこで細部に亘るコードチェックや試行錯誤が必要となり，デバッグは莫大な時間を要する非効率的な作業となる．そのためデバッグの改善が望まれている．

2. デバッグ

デバッグの流れは，一般に次のようになる[(3)]．

1.不具合を再現する．
2.原因を発見する．
3.原因を修正する．
4.修正を確認する．

最初の「不具合を再現する」ことにより，その症状から原因を予測して調査範囲・調査内容を絞り込むことが可能になる．不具合の再現性が低い場合は原因予測が難しく，調査範囲・調査内容が拡がって原因発見への道程が遠くなる．

ソフトウェアは，偶然に動作しているものではなく，必要な条件が整えば不具合は再現するが，その条件を

見落とした場合や,条件が整う確率が非常に低い場合,不具合の再現が困難になる.

不具合の症状は様々であり，デバッグ作業者は，仕様書からコーディングまでの知識と，これまでのデバッグ経験で得られたノウハウを総動員して原因を想定し，何度も対象を精査する．必要に応じてテストデータを入力したりデバッグツールを使用し，予測した原因（仮説）に基づいて関係部分をチェックしてバグを見つけ出す．熟練した作業者でも全ての予測原因を同時にチェックすることはできず，一つずつ仮説を立てて，その仮説毎に関係部分だけを抽出して動作を確認していく．すなわち「仮説に応じて（無意識に）モデル化してバグを探している」と考えることができる．

3. モデル検査

モデル検査[3]とは，対象システムを有限状態遷移系にモデル化し，そのモデルが時相論理式で表現した性質を満たすか否かをモデルの取り得る全状態空間の探索により網羅的に検査する手法である．モデル検査とテストの比較を表１に示す．モデル検査は，開発言語で記述されたプログラムコードを直接的に検査するものではない．しかし，モデル化された範囲においてシステム動作の全ての組み合わせを検査することから,適切なモデル化の下では，不具合を再現するための条件の組み合わせを漏れなく確認することができる．

このモデル検査を計算機上で自動的に実行するソフトウェアツール（モデル検査器）が開発・公開[1]されており，誰でも研究・評価用に利用することができる．対象システムをモデル検査言語により記述して入力し，検査項目を時相論理式により記述・入力することで，モデル検査器の中では状態遷移系が論理関数として表現され，その論理演算により自動的に検査が行われる.モデル検査器は,「モデルが検査項目を満足するか否か」の結果を True もしくは False として出力し，検査結果が False になった場合には，モデルが検査項目を満足しない状態遷移列（反例）を出力する機能がある．この反例出力は，再現した不具合の原因調査に利用できる．

表 1. モデル検査とテストの比較

モデル検査	テスト
モデル（状態遷移系）を検査する	コードを検査する
モデル作成が必要	モデル作成不要
テストデータ不要（検査項目は必要）	テストデータが必要
全状態を網羅的にチェック	特定の状態だけをチェック
状態爆発すると答えが返らない	答えは返るが，何度も検査が必要

4. デバッグへの適用検討

デバッグでは，作業者が原因を予測した上で，その仮説に基づいて，ソースコードを関係箇所のみを精査したり，特定の処理に着目したテストデータを入力して処理を実行している．すなわちソフトウェアのある部分だけに着目して，他の大部分は無視している．これは，ある種のモデル化と考えられるが，明示的なモデルは存在せず，モデル作成の手間も必要としない．しかし，発生頻度の低い不具合を見出すには，何回も繰り返して多くの組み合わせをチェックする必要があり，その労力が大きくなる．さらに，予測した原因が正しくなかった場合には，新たな原因予測を行い，その都度，ソースコードを見直したり，テストデータを作成して処理実行を繰り返すため，作業の効率化は難しい．

一方，モデル検査は，明示的な状態遷移モデルを作成する必要があり，そのための作業時間を必要とするが，モデルに発生する全ての組み合わせ条件をモデル検査器が自動検査するため，必要な要素をモデル化できれば見逃しがなく確実にチェックできる（図１）．さらに，１つのモデルに対して複数の検査項目を自由に設定して検査できるため，原因究明が難しい不具合で検査項目が多くなるほど効率が良くなる．論理演算により自動検査を行うので，チェックしている過程の全てを直接的に確認することはできないが，検査項目を満足しない場合には反例が出力されるため，これを読み取ることで，不具合の発生原因を究明できる．

