情報処理学会論文誌 Vol.56 No.2 569–579 (Feb. 2015)

プレース/トランジションネットとソフトウェア実行履歴を用いた精密化運用プロファイルベースドテスト法

高木 智彦[1,a)] 荒尾 拓矢[2]

受付日 2014年5月19日, 採録日 2014年11月10日

概要：Operational profile-based testing（OPBT）では，運用環境におけるソフトウェアの使われ方やその期待される振舞いを表す確率的モデル（運用プロファイル）に基づいてテストケースを生成，実行する．そしてテスト履歴は，故障の顕在化を含むソフトウェアの実際の動作状況を表す確率的モデル（テストモデル）に要約し，主にソフトウェア信頼性の観点からテスト終了基準を評価する．従来の OPBT では，複数の構成要素が相互に作用しながら並行動作するソフトウェア（並行ソフトウェア）に対して運用プロファイルやテストモデルを精密に構築することが困難であった．そこで本稿では，並行ソフトウェアのために改良された OPBT の手法を提案する．本手法は，プレース/トランジションネットとソフトウェア実行履歴から精密化運用プロファイルを生成し，これに基づいてテストケースの生成，実行，テスト終了基準の評価（精密化テストモデルの生成）を行うというものである．本手法の手順，精密化運用プロファイルと精密化テストモデルの生成アルゴリズム，適用例を示し，有効性や課題について議論する．

キーワード：ソフトウェアテスト，運用プロファイル，プレース/トランジションネット

Detailed Operational Profile-based Testing Technique Using a Place/transition Net and Software Execution Histories

Tomohiko Takagi[1,a)] Takuya Arao[2]

Received: May 19, 2014, Accepted: November 10, 2014

Abstract: In operational profile-based testing (OPBT), test cases are generated from an operational profile that represents the actual use and the expected behavior of software statistically; and then test histories are summarized in a test model that represents actual behavior of the software including the occurrence of failures in order to evaluate test stopping criteria from the viewpoint of software reliability. Traditional OPBT includes the difficulty in constructing an operational profile and test model for concurrent software in detail. Therefore, we propose an OPBT technique improved for the concurrent software. In the proposed technique, a detailed operational profile is generated from a place/transition net and software execution histories, and then the generation/execution of test cases and the evaluation of test stopping criteria (generation of a detailed test model) are performed based on the detailed operational profile. This paper shows the procedure of the proposed technique, algorithms for generating a detailed operational profile and detailed test model, a case study, and discusses its effectiveness and limitations.

Keywords: software testing, operational profile, place/transition net

1. はじめに

ソフトウェアの大規模化，複雑化，短納期化にともなって，ソフトウェア信頼性 [1] をいかに確保するかが開発現場における重要な課題の 1 つとなっている．ソフトウェア信頼性は，運用時に故障が発生しない可能性のことであり，たとえば mean time to failure（MTTF）によって評

[1] 香川大学工学部
Faculty of Engineering, Kagawa University, Takamatsu, Kagawa 761–0396, Japan
[2] 香川大学大学院工学研究科
Graduate School of Engineering, Kagawa University, Takamatsu, Kagawa 761–0396, Japan
[a)] takagi@eng.kagawa-u.ac.jp

価される．ソフトウェア信頼性を改善するには，故障の原因である欠陥を減らす必要がある．出荷前にソフトウェアの故障を発見し欠陥を除去する工程がテストであり，テストを体系的に行うことを可能にするのがテスト手法である．特にソフトウェア信頼性に着目したテスト手法としては operational profile-based testing（OPBT）[2], [3] が提案されている．

OPBT は model-based testing（MBT）[4] の一種であり，運用環境におけるソフトウェアの使われ方やその期待される振舞いを表す確率的モデルに基づいてランダムにテストケースを生成する．この確率的モデルは運用プロファイル（operational profile）と呼ばれ，単純マルコフ連鎖が一般的に用いられている．単純マルコフ連鎖運用プロファイルでは，ソフトウェアの状態を節点，遷移をイベント（ユーザや他のシステムからの入力など）とイベントの運用環境における生起確率でラベル付けした弧で表現する．これは，ソフトウェアの仕様に基づいて作成したステートマシンに，技術者やユーザの予測などに基づくイベントの生起確率を与えることによって作成される．テストケースは開始状態から終了状態に至る状態遷移列として生成され，運用環境におけるソフトウェアの使われ方の特性（すなわち運用プロファイル上の確率分布）を統計的に反映する．ゆえに，運用プロファイルに基づいて生成されたテストケースを実行することによってソフトウェアの運用状態を擬似的に作り出すことができるので，ソフトウェア信頼性を評価したり，ソフトウェア信頼性への影響が大きい故障を発見したりすることができる．最終的には，テスト履歴（テストケース実行結果の累積の記録）を，故障の顕在化を含むソフトウェアの実際の動作状況を運用プロファイルと同様の単純マルコフ連鎖として表現したテストモデル（test model）に要約し，主にソフトウェア信頼性の観点からテスト終了基準を評価する．OPBT は実際のソフトウェアに適用され，その有効性が認められている [5], [6].

我々は過去の研究 [7] において，ステートマシンとソフトウェア実行履歴を用いて精密化した運用プロファイルを生成する手法を構築した．精密化運用プロファイルは，多重マルコフ連鎖（higher-order Markov chain）[8] を用いて作成した運用プロファイルであり，ステートマシン上で定義したアクション（テストデータ適用方法やアサーションをはじめとしたテスト実施のための具体的な手続き）も自動的にマッピングされる．従来の単純マルコフ連鎖ではイベントの生起確率が現在の状態のみによって決まるのに対して，多重マルコフ連鎖では状態遷移の履歴も加味して生起確率を決定することができる．これによって，従来よりも精密にソフトウェアの使われ方の特性を反映したテストケースを生成し，より効果的にソフトウェア信頼性を評価したり，ソフトウェア信頼性への影響が大きい故障を発見したりすることができるようになることを示した．しかしながらこの手法には，以下に示す 2 つの課題がある．

課題 1 並行ソフトウェア，すなわち複数の構成要素（サブシステムやプロセス，手続きなど）が相互に作用しながら並行動作するソフトウェアに対して適用することが困難である．これは，テスト工程に割当て可能なエフォートが限られている中で，並行ソフトウェアの広大な状態空間をテスト技術者がステートマシンによって直接定義しようとするところに原因がある．

課題 2 精密化運用プロファイルを採用した際には，精密化運用プロファイルと同様に多重マルコフ連鎖を用いて作成したテストモデル（精密化テストモデル）が必要となるが，その作成方法が確立されていない．

