5A-01

プレース／トランジションネットに基づく ソフトウェアネガティブテストのフレームワークの提案

高木 智彦

香川大学工学部

1. はじめに

ネガティブテストは，テスト対象ソフトウェアの想定される特定の故障を発見するための手法である[1]．特に広大な状態空間をもつ並行ソフトウェア（複数の構成要素が互いに協調しながら並行動作するソフトウェア）では全体を網羅的にテストすることが困難であるため，リスクのある故障に着目したネガティブテストを行うことは有効なテスト戦術のひとつと考えられる．従来手法では，テスト対象ソフトウェアの期待される振舞いをステートマシンとして定義し，これを変異させたモデルであるミュータントに基づいてネガティブテストケースを生成することが多い．本稿では，ステートマシンではなく，並行ソフトウェアの期待される振舞いの形式的表現に適したプレース／トランジションネット（PN：Place/transition Net）に基づいた，新たなネガティブテスト法のフレームワークについて考察する．本フレームワークは，ミュータント生成器とネガティブテストケース生成器を中心に構成されるもので，テスト技術者が作成した PN と，想定される故障に関する定義である故障パターンを入力として与えると，その想定される故障に特化したネガティブテストケースを生成する．

本稿では，2 節で一般的なネガティブテストの概要を示す．そして 3 節で本研究が提案するネガティブテスト法を実現するためのフレームワークについて考察する．

2. ネガティブテストの概要

本研究でいうネガティブテストは，テスト対象ソフトウェアの期待される振舞いを表す形式的モデルに基づいてテストを実施する手法である MBT（Model-Based Testing）と，故意に挿入した欠陥の検出を試みることによってテストケースの質を改善する手法として知られるミューテーションテスト法を組み合わせたものである．その手順は以下の通りである．

[ステップ 1] オリジナルモデルの作成
仕様書などに基づいて，テスト対象ソフトウェアの期待される振舞いを表す形式的モデル（オリジナルモデル）をテスト技術者が作成する．モデル化の範囲や粒度は，テストの目的やテスト対象ソフトウェアの性質などに基づいて決定される．

[ステップ 2] ミュータントの生成
オリジナルモデルに対して，想定される欠陥を意図的に挿入することによってミュータントを生成する．ここで使用される欠陥挿入方法はミューテーション・オペレータと呼ばれ，オリジナルモデルの表記法毎にあらかじめ定義される．

[ステップ 3] ネガティブテストケースの生成
ミュータントに基づいて，挿入した欠陥を顕在化させるテストケースであるネガティブテストケースを生成する．ネガティブテストケースは入力条件と期待結果からなり，挿入した欠陥を網羅する必要がある．生成アルゴリズムは，オリジナルモデルの表記法毎に開発される．

[ステップ 4] テストの実行
ネガティブテストケースを使用してテスト対象ソフトウェアを実行し，想定される欠陥が顕在化しないことを確認する．欠陥が顕在化した場合は，テスト対象ソフトウェアを修正し，再度実行・確認する．すべてのネガティブテストケースについて欠陥が顕在化しないことを確認できれば，テストを終了する．

3. フレームワークの提案

本節では，PN に基づいた新たなネガティブテスト法の概要，およびそれを実現するフレームワーク（本手法を実装したテスト環境の構成の概要）を提案する．

PN はペトリネットの一種であり，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，状態遷移を表すトランジションとアークから構成される．並行ソフトウェアの状態は PN 上のトークンの配置であるマーキングによって表される．ペトリネット

The Framework of Software Negative Testing based on a Place/transition Net
Tomohiko Takagi
Faculty of Engineering, Kagawa University, Takamatsu, Kagawa 761-0396, Japan

は従来から並行ソフトウェアの設計や MBT（ポジティブテスト）に用いられ，効果的であることが報告されている[2]．ステップ 1 では，従来の PN によるモデリング法に基づいてオリジナルモデルを作成する．これには GUI を備えたモデリングツールを使用する．さらに，テスト技術者はテスト対象ソフトウェアのリスク分析を行い，マーキングに基づく表記法によって故障パターンを定義する．本手法の最終成果物は，故障パターンを顕在化させるネガティブテストケースであり，それには故障パターンを含むミュータントが生成されなければならない．

