5A-03

ユースケーステスト技法に基づく経路抽出手法の提案

張　暁晶 †　　丹野　治門 †
†NTT ソフトウェアイノベーションセンタ

1　はじめに

テストはソフトウェア品質確保における重要な一手段である．開発現場では「テスト実行」の自動化に盛んに取り組んでいる一方，テスト項目の洗い出しを行う「テスト設計」は属人的スキルに基づいて手作業で行うことが多く，稼働がかかるうえ，テスト項目の良し悪しが品質や納期にも大きく影響する．筆者らは，Web 系業務システムの結合テストをスコープとし．シナリオテストのテスト設計技法としてユースケーステスト [1] に着目する．ユースケーステストは，ユーザシナリオやシステムの使い方に基づいてテスト項目を抽出する経験的技法である．従来手作業で技法を適用してテスト設計を行うとき，作業手順の不在や判定基準の曖昧さにより，過不足なくテスト項目を洗い出すことが困難という問題がある．

ユースケースに着目したテスト項目自動抽出の既存研究としては，Nebut ら [2] や Heumann[3] の関連文献があるが，具体的な抽出のアルゴリズムに言及したものではない．そこで本研究では，ユースケーステストの技法に基づくテスト項目自動生成を目的とする経路抽出手法を提案する．提案手法は，ユースケーステストに必要とされる経路が確実に含まれるよう，経路の候補を格納する経路モデルを作成し，かつ，経路数爆発を抑えるために経路モデルを段階的に拡充することを特徴とする．

2　提案手法

入力として，ユーザとシステムのインタラクションを表現し基本／代替／例外フローから構成されるユースケース記述を読込み，ユースケーステスト技法で定義された下記観点 A〜D のいずれかに分類されるテスト項目を網羅的に出力する．入出力の具体例は筆者らの過去文献 [4] に示されている．

観点 A 基本フローを順次実行する
観点 B それぞれの代替/例外フローを 1 回のみ実行する
観点 C なるべく全ての代替/例外フローを実行する
観点 D それぞれの代替/例外フローを i 回繰り返して実行する (本研究では $i = 2$ とする)

以降，図 1 に示す手順に沿って，図 2 の簡単な例を用いて，入力から出力を得る方法を説明する．

Step1. まず，1 つのユースケース記述を 1 つの UML アクティビティモデルに変換する．具体的には，表形式のユースケース記述にある，基本・代替・例外フローにある自然言語記述の 1 ステップ（例 : ユーザが画面 A のボタン B をクリック）を 1 つの UML アクションノードに，基本フローと代替フロー等の間の 1 回のジャンプを 1 つの UML 分岐ノードに，ノード間の流れを「基本／代替／例外」のフロー種別を付与された UML 制御フローにそれぞれ変換する．基本フローの先頭と末尾にそれぞれ UML 初期状態ノード，UML 終了ノードを付加する．さらに，得られた UML アクティビティをノードとエッジで構成される有向グラフに単純化する．

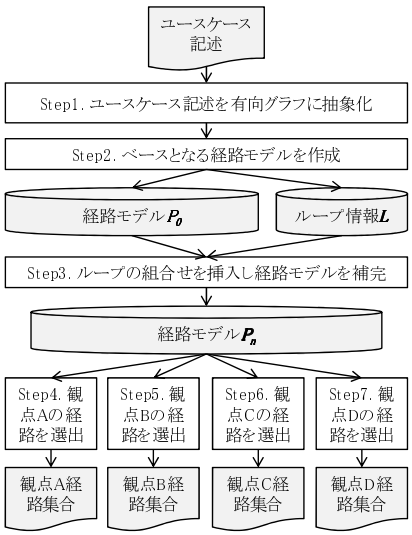

図 1: 経路抽出手法の概要フロー

Step2〜Step3. 有向グラフから木構造の経路モデル P を抽出する．有向グラフを探索中に発見したエッジを P のノードとして登録していく．木 P にある 1 つのリーフは，1 本の経路の最後のエッジを表す．

まず，Step2 では，有向グラフに存在するループを無視して経路モデル P_0 を抽出し，どんなループがあったかをループ情報 L として記憶しておく．深さ優先探索を有向グラフに適用し，これまでに見つけた部分経路と，これから探索しようとしているエッジの行き先を照合し，重複有無によりループか否かを判定し，ループであれば探索を打ち切る．図 2 に示す有向グラフの例で，[a,c,e,b,c] と探索し，次にエッジ e を探索する場合を考える．e を部分経路に加えると [a,c,e,b,c,e] となる．e の行き先であるエッジ群 b と c のうち，部分経路の末端に近いのは c であるため，ループとして [c,e] を記録しておき，ループ部分を消して部分経路を [a,c,e,b] に更新する．同様に続けていくと，ループ [b] も見つかる．ループを含む経路が打ち切られるため最終的に経路モデル P_0 には [a,c,d] という経路のみ含まれる．

A Path Extraction Method Based On Use Case Testing Technique
†Xiaojing ZHANG　†Haruto TANNO
†Software Innovation Center, NTT