そこで本稿では，上述の課題を解決するためにさらに改良した OPBT の手法を提案する．本手法は，place/transition net（PN）とソフトウェア実行履歴から精密化運用プロファイルを生成し，これに基づいてテストケースの生成，実行，テスト終了基準の評価（精密化テストモデルの生成）を行うというものである．PN はペトリネットの一種で，並行動作をともなうシステムの分析や設計に適していることが知られており，一部の MBT には導入されている [9], [10]. この PN とソフトウェア実行履歴を用いて並行ソフトウェアのテストされるべき状態空間，ひいては多重マルコフ連鎖を生成し，これにアクションを自動的にマッピングすることによって精密化運用プロファイルを構築できるので，課題 1 を解決できる．そして PN とテスト履歴に基づいて精密化テストモデルを構築できるので課題 2 を解決できる．

本稿の構成は次のとおりである．まず 2 章で本手法の概要とその実施プロセスを，3 章で精密化運用プロファイルと精密化テストモデルの生成アルゴリズムを提案する．次に 4 章で本手法の適用例を示し，有効性や課題を考察する．そして 5 章で関連研究を示し，最後に 6 章で本研究の結論と今後の方針を述べる．

2. 精密化運用プロファイルベースドテスト法

本手法は，ユーザや他のシステムなどと相互作用を行い，それらからの入力に応じて振舞いが変化する性質を持つ並行ソフトウェアに対して，信頼性を評価したり，信頼性への影響が大きい故障を発見したりするためのテスト手法である．MBT，ブラックボックステスト，ランダムテスト，ポジティブテストなどに分類され，システムテストや検査，回帰テストの工程に導入することができる．従来の我々の OPBT [7] との最も大きな違いは，ステートマシンではなく PN を用いる点やテストモデルを構築する点である．また，その他の OPBT（文献 [2], [3] など）との最も大きな違いは，多重マルコフ連鎖やアクションの導入による運用プロファイルの精密化を行う点である．一般的に，MBT におけるモデルの変更はその実施プロセス全体に影響を与える．そこで本章では，本手法の実施プロセスを以下に提案する．

2.1 PN の作成

テスト技術者が，仕様に基づいてテスト対象ソフトウェアの期待される振舞いを PN として記述する．本手法における PN は以下の 8 種類のモデル要素から構成される．

- テスト対象ソフトウェアの構成要素の状態を表すプレース
- 構成要素の現在状態を表すトークン
- 状態遷移のトリガであるイベントを表すトランジション
- プレースとトランジションを接続するアーク
- トランジション発火時に実行すべき動作を表す Firing アクション
- プレースへのトークン追加時に実行すべき動作を表す Entry アクション
- プレースのトークン保持時に継続的に実行すべき動作を表す Do アクション
- プレースからのトークン削除時に実行すべき動作を表す Exit アクション

PN の簡単な例を**図 1** に示す．PN におけるトークンの配置はマーキングと呼ばれ，テスト対象ソフトウェアの状態を表す．たとえば図 1 では，プレース p_1，p_2，p_3 にそれぞれ 1 個，0 個，0 個のトークンが最初に配置されており，この場合のマーキングは $[1,0,0]$ と表される．最初のトークンの配置は特に初期マーキングと呼ばれ，テスト対象ソフトウェアの開始状態を表す．マーキングはトランジションの発火によって変化し，またマーキングの変化によって発火可能なトランジション集合も変化する．これがテスト対象ソフトウェアの状態遷移に対応しており，発火前のマーキング，発火トランジション，発火後のマーキングの三つ組として表現される．必要に応じて，テスト対象ソフトウェアの終了状態を表すマーキングを定義しておくとよい．PN では，発火可能なトランジションは，複数の部分 PN（テスト対象ソフトウェアの構成要素に対応する PN の部分）において同時に存在することができる．さらに，トランジションの発火可能性が複数の部分 PN のトークンの配置に依存したり，発火が複数の部分 PN に影響を与えたりするといった表現も可能である．なお，プレースは複数のトークンを保持することが可能であり，トークン数が有限とならない PN は非有界な PN と呼ばれる．

本手法では，テスト実施のための具体的な手続きを定義

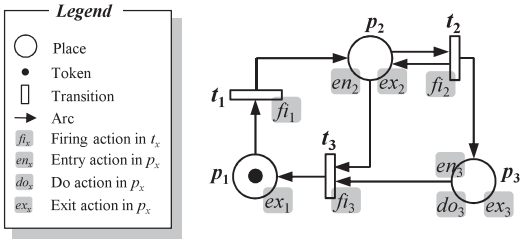


図 1 PN の例
Fig. 1 Example of a PN.

するためのモデル要素として 4 種類のアクションを PN に導入する．これらには，テストデータの作成や適用の方法，アサーション（テスト対象ソフトウェアが満たすべき条件の定義）などを記述することが可能であり，最終的にはテストケースに反映される．

テスト対象ソフトウェアの期待される振舞いが仕様書において PN としてすでに定義されている場合は，それを本手法に流用してもよい．また，PN 以外のモデル（たとえばデータ/制御フロー図）として定義されている場合は，そのモデルを PN に自動変換できることがある [10]．流用や自動変換によって PN を得た場合は，テスト目的に応じて抽象化あるいは詳細化したうえでアクションを追加する．

2.2 ソフトウェア実行履歴の取得

PN に対応付け可能な形式でソフトウェア実行履歴を取得する．これには以下のいずれかの方法を用いる．

(1) フォワードエンジニアリングやラウンドトリップエンジニアリングをサポートした model-driven development（MDD）ツールを用いて，探針をプログラムに挿入する [11]．探針は，トランジションに対応するイベントが生起した際に実行履歴を出力する．
(2) キャプチャ/リプレイツールをはじめとしたソフトウェアテストツールを用いて，トランジションに対応するイベントを記録する [5]．
(3) デバッグツールやセキュリティ対策ツールなどを用いてシステムコールの実行履歴を記録する．そしてシステムコールとトランジションの対応関係を表すマッピングテーブルを定義する [10]．
(4) テスト対象ソフトウェアに対して想定される入力データを運用現場から収集する．そして入力データとトランジションの対応関係を表すマッピングテーブルを定義する．
(5) PN の動作をシミュレーションできるツールを用いて，運用現場での利用を想定して PN を動作させた履歴を取得する．

(1)〜(4) は，テスト対象ソフトウェアのプロトタイプや以前のバージョンなどの類似ソフトウェアに対して，アルファ/ベータテストや実運用を行う中で適用することができる．(5) は，類似ソフトウェアが用意できない場合や運用現場でデータを収集できない場合などに有効である．

図 1 のためのソフトウェア実行履歴の例を**図 2** に示す．ソフトウェア実行履歴は，PN の初期マーキングにおいて発火可能なトランジションから始まる，任意の長さの連続したトランジション列の集合である．終了状態を表すマーキングを PN の作成時に定義した場合は，当該マーキングに到達するトランジションが列の末尾となる．PN とソフトウェア実行履歴との間に不整合がある場合は，PN の内容やソフトウェア実行履歴の取得方法に関する問題を識別

$t_1 \to t_2 \to t_2 \to t_3 \to t_1 \to t_3$.
$t_1 \to t_2 \to t_3 \to t_1 \to t_2 \to t_2 \to t_3 \to t_1 \to t_3$.
$t_1 \to t_2 \to t_2 \to t_3 \to t_1 \to t_2 \to t_3 \to t_1 \to t_3 \to t_1 \to t_2 \to t_3$.
$t_1 \to t_2 \to t_3$.

