トレース情報を使用したテストケースの入力条件の生成
Generating Precondition of each Test Case by Using Trace Data

海老原裕之† 櫻井孝平‡ 古宮誠一‡
Hiroyuki Ebihara† Kouhei Sakurai‡ Seiichi Komiya‡
† 芝浦工業大学　工学部
‡ 芝浦工業大学大学院　工学研究科
† Faculty of Engineering, Shibaura Institute of Technology.
‡ Graduate School of Engineering, Graduate School of Shibaura Institute of Technology.

要旨
ソフトウェアテストの戦略の１つにホワイトボックステストの分岐網羅がある。分岐網羅を行うにはテストの対象としている経路を通るように分岐条件や演算を考えて入力値を決めなければならない。入力値の定義を支援するために１つのテストケースから新たな経路を実行するテストケースの入力条件を求めるという手法を提案する。具体的にはプログラムの実行履歴とテスト対象のプログラムの解析を行い、推論に必要な情報を生成し、推論を行う。

1. はじめに

ソフトウェアテストとはプログラムに存在しうる欠陥を探し出すために、テスト対象のプログラムをテストケースの下で実行して調べる工程である。テストケースは１つのテストの実行を定める条件と条件から想定される結果の２つの要素からなる。

本研究では充分なテストを行うためのテストケースの定義が困難であるという問題に取り組む。開発者にとってテスト対象としている経路を実行するための入力条件を定義することは容易ではない。入力条件の定義を支援するために１つのテストケースのトレース情報から推論を行うことで新たな経路を実行するテストケースの入力条件を求める手法を提案する。本手法はホワイトボックステストの分岐網羅を対象としている。初めにプログラムのトレース情報を取得して、プログラム解析を行う。プログラム解析後には推論を行うための情報を生成し、その情報を基に推論を行うことでホワイトボックステストのテストケースの入力条件を生成することを目標とする。

2. ソフトウェアテスト

分岐網羅[1]とはプログラム上の分岐の判定条件が真と偽の結果を持つようにして、それぞれの分岐方向が少なくとも１回は実行されるようにテストケースを定義して、テストを行うというものである。分岐網羅の例として図 １のフローチャート図を基に作られたログインのプログラムを挙げる。各命令の側にある数字は命令番号である。このプログラムは ID とパスワードが入力されると ID を頼りにデータベースからデータを取得する。（命令２）次に認証を行い（命令３、５）、認証に成功すればユーザー情報（命令７）、失敗すれば null を返す（命令６）というプログラムである。このプログラムで使用されているデータベースはテスト用のデータベースで、ID には A0001、パスワードには abcde が格納されているものとする。このプログラムの分岐網羅を満たすテスト経路とテストケースは表 １に挙げた３つである。

ソフトウェアテストの問題点のひとつに、特定の経路を実行するテストケースの入力条件を求めることの困難さがある。その理由は開発者が分岐の条件判定をテスト対象の経路が実行されるように調整するために、分岐の条件判定に関わる入力を求め、その入力のデータを分岐と入力の間に存在する演算や他の分岐のことも考えて決めて、入力条件を定義する必要があるからである。例としてログインプログラムのテストケース２をテストすると考える。この場合、分岐２の条件判定が行われる時点で変数 pass と p のデータが一致しないかつ、分岐１の条件判定が true になるように途中の演算や分岐も考慮に入れて入力条件を決め、テストケースを作らなければならない。

