B-010

ペトリネットに基づくミュータントを用いたネガティブテストケースの作成
Construction of Negative Test Cases Using Mutants Based on Petri Nets

高木 智彦†
Tomohiko Takagi

1. はじめに

ペトリネットは，ソフトウェアの状態遷移に基づく振舞いを抽象化して表現する際に有用な形式的モデルのひとつであり，形式的モデルに基づいてテストケースを設計する手法である MBT（model-based testing）において用いられている．テストケースは，テスト対象ソフトウェアに対する入力条件と期待結果から構成され，形式的モデルを網羅するように設計することが一般的である．しかしながら，大規模，複雑な形式的モデルから生成されるテストケースは時として膨大であり，テスト工程に投入可能な限られたエフォートですべてを実行することが困難な場合がある．そこで，テスト対象ソフトウェアの想定される故障を顕在化させるテストケース（ネガティブテストケース）を設計する手法である MBMT（model-based mutation testing）が検討されている[1]．MBMT では形式的モデル全体ではなく，想定される故障に関連する部分のみを対象とすればよい．したがって，想定される故障がテスト技術者によって適切に定義されるならば，テストケースの量を抑えることができると考えられる．

ペトリネットを用いた MBMT 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ペトリネット（オリジナルモデル）に対して，想定される故障を挿入したモデル（ミュータント）を作成する[2]．本稿では，このミュータントを用いて，効果的なネガティブテストケースを作成する方法について述べる．

2. ペトリネットに基づくミュータント

本節では，ネガティブテストケースの作成に用いるミュータントについて述べる．

MBMT では，まずテスト技術者がオリジナルモデルをペトリネットによって記述する．本稿で想定するペトリネットは，PN（place/transition net）と EPN（extended place/transition net）の二種類である．PN は，テスト対象ソフトウェアを構成するコンポーネントの状態を表すプレース，コンポーネントの現在の状態を表すトークン，コンポーネントの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．テスト対象ソフトウェア全体の状態はマーキングによって表現される．たとえばテスト対象ソフトウェアの初期状態は初期マーキングとして表現される．一方，EPN は，トランジションの発火に伴うアクションや発火のための制約条件の記述が可能な形式的言語（仕様記述言語やプログラミング言語）を導入した PN である[3]．テスト対象ソフトウェア全体の状態は，マーキングに加えて，アクションで使用される大域変数の値で表現される．たとえばテスト対象ソフトウェアの初期状態は，初期マーキングと大域変数の初期値によって表現される．

オリジナルモデルが完成すると，次にテスト技術者は，想定される故障の状態（故障状態）を定義する．想定される故障とは，実運用において顕在化する可能性の高い故障や，顕在化した場合の影響が大きい故障のことであり，テスト対象ソフトウェアのテスト履歴や適用対象などに基づくリスク分析によって抽出する．故障状態は，PN の場合はマーキング（のパターン），EPN の場合はマーキングと大域変数の値（のパターン）として定義する．

ミュータントは，オリジナルモデルに対して MO（mutation operator）を適用し，故障状態を引き起こす欠陥を挿入することによって生成される．オリジナルモデルが PN の場合は，PN の構成要素の変更を行う model-based MO が用いられる．一方，オリジナルモデルが EPN の場合は，model-based MO に加えて，形式的言語で記述されたアクションのステートメントに対する変更を行う code-based MO が用いられる．いずれも，故障状態と MO の関係は直接的ではなく，定式化による最適解の導出が容易ではない．したがって，メタヒューリスティクス（たとえば遺伝的アルゴリズム）の応用による準最適解の導出が有効である．

3. ネガティブテストケースの作成

前節で述べたミュータントを用いて，ネガティブテストケースを作成する方法について述べる．

ペトリネットを用いた MBT におけるテストケースは一般的に，初期マーキングから始まる，連続するマーキングとトランジションの列である．トランジションがテスト対象ソフトウェアに対する入力条件，マーキングが期待結果に対応している．ペトリネットとして EPN を用いる場合では，大域変数の期待される値がマーキングに付随する．一方，ペトリネットを用いた MBMT におけるネガティブテストケースも，その終端が故障状態に対応すること以外は MBT のテストケースと同じ構成である．しかしながら，初期状態から故障状態に至る最短経路を探索するという方法で作成する点が，ペトリネット全体を網羅することに主眼を置くことが多い MBT とは異なっている．

以下では，ミュータントが PN に基づく場合と，EPN に基づく場合に分けて，ネガティブテストケースの作成方法を議論する．

3.1 PN に基づく場合

ミュータントが PN に基づく場合は，まずミュータントから到達可能グラフを生成する．到達可能グラフは，到達可能なマーキングを節点，マーキングの変化を引き起こすトランジションを弧とする有向グラフであり，素朴な探索アルゴリズム（たとえば深さ優先探索など）によって導出することができる．次に，故障状態が到達可能グラフに含まれていることを確認する．故障状態が複数含まれていることもあるし，ひとつも含まれていないこともある．後者

† 香川大学工学部電子・情報工学科
Faculty of Engineering, Kagawa University
Takamatsu, Kagawa 761-0396, Japan