図 2 ソフトウェア実行履歴の例
Fig. 2 Example of software execution histories.

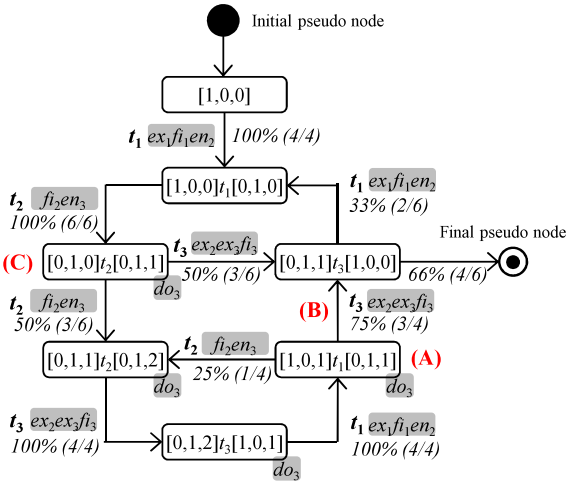


図 3 精密化運用プロファイルの例（$m = 2$）
Fig. 3 Example of a detailed operational profile ($m = 2$).

し修正する．

2.3 精密化運用プロファイルの生成

PN とソフトウェア実行履歴から，3 章で提案するアルゴリズムを用いて多重度 m（$m \geq 2$）の精密化運用プロファイルを生成する．多重度とは，多重マルコフ連鎖として現在状態（現在のマーキング）のいくつ前までの状態遷移（発火前のマーキング，発火トランジション，発火後のマーキングの三つ組）の履歴を考慮するかを決定する値である．精密化運用プロファイルは PN をソフトウェア実行履歴でトレースすることによって導出され，確率付き有向グラフとして表現することができる．その節点は，初期マーキング，初期マーキングから開始される長さ $(m-1)$ 未満の連続する状態遷移列，および長さ $(m-1)$ の連続する状態遷移列であり，トランジションの発火を表す弧が各節点を接続する．節点の状態遷移列の末端が現在状態に，その前の部分が直前の状態遷移の履歴に対応する．さらに，節点や弧には，PN 上で定義されたアクションが状態遷移との対応関係によりマッピングされる．

図 3 は，図 1 の PN と図 2 のソフトウェア実行履歴から多重度 $(m = 2)$ で生成した精密化運用プロファイルである．たとえば，図 3 中の (A) の節点は，$[1,0,1]$ において t_1 が発火することによって現在状態が $[0,1,1]$ となったことを表している．この節点に対応するアクションは do_3 である．また，(B) の弧は，(A) における t_3 の発火によって現在状態が $[1,0,0]$ に遷移することを表している．この弧

$[1,0,0] \to t_1(ex_1 fi_1 en_2) \to [0,1,0] \to t_2(fi_2 en_3) \to [0,1,1]\ (do_3)$
$\to t_3(ex_2 ex_3 fi_3) \to [1,0,0]$.
$[1,0,0] \to t_1(ex_1 fi_1 en_2) \to [0,1,0] \to t_2(fi_2 en_3) \to [0,1,1]\ (do_3)$
$\to t_2(fi_2 en_3) \to [0,1,2]\ (do_3) \to t_3(ex_2 ex_3 fi_3) \to [1,0,1]\ (do_3)$
$\to t_1(ex_1 fi_1 en_2) \to [0,1,1]\ (do_3) \to t_3(ex_2 ex_3 fi_3) \to [1,0,0]$.

図 4 テストケースの例
Fig. 4 Example of test cases.

に対応するアクションは ex_2，ex_3，fi_3 であり，この順番で実行される．(A) では t_3 だけでなく t_2 も発火可能であり，発火確率がそれぞれ 75%と 25%であることが示されている．(C) の節点は，(A) と同じく現在状態が $[0,1,1]$ で t_2 と t_3 が発火可能である．しかしながら直前の状態遷移の履歴は異なっており，これが t_2 と t_3 の発火確率の違いとして反映されている．なお，$(m = 1)$ は従来の OPBT で用いられている単純マルコフ連鎖に相当するが，その場合は状態遷移の履歴を考慮できないので，(A) と (C) が 1 つにまとめられ確率分布が平均化される [7]．ゆえに，このような違いを区別できない．

テスト技術者は，テスト対象ソフトウェアの性質や想定されるユーザ，テストの状況などをふまえて多重度 m を決定する．m の値が大きいほど，運用環境におけるソフトウェアの使われ方をより反映した確率分布を導出できる．すなわち，運用環境におけるソフトウェアの使われ方をより反映したテストケースを生成できることになる．その反面，確率分布に偏りが生じることで，テストケースの多様性が失われる可能性がある．たとえば，ユーザや連携する他のシステムの振舞いの不確実さを加味したり，状態空間を迅速に網羅したりする必要がある場合には m の値を小さくすることもできる．

2.4 テストケースの生成と実行

精密化運用プロファイルを確率分布に従ってトレースする（すなわち，発火可能なトランジションを発火確率に比例する確率でランダムに選択していく）ことによってテストケースを生成する．本手法におけるテストケースは，初期マーキングから始まる連続するトランジションとマーキング，およびそれらに付随するアクションの列である．終了状態を表すマーキングを PN の作成時に定義した場合は，当該マーキングが列の末尾となる．

図 3 から生成したテストケースの例を**図 4** に示す．PN の作成時にアクションを形式言語で記述した場合，テストケースはテストスクリプトあるいはテストドライバとして生成されるので，テスト環境に適用して自動実行する．一方，自然言語で記述した場合は，テスト技術者がテストケースの指示内容に従ってテストを実行する．

テストケースの実行後，テスト履歴を記録する．テスト履歴は，基本的にはテストケースと同様，初期マーキング

$[1,0,0] \rightarrow t_1 \rightarrow [0,1,0] \rightarrow t_2 \rightarrow [0,1,1] \rightarrow t_3 \rightarrow [1,0,0]$.
$[1,0,0] \rightarrow t_1 \rightarrow [0,1,0] \rightarrow t_2 \rightarrow [0,1,1] \rightarrow t_2$ (slight failure in en_3)
$\rightarrow [0,1,2] \rightarrow t_3 \rightarrow [1,0,1] \rightarrow t_1 \rightarrow [0,1,1]$ (fatal failure in do_3).

図 5 テスト履歴の例
Fig. 5 Example of test histories.