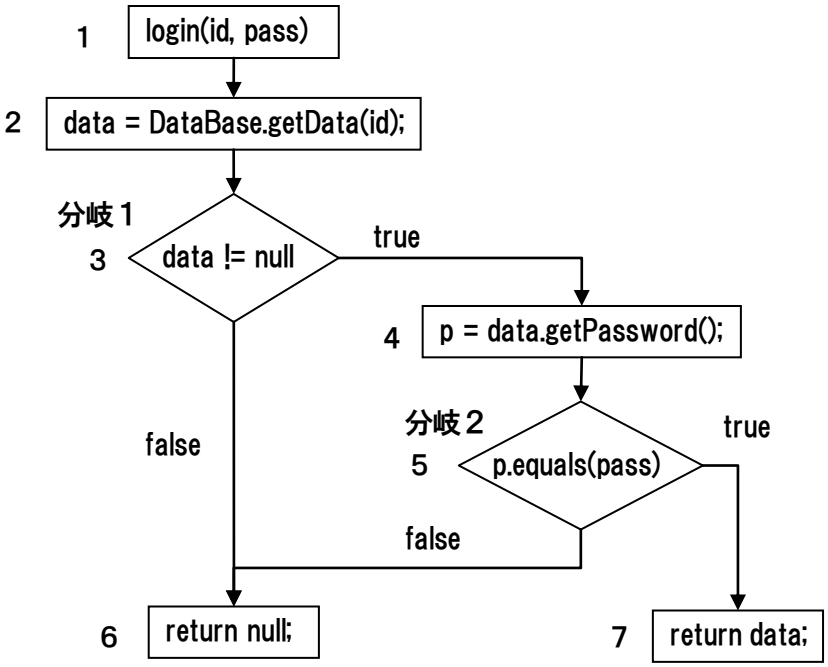

図 1 ログインプログラムのフローチャート図

表 1 テスト経路とテストケース

	テスト経路	入力値	予想結果
テストケース 1	1-2-3-6	ID：B0001、PASS：abcde	null
テストケース 2	1-2-3-4-5-6	ID：A0001、PASS：aaaaa	null
テストケース 3	1-2-3-4-5-7	ID：A0001、PASS：abcde	data

3. 提案手法

テストケースの入力条件の定義を支援するために本研究ではプログラムの実行履歴であるトレース情報とテスト対象のプログラムの解析を用いて推論を行い、新たなテストケースの入力条件を求めることを提案する。この手法はテスト対象のプログラムを実行し、そのトレース情報を取得してプログラム解析を行うことで、推論を行うための情報を生成する。そして生成した情報を使い、まだ実行されていない経路を実効するテストケースの入力条件を求めるという手法である。

3.1. 前提条件

本手法を行うためには前提条件が存在し、それはブラックボックステストなどのテストを事前に行っておくことである。テストを行った後で本手法を使用することでソフトウェアテストのカバレッジを上げることができる。例えば、図 1 のプログラムに対して表 1 のテストケース 3 があらかじめ定義されている場合に、あらたにテストケース 2 を定義することができる。

3.2. 手順

本手法は大きく分けて 4 つのステップからなる。

I. トレース情報の取得（3.2.1 節）
II. プログラム解析（3.2.2 節）
   i. 制御依存関係の探索（3.2.2.1 節）
   ii. データ依存関係の探索（3.2.2.2 節）
III. フローグラフの生成（3.2.3 節）

IV. 入力条件の推論（3.2.4 節）

3.2.1. トレース情報の取得

トレース情報の取得には Traceglasses[3]を使用する。取得するデータは実行された命令や変数の値である。Traceglasses とはプログラムの実行のトレース情報を取得し、実行履歴を図 2 のような木構造で表示することができるデバッガである。ノードは代入文、分岐、メソッド呼び出し、return 文のいずれかで構成されている。ノードにある値は定数や文字列、オブジェクト ID である。図 2 のトレースデータはログインプログラムにて表 1 のテストケース 3 を実行して得た結果である。

```
<4> = Login.login(id:"A0001",pass:"abcde")
    data:<4> = DataBase.getData(id:"A0001")
    if (null == data:<4>)
    p:"abcde" = data:<4>.getPassword()
    true = p:"abcde".equals(pass:"abcde")
    if (0 == 1)
    return data:<4>
```

図 2 Traceglasses で取得したトレース情報

3.2.2. プログラム解析

プログラムの解析のステップでは Soot[4]を使用して制御依存の関係 とデータ依存の関係を探索する。Soot とは Java バイトコードを最適化するためのフレームワークであり、バイトコードの命令を 3 アドレスの中間表現のコードとして扱うことができる。また、 プログラムのコントロールフローを取得することができる。

3.2.2.1. 制御依存関係の探索

制御依存の関係の探索では命令と命令の依存関係を調べる。命令間の依存関係とはとある命令 A が実行されるかどうかが分岐命令 B の結果で決まる場合のことを示す。ログインプログラムでは命令 3 と命令 4、命令 3 と命令 5 などが該当する。それぞれ、後者の命令が実行されるには前者の分岐命令の実行結果が関わってくるため、命令間に依存関係が生じていると定義することができる。

3.2.2.2. データ依存関係の探索

データ依存の関係では命令に使われている変数の定義を追跡することで分岐と入力条件の関係を探索する。探索のアルゴリズムは以下の 6 つのステップである。

1. 分岐命令の探索
2. 分岐命令で使われている変数を取り出す
3. 取り出した変数が定数や入力ならば探索を終了する
4. 取り出した変数が定義されている命令を探索する
5. 命令で使われている変数を取り出す
6. ステップ 3 へ