ミュータントを生成するにはミューテーション・オペレータが必要であるが，著者らは PN のためのミューテーション・オペレータとして，プレース，トークン，トランジション，アークのそれぞれについて追加または削除するというものを提案している[3]．オリジナルモデルに対してミューテーション・オペレータを適用した部分が欠陥となるが，PN の場合は欠陥と故障（欠陥によって生じるソフトウェアの不具合の現象）の対応関係が直接的ではない．ゆえに，故障パターンを含むミュータントを生成するためのミューテーション・オペレータを決定論的に選択することは困難である．そこで本手法では，メタヒューリスティクスを応用したミュータント生成アルゴリズムを開発し，ツール（ミュータント生成器）として実装する．たとえば，メタヒューリスティクスの代表的なものの一つである GA（Genetic Algorithm）を採用した場合のミュータント生成アルゴリズムとしては以下が考えられる．

1. 初期集団を生成する．初期集団を構成する各個体（解候補）は，ミューテーション・オペレータと，オリジナルモデルにおける当該ミューテーション・オペレータの適用対象（プレース，トークン，トランジション，アーク）のランダムな組合せの集合である．個体を構成する各組合せを本稿では遺伝子と呼ぶことにする．
2. 交叉と突然変異によって新たな個体を生成し，集団に追加する．交叉では，ランダムに選択した個体間で遺伝子を交換することによって新たな個体を生成する．一方，突然変異では，ランダムに選択した個体の一部の遺伝子を，新たに生成した遺伝子で置き換えることによって新たな個体を生成する．
3. 各個体の適合度を求める．ある個体の適合度は，総故障パターン集合 F の要素数に対する，当該個体により生成されるミュータントに含まれる到達可能な故障パターン集合 I（$I \subseteq F$）の要素数の割合であり，この割合が大きいほど優れた個体と判定される．
4. 集団の世代交代が一定回数に達するか，最良の個体の適合度が一定期間変化しない場合，最良の個体を最終的な解候補とする．それ以外の場合は，優れた個体をエリート保存法やルーレット選択法により選択し，その選択した個体群を次世代集団として 2. に戻る．
5. 4.で得られた最終的な解候補に含まれる故障パターンを F から除く．F が空でなければ 1. に戻る．

上述のようなアルゴリズムによってミュータントが得られれば，各ミュータントに基づいてすべての故障パターンを網羅するネガティブテストケースを生成できる．ここでいうネガティブテストケースは，ミュータントの初期マーキングからはじまり故障パターンに合致するマーキングで終了する，マーキングとトランジションの連続する列である．本研究では，グラフ探索アルゴリズムなどを用いてネガティブテストケースを生成するツール（ネガティブテストケース生成器）を開発する．

4. おわりに

今後の研究では，ミュータント生成器とネガティブテストケース生成器を試作し，有効性の評価とアルゴリズムの改良を行う予定である．

謝辞

本研究は JSPS 科研費 26730038 の助成を受けた．

参考文献

[1] M. Weiglhofer, B. Aichernig and F. Wotawa, "Fault-based Conformance Testing in Practice", International Journal of Software and Informatics, Vol.3, No.2-3, pp.375-411, 2009.

[2] H. Li, X. Ye, C. Wu, L. Liu and L. Wang, "Modeling Interactive Property of MIPv6 with Petri Net for Interoperability Testing", Proc. 2nd International Conference on Information and Computing Science, pp.313-316, 2009.

[3] T. Takagi, R. Takata, Z. Furukawa, F. Belli, M. Beyazıt, "Metrics for Model-Based Mutation Testing Based on Place/Transition Nets", Proc. of Joint Conference of 21st International Workshop on Software Measurement and 6th International Conference on Software Process and Product Measurement (IWSM-MENSURA), pp.7-10, Nov. 2011.