から始まる連続するトランジションとマーキングの列の集合であるが，故障が検出された場合はその情報も含まれる．図 4 の実行結果であるテスト履歴の例を図 5 に示す．このテスト履歴は 2 つの列から構成されており，2 つ目の列において軽微な故障（テスト対象ソフトウェアを引き続き実行することが可能な故障）と致命的な故障（テスト対象ソフトウェアを引き続き実行することが困難な故障）が検出されている．故障が検出された場合，故障原因となっている欠陥を除去するためにテストを中断することができる．

2.5 テスト終了基準の評価

テスト終了基準を満足すればテストを終了する．テスト技術者は，以下に示すメトリクスを組み合わせることによって，テスト対象ソフトウェアの性質やテストの状況に応じたテスト終了基準をあらかじめ定義しておく．

(1) MTTF [3]
(2) Kullback discriminant（KD）値 [3], [12]
(3) 節点網羅率，弧網羅率
(4) 平均訪問時間網羅率

(1) は最もよく用いられるソフトウェア信頼性のメトリクスの 1 つであり，OPBT では故障間の平均状態遷移回数として計測される．(2) は運用プロファイルとテストモデルの差異の程度を明らかにするために従来から用いられているもので，故障が十分少なく，かつ，実行したテストケースの量が十分であれば，運用プロファイルとテストモデルの差異は十分小さくなるという原理による．これは本手法の精密化運用プロファイルと精密化テストモデルにもあてはまる．ここで精密化テストモデルとは，3 章で提案するアルゴリズムによって PN とテスト履歴に基づき生成される確率付き有向グラフである．精密化運用プロファイルと同様，その節点は，初期マーキング，初期マーキングから開始される長さ $(m-1)$ 未満の連続する状態遷移列，および長さ $(m-1)$ の連続する状態遷移列であり，トランジションの発火を表す弧が各節点を接続する．ただし，故障が検出された場合は当該故障の顕在化を表す特別な節点と弧が追加される．従来の OPBT では，単純マルコフ連鎖としてテストモデルが作成されていたが，本手法では精密化運用プロファイルに合わせて多重マルコフ連鎖へと拡張されている．図 1 の PN と図 5 のテスト履歴に基づいて生成された精密化テストモデルを図 6 に示す．図中の (A), (B) がそれぞれ軽微な故障と致命的な故障の顕在化を表す節点である．

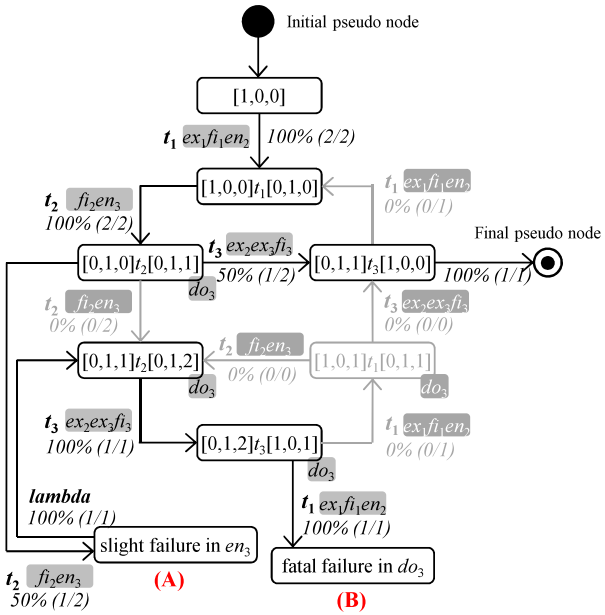


図 6 精密化テストモデルの例（$m = 2$）
Fig. 6 Example of a detailed test model ($m = 2$).

(3) は，精密化運用プロファイルの節点または弧をどれだけ網羅したかを表すメトリクスで，従来用いられていた状態網羅率や遷移網羅率に代わるものである．ソフトウェア信頼性を評価するものではないのでテスト終了基準として単独で用いることはできないが，必要に応じて補助的に採用することで多角的な評価が可能となる．たとえば，(2) は弧をすべて網羅した後でなければ算出できないので，テスト序盤における進捗を把握するのに役立つ．(4) は，精密化運用プロファイルにおけるテスト実行済みの節点または弧について平均訪問時間 [8] の総和をとった値であり，(3) を精密化運用プロファイルの確率分布で重み付けしたものといえる．ここで，ある節点 n（ある弧 e）の平均訪問時間とは，精密化運用プロファイルから確率分布に基づいてランダムに連続する節点（弧）の列を生成したときに，n（e）が出現する割合の期待値である．

3. 精密化運用プロファイルと精密化テストモデルの生成アルゴリズム

精密化運用プロファイルの生成アルゴリズムを以下に提案する．アルゴリズムへの入力は，PN とソフトウェア実行履歴，多重マルコフ連鎖の多重度 m（$m \geq 2$）である．なお，テスト対象ソフトウェアの状態（すなわちマーキング）の集合と遷移の集合をそれぞれ S，T，精密化運用プロファイルの節点の集合と弧の集合をそれぞれ N，E と記述するものとする．最初はすべて空集合である．

（ステップ 1） PN の初期マーキングを開始状態，開始節点としてそれぞれ S と N に追加する．そして開始節点に対応する Do アクション（トークンを保持しているプレースにおいて定義されている Do アクション）

を PN から抽出して開始節点に関連付ける．

（ステップ 2） 初期マーキングから探索を開始するために，現在注目している PN の状態 s_{from} および精密化運用プロファイルの節点 n_{from} を，それぞれ開始状態および開始節点とする．ソフトウェア実行履歴の中から 1 本のトランジション列を取り出す．

（ステップ 3） トランジション列の先頭からトランジションを 1 つ取り出し，s_{from} においてその取り出したトランジション t を発火させる．もし発火後の状態 s_{to} が S の要素でないなら，s_{to} を S に追加する．また，遷移 $<s_{from}, t, s_{to}>$ が T の要素でないなら，$<s_{from}, t, s_{to}>$ を T に追加する．さらに，もし s_{to} の直近の長さ $(m-1)$ の状態遷移列 n_{to} が N の要素でないなら，n_{to} を N に追加するとともに，n_{to} に対応する Do アクションを PN から抽出して n_{to} に関連付ける．また，$<n_{from}, t, n_{to}>$ が E の要素でないなら，$<n_{from}, t, n_{to}>$ を E に追加するとともに，対応する Exit アクション（t の発火によってトークンが削除されるプレースにおいて定義されている Exit アクション），対応する Firing アクション（t において定義されている Firing アクション），対応する Entry アクション（t の発火によってトークンが追加されるプレースにおいて定義されている Entry アクション）を PN から抽出して $<n_{from}, t, n_{to}>$ に関連付ける．

（ステップ 4） $<n_{from}, t, n_{to}>$ の実行回数を 1 増やす．そして，もしトランジション列の終端に達していなければ（すなわち取り出されていないトランジションがあれば），s_{from} を s_{to}，n_{from} を n_{to} としてステップ 3 に戻る．もしトランジション列の終端に達したならば，s_{to} と n_{to} をそれぞれ終了状態および終了節点とし，ソフトウェア実行履歴が空でなければステップ 2 に戻る．