これらのことから，モデル化が容易であればモデル検査をデバッグに使用するができ，デバッグ作業の効率化に有効であると考えられる．

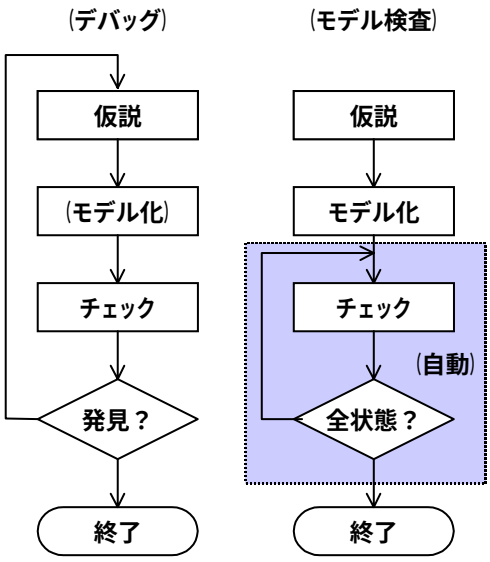

図 1. デバッグとモデル検査の流れ

もちろん，最初にモデルを作成する際に，バグ発見に必要な要素を見落とせば不具合が再現しないため，モデルを修正する必要が生じる．

一般に，上流工程にモデル検査を適用する場合には，「モデル化の範囲（要素）と検査項目の選定」が適当かどうか問題になる[4]．しかし，デバッグ作業に利用する場合には，発見すべき不具合が明らかな場合，もしくはある程度予想できる場合であり，モデル化の範囲と検査項目を容易に決定することができる．

5. モデル検査によるデバッグ手順

実際にモデル検査を適用してデバッグを行う手順を以下に示す．

・検査対象の選定

システム概要，不具合の発生状況，原因予測の難しさ等からモデル検査によるデバッグの有効性を予想して対象を選定する．これまでの経験では，不具合に関係する要素の組み合わせパターン数が多いもの，動作タイミングに関係する不具合の発生しているものは，モデル検査の効果が大きい．

・検査メンバーの選定

現状ではモデル検査を行う技術を有するのがモデル検査の専任者に限られている．さらに，対象ソフトウェアの開発者が協力することが不可欠である．残念ながら，この段階で理解が得られず，モデル検査を適用できなかった事例もある．

・モデル化の準備

モデル検査専任者が開発者から詳細な聞き取り調査を行い，システムを理解した上で原因を予測し，モデル化する範囲（要素）を決定する．従来のデバッグと異なり，この段階で不具合を再現させるための各要素やデータの組み合わせを考える必要はなく，個々の要素を決定するだけで良い．

・モデルの作成

モデル化の範囲（要素）に着目して，対象ソフトウェアのソースコード全体から実際にモデル化する部分を抽出する．この段階でソースコードの量を大幅に絞り込み，モデルを簡単化して状態数を削減することができる．抽出したソースコードに基づきモデル検査言語を使って状態遷移モデルをコーディングする．さらに，不具合症状から検査項目を決定して，時相論理式に記述する．

・モデル検査の実施

状態遷移モデルと時相論理式をモデル検査器に入力して実行すると，自動検査が行われて結果が出力される．検査結果が False になった場合でも，モデルのコーディングやモデル化の誤りに起因する場合があるため，反例を見てこれらの誤りを修正しながら，真の不具合状況を再現させて原因究明に至る．

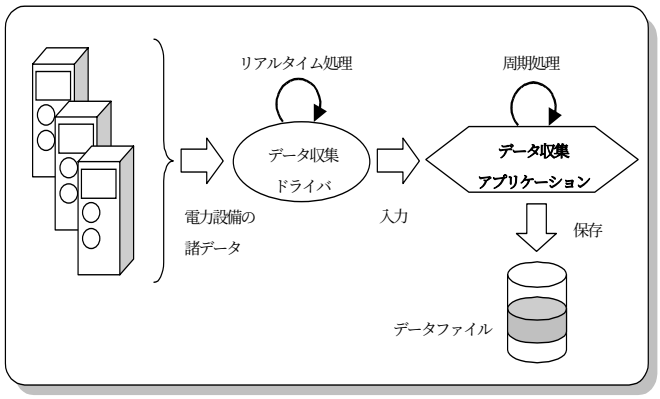

図 2. 適用したシステムの概要

表 2　データ収集アプリケーションの概要

使用言語	C 言語
行数	1730 行（コメント行を含む）
プロセス数	2 プロセス（データ解析プロセスとデータ保存プロセス） データ解析プロセスからデータ保存プロセスを起動するが，起動後は非同期で処理を行う．