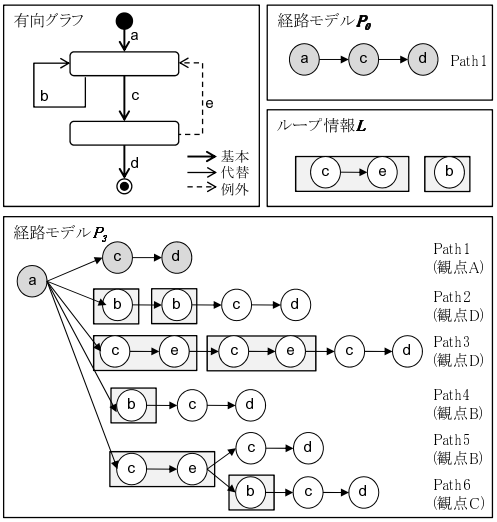

図 2: 経路モデルの生成例

つぎに，Step3 では，予め記憶したループ情報を用いて，ユースケーステスト技法で必要とされる特徴を持った経路を，経路モデル P_0 へ追加しモデルを補完する．なお以降の Step3-1 と Step3-2 の 2 種類の補完は独立しており，順番は問わない．

Step3-1：ループの繰り返しを考慮した補完
図 2 に示す例で，ループ情報 L の各ループについて処理する．ループ [b] の最初のエッジ b の直前のエッジでありかつ経路モデル P_0 に含まれているエッジを「挿入点」として取得し，この場合エッジ a のみ取得される．次にループを i 回（2 回）繰り返した部分経路群 [b,b] を生成する．ベースとなる経路モデル P_0 から新しい経路（図中の Path2）を派生させるため，部分経路群 [c,d] を複製したうえ，挿入点 a の後段に [b,b] を挿入する．同様に，次のループ [c,e] についても挿入を行った新しい経路（図中の Path3）を追加し，経路モデル P_{1a} を得る．

Step3-2：ループの組み合わせを考慮した補完
図 2 に示す例で，ループ情報 L のループの組合せを考慮した経路を追加する．Step3-1 同様にまずループ [b] の挿入点としてエッジ a を取得する．ベースの経路モデル P_0 から部分経路群 [c,d] を複製したうえ，ループ単体に相当する部分経路群 [b] を生成し，挿入点 a の後段に挿入する．ここで経路 [a,c,d] と [a,b,c,d] を持つ経路モデル P_{1b} が新しいベースとなる．次に，ループ [c,e] の挿入点として同じくエッジ a が取得される．ベースである P_{1b} から部分経路群 [c,d]，[b,c,d] を複製したうえ，ループ単体に相当する部分経路群 [c,e] を生成し，挿入点 a の後段すべてに接続する．結果的に 2 種類のループを片方のみ通る場合と両方通る場合の全組合せを網羅した経路モデル P_2 が得られる．

Step3 の最後に，Step3-1 の P_{1a}（Path1,2,3 を含む）と Step3-2 の P_2（Path1,4,5,6 を含む）をマージすると，図 2 に示す経路モデル P_3 が最終形として得られる．

Step4〜7. 最終形の経路モデル P_n から，それぞれの観点にかなう経路を選出し，1 本の経路を 1 つのテスト項目として生成する．事前準備として，有向グラフを参照して，フロー種別が基本フロー以外（すなわち代替・例外）のエッジを「着目エッジリスト」として把握しておく．Step4 では，各経路に含まれる各エッジを調べ，エッジのフロー種別が基本フローのみで構成されている経路を，全て観点 A として選出する．Step5 では，着目エッジリストが空になるまで，リスト内の各エッジ（代替もしくは例外エッジ）について，該当エッジ以外に含む代替・例外エッジ種類数が最少（一切含まないのがベストだが該当経路が存在するとは限らない）そして最短な経路を 1 本（もしくは 0 本）にまで絞り込み，該当エッジに着目した観点 B として選出する．Step6 では，なるべく多くの種類の代替例外エッジを一筆書きのように通る経路が望ましいため，経路が含む代替・例外エッジの種類が最多そして最短な経路を 1 本抽出し，着目エッジリストに未踏のエッジがなくなるまで，2 本目以降も同様に経路選択を繰り返し，それらを全て観点 C として選出する．Step7 では，着目エッジリストが空になるまで，リスト内の各エッジ（代替もしくは例外エッジ）について，該当エッジの出現回数が指定の繰り返し回数 i と一致しない経路を除外したうえで，該当エッジ以外に含む代替・例外エッジ種類数が最少そして最短な経路を 1 本（もしくは 0 本）にまで絞り込み，該当エッジに着目した観点 D として選出する．補足として，図 2 の例では Step7 完了時に P_3 の 6 本の経路が全て一意な観点に区分されたが，一般的に経路と観点は互いに多重度 [0..*] である．

3 おわりに

ユースケーステスト技法に基づくテスト項目自動生成を目的として，技法で定義された観点を満たす経路を網羅的にかつ効率的に抽出できる経路抽出手法を提案した．33 画面を持つ実際の Web システムを用いて適用評価した結果 [4]，ユースケーステスト技法に基づく自動生成のテスト項目は，従来手作業で作成していたシナリオテストのテスト項目の 87.9%の内容をカバーした．今後は提案手法の資源効率性をさらに高め，開発現場からのフィードバックをもとに改良していきたい．

参考文献

[1] 大西 建児他．ソフトウェアテスト教科書 JSTQB Foundation 第 3 版. 翔泳社, 2011.

[2] Clementine Nebut, Franck Fleurey, Yves Le Traon, and J-M Jezequel. Automatic test generation: A use case driven approach. *Software Engineering, IEEE Transactions on*, Vol. 32, No. 3, pp. 140–155, 2006.

[3] Jim Heumann. Generating test cases from use cases. *The rational edge*, Vol. 6, No. 01, 2001.

[4] 丹野治門, 張暁晶, 星野隆．シナリオテストにおけるテスト項目自動生成手法の提案. 情報処理学会第 76 回全国大会, 2013.