（ステップ 5） 各節点における弧の実行確率を算出する．E の要素である任意の弧 $e = <n'_{from}, t', n'_{to}>$ の実行確率は，n'_{from} を起点とするすべての弧の実行回数の合計（すなわち n'_{from} の訪問回数）に対する，e の実行回数の割合として求めることができる．e の実行確率は，n'_{from} が示す状態遷移列の終端の状態が現在状態で，その直前に当該状態遷移列が実行されている場合に t' が発火する確率を意味する．以上で精密化運用プロファイルが完成する．

精密化テストモデルは，上述のアルゴリズムを以下のように変更することで生成できる．

- ソフトウェア実行履歴ではなくテスト履歴を入力とする．テスト履歴は，ソフトウェア実行履歴と同様，トランジション列の集合と見なすことができる．
- 多重度 m は，2.3 節で生成した精密化運用プロファイルと同じ値とする．
- ステップ 3 の末尾に次の手続きを追加する．「もし t の発火を試みたものの故障のため $<n_{from}, t, n_{to}>$ が期待どおりに実行できなかったならば，当該故障の顕在化を表す特別な節点 n_{fail} を識別し，これが N の要素でなければ（すなわち初見の故障であれば）N に追加する．そして $<n_{from}, t, n_{fail}>$ が E の要素でなければ E に追加する．当該故障が軽微であるためそのままテストが続行され，$<n_{fail}, lambda, n_{to}>$ が E の要素でない場合は，これを E に追加する．$lambda$ は当該の弧がトランジションの発火によらず無条件で実行されることを意味する．最後に，$<n_{from}, t, n_{fail}>$，および該当する場合は $<n_{fail}, lambda, n_{to}>$ の実行回数を 1 増やす」．

4. 考察

本章では，本手法が実際に機能することを確認するための適用例を示す．そしてその結果に基づいて有効性や課題を考察する．

4.1 適用例

我々は on-line file management system（OFMS）[13] に対して本手法を試験適用した．OFMS は，サーバに格納された種々のファイルにアクセスする機能を複数のユーザに対して同時に提供する商用ソフトウェアで，規模はおよそ 100 KLOC である．ユーザとの相互作用によって状態が変化する並行ソフトウェアであり，本手法の適用対象に関する想定と一致する．OFMS はすでにリリースされ，安定して稼動している．

OFMS の PN は文献 [13] においてすでに作成されており，本適用例でもこれを用いた．作成された PN は非有界であり，6 個のプレース，11 個のトランジション，28 個のアークなどから構成される．本適用例ではまず，2.2 節の (5) の方法によって約 200 のトランジションから構成されるソフトウェア実行履歴を取得した．次に，3 章のアルゴリズムを実装したテストツールを用いて，PN とソフトウェア実行履歴から精密化運用プロファイルを生成した．比較のため多重度 m（$2 \leq m \leq 7$）で生成した結果の概要を**表 1** に示す．最後に，精密化テストモデルの生成を含むテスト終了基準の評価を試行するために，先述のテストツールを用いて以下に示す model-based mutation testing（MBMT）を実施した．

(1) 故障を挿入した PN（ミュータント）を生成する．特定のトランジション発火後の特定のアクションの実行によって顕在化するという，現実にありうる故障を想定し，顕在化条件となるトランジションとアクションの組合せはランダムに決定する．

(2) $(m = 4)$ の精密化運用プロファイルからテストケースを生成し，ミュータント上で実行する．もし故障の顕

表 1 生成された精密化運用プロファイルの概要
Table 1 Summary of generated detailed operational profiles.

m	節点の数	弧の数	利用分布網羅率 1)	生成時間 2)（ミリ秒）
2	33	50	0.212	39.9
3	51	69	0.312	48.1
4	70	85	0.469	44.9
5	86	99	0.740	50.6
6	100	108	0.855	54.0
7	109	113	0.909	59.7

1) 精密化運用プロファイルがソフトウェア実行履歴と同様のテストケースを生成できる可能性を表す．
2) 実行環境は，2.5 GHz CPU，2 GBytes RAM を搭載したラップトップコンピュータである．

在化条件を満たせば，当該故障は発見，除去され，テストケースの実行が再開されることとする．

(3) テスト終了基準を，MTTF（トランジション発火回数）が 1,000 以上，KD 値が 0.01 以下，節点網羅率，弧網羅率および平均訪問時間網羅率が 100%であることとする．これを満たすまで (2) の操作を続行する．

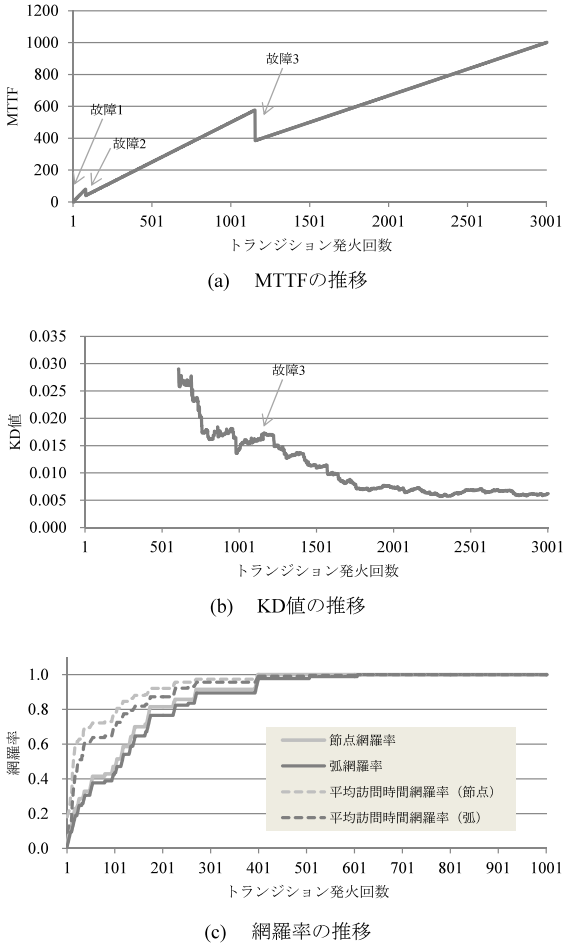
MBMT を実施した結果，テスト終了基準を満たすまでに約 3,000 のトランジションが発火された．テスト終了基準の各メトリクス値の推移を**図 7** に示す．テスト前半で 3 件の故障が発見され，MTTF に関する条件を発火回数約 3,000 で，KD 値に関する条件を発火回数約 1,600 で満たした．また，節点と弧の網羅にはそれぞれ発火回数約 400 と約 600 を要した．