ログインプログラムの命令 5 を例に挙げると、命令 5 は変数 p と変数 pass を使用している。変数 p を定義しているのは命令 4、変数 pass を定義しているのは命令 1 である。そのため、命令 5 というのは命令 1 と命令 4 にデータの依存関係があると言える。命令 1 と命令 4 で使われている変数にも同じような探索を繰り返していく。

3.2.3. フローグラフの生成

制御依存とデータ依存の関係を探索することで図 3 のようなフローグラフを生成することができる。このフローグラフはノードが命令コードで構成されている。エッジは 2 種類あり、破線の矢印が制御の

依存関係を表わし、直線の矢印がデータの依存を表わす。図 3 の例では、命令５と命令４は命令３の分岐の条件判定が真であることが実行条件であるため、命令５と命令４のノードから命令３へのノードへヘッジができる。また、命令５というのは変数 p と変数 pass を使用している。そのため、データの依存関係として命令５のノードから変数 p を定義している命令４のノード、変数 pass を定義している命令１のノードへのエッジができている。

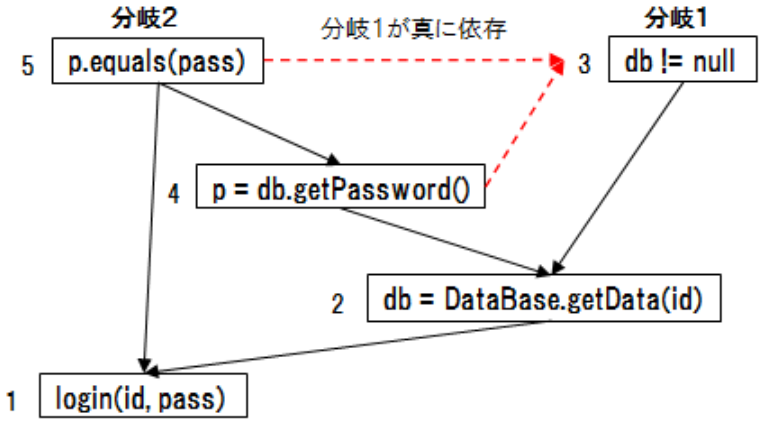

図 3 フローグラフ

3.2.4. 入力条件の推論

入力条件の推論では 3.2.2 のプログラム解析で得たフローグラフを用いて入力条件の推論を行う。例としてログインプログラムの分岐２の推論を挙げる。まず初めにノードの持つコードから命令５は文字列の比較を行っていることがわかる。このときの変数の情報はトレース情報より「p = “abcde”」、「pass = “abcde”」であり、この分岐の実行結果は真である。そのため、今度は偽にすることを考える。条件判定が偽になるには変数 p か変数 pass のどちらかを変えればよいのでここでは変数 pass を変更することにする。変数 pass を定義しているノードを辿ると命令１を持つノードに辿り着く。ここで変数 pass というのはメソッドの引数であることが分かる。よって変数 pass のみを変更すると考えて推論される入力条件は「id = “A0001”、pass = “aaaaa”」である。

4. まとめ

本研究では問題点としてテスト対象としている経路を実行するためのテストケースの入力条件を求めることが容易ではないということを挙げた。入力条件の定義を支援するために１つのテストケースのトレース情報とプログラム解析を用いて推論に必要な情報を作り出し、その情報を使用して新たなテストケースの入力条件の推論を行う手法を示した。今後の課題としては推論を行う情報の検証と推論を行うシステムの作成である。

参考文献

[1] 長尾真[監訳], 松尾正信[訳], “ソフトウェア・テストの技法 第２版”, 近代科学社, 1980
[2] 下村隆夫, “プログラムスライシング技術と応用”, 共立出版株式会社, 1995
[3] 櫻井孝平, 増原英彦, 古宮誠一, “Traceglasses：欠陥の効率良い発見手法を実現するトレースに基づくデバッガ”, 情報処理学会論文誌プログラミング(PRO),3(3),1-17 (2010-06-16), 1882-7802
http://www.komiya.ise.shibaura-it.ac.jp/project/td/traceglasses/
[4] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, Vijay Sundaresan, “Soot – a Java Bytecode Optimization Framework”, 1999
http://www.sable.mcgill.ca/soot/