の場合は，ミュータントが有効ではない（すなわち，ネガティブテストケースの生成が不可能である）ため，ミュータントを生成し直さなければならない．ミュータントが有効であることが確認できた後，当該ミュータントの到達可能グラフに基づいて，初期状態から故障状態に至る最短経路をネガティブテストケースとして生成する．最短経路は，たとえばダイクストラ法によって求めることができる．故障状態が複数ある場合，基本的にはその中のいずれかに到達すればよい．ただし，故障状態に至る複数の経路をテストすることも有意義であり，*k*-shortest path 探索法（*k* は 2 以上の整数）を利用する選択肢が考えられる．

3.2 EPN に基づく場合

一般的に EPN では，特定の状態への到達可能性を証明したり，特定の状態に最短で到達するための条件を最適解として導出したりすることは困難なことが多い．これは，大域変数が様々なアクションの実行を通して更新されたり，制約条件が大域変数の値に依存したりするためである．したがって，ミュータントが EPN に基づく場合では，メタヒューリスティクスを用いて準最適解としてネガティブテストケースを導出するのが現実的である．メタヒューリスティクスとしては，アントコロニー最適化[4]や遺伝的アルゴリズムなど，様々な手法が存在する．アントコロニー最適化によってネガティブテストケースを生成する場合では，たとえば以下のようなアルゴリズムが考えられる．

[ステップ 1] EPN を PN として解釈（すなわち，アクションや制約条件を無視）した場合の到達可能グラフを生成する．そして，到達可能グラフのすべての弧に対して初期フェロモン濃度 τ_0 を付与する．

[ステップ 2] エージェントを *a* 個生成し，初期マーキングに対応する到達可能グラフの節点に配置する．そして，各エージェントに EPN の大域変数の初期値を記憶させる．

[ステップ 3] 以下のように各エージェントが探索を行う．まず，現在滞在している節点を起点とするトランジション（PN の現在のマーキングにおいて発火可能なトランジション）を抽出する．次に，抽出した各トランジションについて，発火のための制約条件を満たすか否かを，現在のマーキングおよび記憶している大域変数の値に基づいて EPN 上で確認し，制約条件を満たさないものを除外する．そして，それらの中からひとつのトランジションをフェロモン濃度に比例する確率で選択し，発火させる．発火させたトランジションにアクションが付随している場合は当該アクションを実行する．この結果にしたがって，各エージェントは節点を移動し，記憶している大域変数の値を更新する．以上のような方法で，故障状態に到達するか，または移動可能距離 *D*（各エージェントにおけるトランジションの発火可能回数の上限）に達するまで，各エージェントは探索を繰り返す．

[ステップ 4] 故障状態に到達できたエージェントを有効なエージェントと呼ぶことにする．そして，今回得られた有効なエージェントについて，探索した経路が最短のもの（トランジションの発火回数が最小のもの）を抽出し，これを *iteration-best* と呼ぶことにする．さらに，本アルゴリズムを開始以来得られたすべての有効なエージェントの中で経路長が最短のものを *best-so-far* と呼ぶことにする．

[ステップ 5] ステップ 3〜4 の実行回数が *s* に到達した場合は，*best-so-far* の経路を準最適解として確定し，本アルゴリズムを終了する．ただし，*best-so-far* が存在しない場合は，ミュータントが有効ではないので解が存在しないと判断し，本アルゴリズムを終了後にミュータントを生成し直す．

[ステップ 6] 今回得られた有効なエージェント，または *iteration-best* のみを用いて各弧のフェロモン濃度を更新する．すべての弧について，フェロモン濃度は ρ の割合で蒸発して減少する．同時に，これらのエージェントが通過した弧については，当該エージェントの最終的な経路長に反比例する量のフェロモンを追加する．

[ステップ 7] ステップ 2 に戻る．

以上のアルゴリズムを実行する際には，τ_0，*a*，*D*，*s*，ρ などのパラメータの適切な値を試行錯誤法により決定する．

4. おわりに

PN または EPN に基づくミュータントを用いて，想定される故障が実際に潜在していないことを確認するためのテストケース（ネガティブテストケース）を作成する方法について述べた．初期状態から故障状態に至る最短経路を探索するという戦略を採用し，PN に基づくミュータントに対してはダイクストラ法や *k*-shortest path 探索法を，EPN に基づくミュータントに対してはアントコロニー最適化をはじめとしたメタヒューリスティクスを適用する．

EPN のほうが形式的モデルとしての表現力が高いため，適用できる範囲は広いと考えられる．ただし，アルゴリズムのパラメータについては，EPN の規模や構造上の特徴などにより適切な値が異なるため，効果的にその値を探し当てることができる必要がある．また，大規模で複雑な EPN に対しても限られた時間内で探索を行い，有効な解を導出できなければならない．

今後の研究では特に，EPN に基づくミュータントからネガティブテストケースを生成するアルゴリズムを検討する．検討したアルゴリズムをツールとして実装するとともに，例題への適用を通して有効性の評価と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] T. Takagi, T. Arao, "Overview of a Place/Transition Net-Based Mutation Testing Framework to Obtain Test Cases Effective for Concurrent Software", Proc. of 16th Int. Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 3 pages, June 2015.

[2] 高木智彦, "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", 情報処理学会第 77 回全国大会, pp.189-190, 2015.

[3] 高木智彦, 赤木章紀, "拡張プレース/トランジションネットに基づく VDM 仕様の構築手法の提案", 情報処理学会第 79 回全国大会, pp.195-196, 2017.

[4] 筒井茂義, "ACO: アントコロニー最適化", システム/制御/情報, Vol.52, No.10, pp.390-398, 2008.