4.2 有効性および課題

表 1 より，多重度が大きくなるにつれて節点や弧の数が増加しており，生成時間も増加の傾向を示していることが分かる．しかしながら，特に後者の増加の程度は小さい．これは，3 章のアルゴリズムが PN 全体ではなくソフトウェア実行履歴に基づく探索を行うためである．特に OFMS のような非有界な PN では到達可能なマーキング（状態空間）が無限に存在する難しさがあるため，OPBT においてテストするべき有限の状態空間を自動的に導出できるという点は本手法の特長といえる．適切に状態空間を導出できるか否かはソフトウェア実行履歴の取得にかかっているが，それには一定の技量やツールが必要となる点が課題である．

本研究では，精密化運用プロファイルの精度，すなわちソフトウェアの使われ方の特性を精密化運用プロファイルが反映する程度を評価するためのメトリクスとして，利用分布網羅率 [14] を導入する．利用分布網羅率は，ソフトウェア実行履歴*1 を構成する各列について精密化運用プロファイル上でのトランジションの発火確率の総積を求め，

*1 利用分布網羅率はテスト十分性評価のメトリクスであるため，本来はテストケースに対して適用される．



(a) MTTFの推移
(b) KD値の推移
(c) 網羅率の推移

図 7 テスト終了基準のメトリクス値の推移
Fig. 7 Progress of metrics values for test stopping criteria.

それらの総和をとる（ただし重複する列は除外する）ことによって算出される．これは，精密化運用プロファイルがソフトウェア実行履歴と同様のテストケースを生成できる可能性を表しており，この可能性が大きいほど精度が高いといえる．表 1 より，多重度が大きくなるにつれて利用分布網羅率が大きくなっており，より高い精度を得られることが分かる．ただし，高い精度を追求しすぎると，精密化運用プロファイルの確率分布に偏りが生じ，テストケースの多様性が失われる可能性もある [7]．本適用例では ($m = 4$) としたが，このように，精度と多様性のバランスをとるために利用分布網羅率が 0.5 付近となる（すなわち，生成されるテストケースの半分がソフトウェア実行履歴と同様になることが期待される）多重度を設定するというのが，多くのテストの現場で適用できる考え方の 1 つである．

MBMT を実施した結果，テスト終了基準の評価を多角的に行うことができることが分かった．テスト序盤では網羅率（図 7 (c)）によって，中盤以降では MTTF や KD 値（図 7 (a)，(b)）によって進捗を把握できる．平均訪問時間網羅率は，節点網羅率や弧網羅率と比較して成長が早い．これは，OPBT では確率の高い状態遷移が優先的にテストされているからである．KD 値は，確率の低い状態遷移

をテストしたり故障を検出したりすると上昇するが，十分にテストを実行し故障の検出がなくなると低い値に収束する．特にこれをグラフ化するとOPBTの進捗を把握することに役立つ．KD値の算出には精密化テストモデルを使用したが，精密化テストモデル自体もテスト履歴を俯瞰するのに有用であり，その可視化ツールを利用できるようになることが望ましいと考えられる．

本手法におけるテストの有効性はソフトウェア実行履歴に影響される．たとえば，ソフトウェア実行履歴の量が少ない場合は，実際はテストが十分でないにもかかわらずテスト終了基準を早期に満足する可能性がある．これは，精密化運用プロファイルの状態空間が狭い範囲に限定されるため，KD値が早期に収束したり，節点や弧を容易に網羅できたりすることが要因である．また，ソフトウェア実行履歴の内容が現実の運用環境における使われ方を反映していない場合も，テスト終了基準の評価が妥当とならない．これは，テストにおいて故障が検出されるタイミングや頻度が現実とは乖離したものとなってしまい，MTTFやKD値などを正確に評価できないからである．ゆえに，2.2節で示した自動化された方法によって，想定される運用環境からソフトウェア実行履歴を効率的に取得できることが重要である．

本適用例では，テストケース実行はMBMTによるシミュレーションとして実施した．実際のプログラムに対して効果的にテストケースを適用できるか否かは，自動テスト環境の性能やテスト技術者の技量に依存する．OPBTをはじめとしたMBTは自動化を指向した手法であるが，テストケースの自動実行は発展途上の技術であり，今後MBT全体の課題として取り組まれる必要がある．

4.3 従来手法との比較

1章において，従来手法[7]に対して2つの課題（課題1と課題2）が存在することを指摘した．まずは，従来手法によって精密化運用プロファイルを作成する方法を示しつつ課題1について考察する．

従来手法ではまず，テスト対象ソフトウェアの期待される振舞いを表すステートマシンを仕様に基づいて作成する必要がある．テスト対象ソフトウェアが並行ソフトウェアの場合，これはPNのマーキングを節点，トランジションの発火によるマーキングの変化を弧とする有向グラフを直接手作業で作成することに相当する．ここで，本適用例のOFMSのPNは非有界であるため，節点や弧を有限とするために特別なシンボルを導入しなければならない[13]．たとえば，同じく非有界である図1のPNでは，p_3のトークン数は際限なく増加しうるので，トークン数がk以上となったことを表すシンボルω_kを導入して$[0,1,\omega_k]$のようにマーキングを表現する．ステートマシンの作成においては，ω_kの導入に相当することは抽象化の一環として一般的に行われる．ω_2を本適用例に導入すると，約170の弧からなる有向グラフが構築される．次に，本手法と同様の方法でソフトウェア実行履歴を取得する．そして最終的には，従来手法[7]のアルゴリズムによって，これらの有向グラフとソフトウェア実行履歴から精密化運用プロファイルを生成できるものの，ω_2の導入が精密化運用プロファイルの精度の劣化という問題を引き起こす．たとえば，先述の図1の例で$[0,1,2]$と$[0,1,3]$の間でトランジションの発火確率に違いがあったとしても，$[0,1,\omega_2]$として1つにまとめられるため，発火確率の違いを識別できない．後でkの適切な値が分かったとしても，有向グラフを作成し直さなければならない．kの値を大きくすれば発火確率の違いを識別できる範囲を広げることができるが，kの値が大きくなるにつれて有向グラフは大きくなるため，作成に要するエフォートが問題となる．たとえば本適用例の弧の数は$(k=3)$で約320，$(k=4)$で約510となる．したがって，本適用例に従来手法を用いることは困難である．従来手法でこのような問題が起きる原因は，ソフトウェア実行履歴を確率分布の導出のみに用いている点にある．これに対して本手法では，確率分布の導出だけでなく，精密化運用プロファイルの節点や弧の導出にも用いている．状態や遷移は，ソフトウェア実行履歴中に出現するもののみを対象とするため，ω_kを導入しなくても必ず有限となる．