6. モデル検査の事例

実際のソフトウェアのデバッグにモデル検査を適用した最新の事例を紹介する．

・検査対象

対象システムは図 2 に示す電力設備保全システムである．従来テストの結果，システムの中枢機能であるデータ収集アプリケーションが稀にデータを取り損ねる不具合が発生していたが，再現性が低いため原因究明の手掛かりが得られなかった．このシステムは現場設置機器への組込みシステムであり，出荷前の品質向上が重要であるため早急に問題点を解消する必要があった．

・検査メンバー

モデル検査専任者と対象ソフトウェアの開発者が協力した．以前にも同種ソフトウェアにモデル検査を適用した実績があり,円滑に開発者の協力が得られた．また，モデル検査専任者も電力関係システムの経験があり，効率良く検査できる条件が揃っていた．

・モデル化の準備

データ収集ドライバは，電力設備の諸データをリアルタイム処理によって収集し計算機のメモリに保存する．データ収集アプリケーションは，それらのデータを入力しデータの変化点抽出等の解析を行い，結果をデータファイルとして保存する．同アプリケーションはリアルタイム処理ではないが，全てのアプリケーションの中で最も実行優先順位が高い．このデータ収集

アプリケーション内で，データが変化する微妙なタイミングに影響されて不具合が発生すると予想された．

そこで，モデル化の範囲をシステムの中枢機能であるデータ収集アプリケーション（表 2）とし，モデル化する要素をデータ収集・解析・保存に関わる部分とした．各手順の所要時間を表 3 に示す．モデル検査専任者がシステムの要点を短時間で理解できたため，4 時間と短い時間でモデル化準備を完了できた．

・モデルの作成

モデル化の範囲（要素）に従ってソースコードを絞り込み，そのソースコードの処理に従って忠実にモデル検査言語によるコーディングを行い，状態遷移モデルを作成した．コーディングと検査項目の作成では特定の予測原因を考慮する必要はない．検査項目は，“不具合症状は絶対に発生しない”という意味を時相論理式で記述した．この検査項目は False となり，反例として不具合に至るまでの実行列が出力される．

・モデル検査の実施

モデルと検査項目を入力したモデル検査器を実行して反例を確認し，モデルのコーディングやモデル化の誤りを修正しながら真の不具合原因を調査した．

モデル検査の所要時間は，3 時間，モデル化準備からでも全 13 時間という短時間で原因を究明することができた．その後，バグ改修に合わせてモデルを改良し，改修内容が正しいことを確認するのに 7 時間を要した．さらに，改良後のモデルを利用してモデル検査を継続した結果，合計 3 件の不具合を改修･確認した．

表 3　モデル検査に要した作業時間

項　　目		時間(H)	
当初バグのモデル検査	モデル化準備	4.0	13
	モデル作成	6.0	
	モデル検査	3.0	
改修内容確認	モデル改造	5.5	7
	再検査	1.5	
別の不具合 1	追加検査項目作成	2.0	11
	モデル検査	1.5	
	モデル改造	4.5	
	再検査	3.0	
別の不具合 2	追加検査項目作成	3.0	14
	モデル検査	2.0	
	モデル改造	3.0	
	再検査	6.0	
		合　計	45

・不具合の事例

モデル検査で発見できた不具合の 1 つを説明する．

本アプリケーションのデータ解析プロセスは周期的に起動され，C 言語の反復処理（for 文）を行っている．その中で，データの変化点を抽出してその数をカウントアップする処理と，変化点の数が既定の数以上となったことをトリガとしてデータ保存要求を出力する処理を行っている．データ保存要求はデータ保存プロセスに対して出力される．図 3 にソースコードの要点を示す．

```
1: for( i=0;i＜4000;i++ ){
       データ変化点検出処理;
       if (データ変化点有){
2:         if(変化点の数＜規定の数){
3:             変化点の数+= 1;
4:         }else{
5:             データの保存要求を出力;
           }
       }
6: }
```

図 3　データ収集アプリケーションの事例

モデル検査によって発見した不具合は，「ある周期の中で，変化点の数が既定の数に達しているにもかかわらず，当該周期中にデータ保存要求を出力せず，その後に 1 周期分遅れて要求を出力してしまう」であった．図 3 のソースコードにより不具合発生に至るまでの時系列を表 4 に示す．