本手法では，節点や弧の導出にはソフトウェア実行履歴に加えてPNが不可欠である．PNの作成は，並行ソフトウェアの各構成要素における状態遷移，および構成要素間の相互作用を明らかにする作業と考えることができる．このような作業は，従来手法でステートマシンを作成する際にも必ず行う．そして，並行ソフトウェア全体の状態空間を導出するためには，構成要素間の相互作用を考慮しつつ，すべての構成要素の状態遷移を合成する作業がさらに必要である．従来手法ではステートマシン作成作業の一部としてこれを手作業で行うのに対して，本手法では精密化運用プロファイルの生成過程で自動で行うので，有界・非有界にかかわらず本手法の方が作業に要するエフォートは少なくなると考えられる．ただし，一般的にMBTにおけるモデル作成の品質やエフォートは，テスト技術者個人の能力にも依存するため，能力を維持・向上させるための取り組みは重要である．

以上より，本手法によって課題1を解決できる．

次に課題2についてであるが，精密化テストモデルの作成における問題は，上述の精密化運用プロファイルの作成における問題と基本的に同じである．ゆえに精密化テストモデルの生成アルゴリズムは，3章で示したように，精密化運用プロファイルの生成アルゴリズムとほぼ同じ形で実現されており，実際に生成しKD値を算出できることは本適用例において示した．以上より，本手法によって課題2も解決できる．

5. 関連研究

OPBT は文献 [2], [3] などによって提案され，その後も適用事例や改良された手法が報告されている．たとえば Hartmann ら [5] は，医療用システムのテストに OPBT を導入することによって従来よりも工数を削減することに成功した．Poore ら [15] は，運用プロファイル上の確率分布を数理計画法における制約条件としてまとめることで再利用を促進する方法を示した．また，Popovic ら [6] は，通信プロトコルの運用プロファイルからテストドライバを生成することで自動テストを可能とする手法を確立した．従来の OPBT の方法では，仕様に基づいてステートマシンを作成し，これに技術者やユーザの推測，運用現場の調査などに基づく確率分布を与えることで単純マルコフ連鎖運用プロファイルを作成することが多い．OPBT の有効性をさらに高めるには，ユーザの振舞いの文脈をより詳細にとらえることができる多重マルコフ連鎖を導入する必要があり，多重マルコフ連鎖として精密に運用プロファイルを生成するにはソフトウェア実行履歴の利用が不可欠であった．そこで我々は文献 [7] において，ステートマシンとソフトウェア実行履歴から精密化運用プロファイルを生成する手法を提案したが，並行ソフトウェアに対して精密に運用プロファイルやテストモデルを作成する方法は依然として確立されていなかった．これらをふまえて，ステートマシンではなく PN を使用するという基本的な考えのもと改良された OPBT が本稿の提案手法である [16]．PN の特殊なものがステートマシンである[*2]という意味では，本稿の提案手法は文献 [7] を包含するともいえる．精密化運用プロファイルや精密化テストモデルは，PN や多重マルコフ連鎖などの導入によって大幅に拡張されたものとなっているが，確率付き有向グラフの形式を保っているため，従来の OPBT のための方法論やツールとの親和性が高いと考えられる．

ペトリネットを用いた MBT はいくつか提案されている．たとえば Li ら [9] は，MIPv6 の相互運用テストをペトリネットに基づいて体系的に行う方法を提案した．Dong ら [17] は，Web サービス合成を高水準ペトリネットによってテストする方法を示した．Ho ら [18] は，時間ペトリネットを用いてリアルタイムソフトウェアをテストする手法を提案した．我々も文献 [13] において，PN の長さ N（$N \geq 1$）の連続する状態遷移列を網羅するためのテスト終了基準，およびテストケース生成アルゴリズムを提案した．これらは，いずれもペトリネットのモデル要素やタイミングなどを網羅するためのテスト手法，すなわち coverage-based testing（CBT）であり，ソフトウェア信頼性に着目したテスト手法である OPBT とは異なっている．網羅することがテストの主目的である場合は，OPBT ではなく CBT を採用すべきである．ただし，CBT で用いられている網羅基準を OPBT のテスト終了基準の一部として補助的に導入することは可能である．実際，2.5 節で述べた節点網羅率と弧網羅率は CBT の網羅基準であり，精密化運用プロファイルの多重度を m とすると，文献 [13] における $(N = m - 1)$，$(N = m)$ にそれぞれ対応する．ただし，本手法ではソフトウェア実行履歴を用いて節点や弧を導出するため，ソフトウェア実行履歴中に出現しない状態遷移列は考慮できない．

PN とソフトウェア実行履歴を品質管理に活用する研究としては，平井ら [10] による分散制御システムのデバッグ手法がある．この手法では，稼働中の分散制御システムから収集した実行履歴で，当該システムの仕様に基づいて作成した PN をトレースする．これによって，動作が複雑で検査が困難な分散制御システムの実行順序や各種実装ミスなどに関する故障を効果的に発見することができる．本稿の提案手法においても，精密化運用プロファイルを生成する過程で PN をソフトウェア実行履歴でトレースするため，PN が正しいか，あるいは実行履歴の取得に用いたソフトウェアに問題はないか，といった観点でデバッグを行うことができる．

ペトリネット以外による並行ソフトウェアのテスト手法としては，たとえば片山ら [19] の事象同期グラフに基づくテストケース生成手法がある．事象同期グラフは，プログラムの逐次的に動作する各構成要素についてイベントの生起順序を表す有向グラフを作成し，構成要素間の同期関係に基づいてそれらを接続することで作成される．事象同期グラフ上の節点や弧などを網羅するパスがテストケースとなる．本手法と異なってモデルはソースコードから生成可能で，ホワイトボックステストにおいて特に有効な CBT である．清野ら [20] は，形式仕様言語 CafeOBJ で記述された並行ソフトウェアの仕様（振舞いとデータのモデル）について形式手法による検証を行うとともに，実装された並行ソフトウェアを自動テストするためのテストコードを生成する手法を示した．形式手法を導入している点や，各遷移規則をテストしたうえで負荷テストを行うというアプローチをとっている点が特徴的である．並行ソフトウェアのテスト手法は，たとえば探索的テストやバグパターン，静的解析を用いるものなど，他にも数多く提案されている [21]．

近年では，MBT にミューテーションテストを導入した MBMT [22] が提案されており，ミューテーション解析やネガティブテストを実現する新たな手法として注目されている．前者は，任意のテストケースを評価するために，当該テストケースをミュータント（故障を故意に挿入したモデル）上で実行するというもので，4 章において用いられ

[*2] トークンが 1 つだけ存在し，すべてのトランジションが入力アークと出力アークを 1 つずつしか持たない場合，PN をステートマシンと見なすことができる．

た．一方，後者ではミュータント中の故障に焦点を当てたテストケースを設計する．たとえば，精密化テストモデルに蓄積されている故障や信頼性に関する情報を基に，潜在する可能性が高く信頼性への影響が大きい故障を挿入したミュータントを生成し，その故障を効果的に発見するためのテストケースを選りすぐって生成するといった応用が考えられる．

6. おわりに

本稿では，並行ソフトウェアに対して運用プロファイルやテストモデルを精密に構築することが困難であるという課題を解決した新たな OPBT の手法を提案した．本手法では，PN とソフトウェア実行履歴から精密化運用プロファイルを生成し，これに基づいてテストケースの生成，実行，テスト終了基準の評価（精密化テストモデルの生成）を行う．精密化運用プロファイルの生成においては，非有界な PN についても有界な PN と同等に扱うことが可能であり，OPBT においてテストするべき有限の状態空間を自動的に導出できる．また，多重度を高く設定することで精密化運用プロファイルはソフトウェアの使われ方の特性をより精密に反映する．テスト終了基準としては MTTF，KD 値，節点網羅率，弧網羅率，平均訪問時間網羅率を導入し，多角的にテストの進捗を把握することができる．精密化テストモデルは PN とテスト履歴から生成され，KD 値の算出などに用いられる．一方で，ソフトウェア実行履歴の取得やテストケースの実行には一定の技量やツールが必要となる．

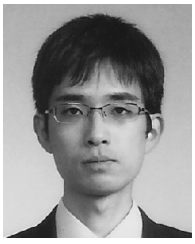
今後の研究では，PN を高水準ペトリネットや時間ペトリネットなどに拡張することで精密化運用プロファイルの表現力を高めるとともに，再利用が容易となるように抽象化手法を確立する．また，テストの目的や状況に応じて最適化されたテストケースを精密化運用プロファイルから生成する手法を開発する予定である．

謝辞 本研究は JSPS 科研費 23700038 の助成を受けた．また，株式会社ジャストシステム多田雅信氏には本研究に関して有益なご意見をいただいた．ここに深く感謝の意を表する．

参考文献

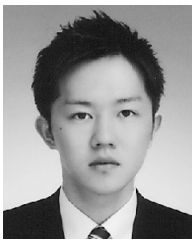
[1] Rook, P.: *Software Reliability Handbook*, Elsevier Science (1990).
[2] Musa, J.D.: Operational Profiles in Software Reliability Engineering, *IEEE Software*, Vol.10, No.2, pp.14–32 (1993).
[3] Whittaker, J.A. and Thomason, M.G.: A Markov Chain Model for Statistical Software Testing, *IEEE Trans. Softw. Eng.*, Vol.20, No.10, pp.812–824 (1994).
[4] Utting, M., Pretschner, A. and Legeard, B.: A taxonomy of model-based testing approaches, *Software Testing, Verification and Reliability*, Vol.22, pp.297–312 (2012).
[5] Hartmann, H., Bokkerink, J. and Ronteltap, V.: How to reduce your test process with 30% – The application of Operational Profiles at Philips Medical Systems, *Supplementary Proc. 17th International Symposium on Software Reliability Engineering*, CD-ROM (2006).
[6] Popovic, M., Basicevic, I., Velikic, I. and Tatic, J.: A Model-Based Statistical Usage Testing of Communication Protocols, *Proc. 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, pp.377–386 (2006).
[7] 高木智彦，古川善吾：多重マルコフ連鎖に基づく精密化利用モデルの構築とその適用例，情報処理学会論文誌，Vol.51, No.9, pp.1794–1804 (2010).
[8] Ching, W.-K. and Ng, M.K.: *Markov Chains: Models, Algorithms and Applications*, Springer (2006).
[9] Li, H., Ye, X., Wu, C., Liu, L. and Wang, L.: Modeling Interactive Property of MIPv6 with Petri Net for Interoperability Testing, *Proc. 2nd International Conference on Information and Computing Science*, pp.313–316 (2009).
[10] 平井健治，杉本 明，阿部 茂：分散制御システムのデバッグ手法：要求仕様を用いたイベントヒストリの検査，情報処理学会論文誌，Vol.33, No.4, pp.491–500 (1992).
[11] Takagi, T. and Furukawa, Z.: Constructing a Usage Model for Statistical Testing with Source Code Generation Methods, *Proc. 11th Asia-Pacific Software Engineering Conference*, pp.448–454 (2004).
[12] Kullback, S.: *Information Theory and Statistics*, Wiley (1958).
[13] Takagi, T. and Furukawa, Z.: Test Case Generation Technique Based on Extended Coverability Trees, *Proc. 13th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp.301–306 (2012).
[14] Takagi, T., Nishimachi, K., Muragishi, M., Mitsuhashi, T. and Furukawa, Z.: Usage Distribution Coverage: What Percentage of Expected Use Has Been Executed in Software Testing?, *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, Studies in Computational Intelligence, Vol.209, pp.57–67, Springer (2009).
[15] Poore, J.H., Walton, G.H. and Whittaker, J.A.: A constraint-based approach to the representation of software usage models, *Information and Software Technology*, Vol.42, No.12, pp.825–833 (2000).
[16] 高木智彦，荒尾拓矢：プレース/トランジションネットとソフトウェア実行履歴を用いた精密化運用プロファイルの生成，信学技報（KBSE2013-74），pp.53–58 (2014).
[17] Dong, W.-L., YU, H. and Zhang, Y.-B.: Testing BPEL-based Web Service Composition Using High-level Petri Nets, *Proc. 10th International Enterprise Distributed Object Computing Conference*, pp.441–444 (2006).
[18] Ho, I. and Lin, J.: Generating Test Cases for Real-Time Software by Time Petri Nets Model, *Proc. 8th Asian Test Symposium*, pp.295–300 (1999).
[19] 片山徹郎，菰田敏行，古川善吾，牛島和夫：並行処理プログラムにおけるテストケースの定義と生成ツールの試作，情報処理学会論文誌，Vol.34, No.11, pp.2223–2232 (1993).
[20] 清野貴博，中村正樹：OTS/CafeOBJ 法に基づく並行システムの実装とテスト生成，信学技報（CST2010-33），pp.7–12 (2010).
[21] Souza, S.R.S., Brito, M.A.S., Silva, R.A., Souza, P.S.L. and Zaluska, E.: Research in concurrent software testing: A systematic review, *Proc. Workshop on Parallel*

and Distributed Systems: Testing, Analysis, and Debugging (2011).
[22] Weiglhofer, M., Aichernig, B. and Wotawa, F.: Fault-based Conformance Testing in Practice, *International Journal of Software and Informatics*, Vol.3, No.2-3, pp.375–411 (2009).



高木 智彦 （正会員）

2002 年香川大学工学部卒業．2004 年同大学大学院工学研究科修士課程修了．2007 年同大学院博士後期課程修了．2008 年香川大学工学部助教，2013 年講師，現在に至る．博士（工学）．ソフトウェア工学，特にソフトウェアテスト法の研究に従事．電子情報通信学会，ソフトウェア科学会各会員．



荒尾 拓矢 （学生会員）

2014 年香川大学工学部卒業．現在，同大学大学院工学研究科博士前期課程に在学．ソフトウェア工学，特にソフトウェアの品質管理技術に興味を持つ．