表 4　不具合事例の動作時系列

内　　容	行番号
反復処理の最終回になった時点で変化点の数が既定の数－1 個であった．	1
条件分岐（if－else 文）の第一条件（if 文）に合致する．	2
変化点の数をインクリメントする．この時点で変化点の数が既定の数となる．	3
条件分岐（if－else 文）の第二条件（else 文）は実行されない	4
データ保存要求は出力されない	5
反復処理の最終回であるためデータ解析プロセスは処理を終了する	6
データ解析プロセスが次に起動した時にデータ保存要求が出力される	次周期の 5

この不具合は，変化点の数が既定の数に達するタイミングと反復処理の最終回が一致したときだけ発生する．これは通常の動作試験では発見が極めて困難な不具合である．また，単独で取り出せば明瞭な不具合も，数多くの処理に埋もれた状態でコードチェックを行えば，検出が難しくなる．特に，開発者自身がチェックを行う場合には，思い込みにより見逃してしまう可能性が高くなる．

7. 評　価

前章の事例では，モデル検査の総作業時間が 45 時間，発見・改修された不具合は 3 件である．特に，最初の希頻度不具合のモデル検査を,わずか 13 時間で行い，その原因を究明できている．

このようなタイミングに関係する不具合の場合，通常の動作試験で原因を究明するためには，専用のシミュレーション用ドライバを作成するか，組み込み先の機器に実際にロードした上でタイミングを作り出して試験しなければならず多くの作業時間を要する．従ってモデル検査と同じ 13 時間で完了できた可能性は低い．別の不具合 2 件でも短時間で原因究明から改修後の再検査までを完了しており，モデル検査は十分な効果を上げている．一般に，タイミングに関係する不具合はチェックすべき状態が多く，再現の確率が低いためデバッグが難航するが，モデル検査ではその網羅性を活かして効果的な原因究明が期待できる．

また，この事例でも明らかなように，適切なモデル化により大量のソースコードを相手に試行錯誤を行うことなく，モデル検査器の反例を利用して原因究明が行える．さらに，モデル検査の欠点であるモデル化作業についても，デバッグ対象のシステムソフトウェアを熟知している開発者がモデル検査に協力し，モデル検査の専任者が対象システム分野の知識を有することで，短時間に効果的なモデル化作業が行えることが確認できた．今後，モデル検査を利用するための支援技術の開発が進み，開発者自身がモデル化を行えるようになれば，さらに作業時間短縮が可能と考えられる．

これらのことから，検査対象を適切に選定すれば，モデル検査をデバッグに適用することは，十分実用化できるスキームであると評価できる．

8. まとめ

本稿では，デバッグとモデル検査の類似点と相違点を考察し，発生頻度が低くデバッグの難航する不具合に対して，モデル検査を適用するスキームを示した．さらに，そのスキームを実施するための手順と適用事例を紹介した．検査対象を適切に選定すれば，モデル検査をデバッグに適用することは，十分実用化できるスキームである．

従来，モデル検査は，時間をかけて大規模なモデルを作成し，数多くの検査項目を検証するような使い方[(4)(5)]で捉えられてきたが，デバッグのように限定的な用途に適用することで，モデル化の手間を軽減して効果的に活用することができる．

我々は引き続き，様々なソフトウェアのデバッグにモデル検査を活用して成果を上げつつあり，これらの事例からノウハウを蓄積することで一層の品質向上と効率化を目指した取り組みを行っていく．

なお，我々の活動に先立ち，モデル検査の技術習得にあたって独立行政法人産業技術総合研究所システム検証研究センターから御協力を頂きました．厚く御礼申し上げます．

文　献

[1] B.Berard, M.Bidoit, A.Finkel, F.Laroussinie, A.Petit, L.Petrucci, Ph.Schnoebelen, and P.Mckenzie: Systems and Software Verification: Model-Checking Techniques and Tools, Springer, 2001.

[2] E.M.Clarke, O.Grumberg, and D.A.Peled: Model Checking, The MIT Press, 2000.

[3] S.McConnel: Code Complete. 2nd Ed (Code Complete 第 2 版-完全なプログラミングを目指して，日経 BP ソフトプレス, 2005）

[4] 水口大知，渡邊宏：組み込みソフトウェア開発におけるモデル検査の適用事例，産業技術総合研究所算譜科学グループ研究速報，AIST-PS-2005-001, Jan.2005.

[5] 篠崎孝一，水口大知，石井健志：組み込みソフトウェア開発のイン-デザインモデル検査，ソフトウェアテストシンポジウム,B2c,東京,2004.