Vol.2014-SE-186 No.6
2014/11/13

ドメインテスト技法に基づく網羅的なテストデータ自動生成手法の提案

丹野 治門[1,a)]　張 暁晶[1]

概要：本研究では，業務システムにおける画面の入力バリデーションやビジネスロジックのテストにおいて，同値分割と境界値分析に基づいて網羅的にテストデータの自動生成することで，テストデータ作成のコスト削減とテスト品質確保を狙う．既存技術では，画面から入力される変数同士や，ビジネスロジックで扱う変数同士に依存関係がある場合，適切なテストデータを自動生成することができず，現実的なアプリケーションにおいて，特にテストデータ作成に手間がかかる領域が自動化できていないという問題があった．本研究ではこのような問題を解決するため，変数同士に依存関係がある場合でも，依存関係を考慮しかつ境界値等のテストデータとなる値を，制約ソルバを用いることで，網羅的に生成する手法を提案する．

キーワード：ソフトウェアテスト, テストデータ生成，ドメインテスト，同値分割，境界値分析

Automatic Test Data Generation Based on Domain Testing

Abstract: Our research aims to reduce the cost of software while maintaining its quality by automatically and exhaustively generating test data based on equivalence partitioning and boundary-value analysis when testing behaviors of each input validation of a screen and each business logic in enterprise systems. Some approaches which automatically generate test data are proposed. However, these existing approaches cannot handle input variables in which there are a dependence among the input variables. To solve these problem, based on domain testing, we propose an approach to automatically generate test data such as boundary values by using constraint solver even where there are a dependence between the input variables.

Keywords: Software Testing, Test Data Generation, Domain Testing, Equivalence Partitioning, Boundary-Value Analysis

1. はじめに

ソフトウェアの品質を確保するためにテストは重要であり，テストで特に重要なのが，どのようにテストを行うかを考えるテスト設計である．テスト設計では網羅的にテストケースを作成する必要がある．本論文では，テストケースはテストを実行するための具体的なテストデータの値を含むものとして考える．

ソフトウェアが設計通りに実装されているか確認するブラックボックステストを行うときに，少ないテストケースで効率よくテスト設計を行える手法として同値分割法と境界値分析 [2] がある．同値分割法では，期待される結果が同じであるような入力を一つの集団 (同値クラスと呼ばれる) にまとめ，その中から適当に選んだ一つの入力のみを試すことで効率よくテストを行う．境界値分析では，出力が同値になるグループが隣接する境界となる値をテストデータとして用いてテストを行う．また，同値分割法，境界値分析において複数の変数同士に依存関係があるとき，これを特にドメインテスト [2](ドメイン分析テストとも呼ばれる) と呼ぶ．

同値分割法，境界値分析に基づいてテストケースを自動生成する研究としては，設計情報を入力として，テストケースを網羅的に自動生成する研究 [15] があるが，変数同士に依存関係がある場合，すなわちドメインテストにおいて，適切なテストデータを自動生成する技術はないため，現実的なアプリケーションにおいて，変数同士に依存関係

[1] NTT ソフトウェアイノベーションセンタ，東京都港区港南 2-13-34 NSS-II ビル 6F
[a)] tanno.haruto@lab.ntt.co.jp

ⓒ 2014 Information Processing Society of Japan

があり特にテストデータ作成に手間がかかる部分が自動化できていないという問題がある．

変数同士に依存関係がありドメインテストが必要となる問題を，具体例を用いて説明する．カップルである男性，女性それぞれの年齢を入力とし，以下の様なビジネスロジックのルールをもつ映画館チケット購入機能をテストすることを考える．

- 男性の年齢と女性の年齢の合計が 50 歳以下だと映画チケットが割り引き価格になる．
- 男性の年齢は 18 歳以上，女性の年齢は 16 歳以上である．

このビジネスロジックの割り引き価格適用が正常に動作する場合と，準正常系の動作 (上記のいずれかのルールに違反）の場合のテストケースを，ドメインテストの考え方に基づき網羅的に作成すると，上記のルールは直線の式として 2 次元平面上で表現でき，図 1 のようになる．この例では，映画チケットの割引が行われる結果となる同値クラスと，各ルールを違反する準正常の同値クラスが存在し，「男性の年齢と女性の年齢が 50 歳以下」，「男性の年齢が 18 歳以上」，「女性の年齢が 16 歳以上」という条件はそれぞれ同値クラスの境界となる．ドメインテストのにおいて，正常に動作する同値クラスの値，準正常の動作をする同値クラスの値をそれぞれ in ポイント，out ポイントと呼ぶ．また，各境界における境界線上の点を on ポイント，境界線を挟んで最近傍の点を off ポイントと呼ぶ．この例だと，in ポイントは 1 つ (図 1 の in)，out ポイントはそれぞれの境界に対して 3 つ (図 1 の out1,2,3) 存在する．また，on ポイントもそれぞれの境界に対して 3 つ (図 1 の on1,2,3) 存在し，off ポイントは各境界において，それぞれの変数 (男性の年齢，女性の年齢) を正負それぞれの方向に 1 移動させた点として 6 つ存在 (off1,2,3,4,5,6) する．これら in,out,on,off ポイントの選び方はいくつか存在するが，バグ発生時の問題切り分けの観点から，それぞれ互いに重ならないように選ぶことが望ましい．このように，複雑な変数の依存関係や様々な条件を考慮しながら，ドメインテストを用いてテストケースを網羅的に作成するのは大きな労力がかかる．

本研究では，このような問題点を解決し，設計情報からドメインテストの考え方に基づき，テストケースを網羅的に自動生成し，これまでの手動作成作業を自動生成で代替することで，手動作成の労力を削減することを目指す．本研究では，業務システムにおける画面の入力バリデーションやビジネスロジックのテストをスコープとする．

本論文の貢献は以下の 2 点である．

- 入力バリデーションやビジネスロジックを記述できる設計モデルを策定し，複数の変数同士に依存関係がある場合でも，in，out，on，off ポイントに対応するテストケースと，テストケースに含まれるテストデータを網羅的に自動生成する手法を提案した．本手法は，ポイントごとに，そのポイントが満たすべき条件 (例えば，on ポイントならば着目した条件式上のポイントである必要がある) と，互いのポイントが重なり合わないようにする条件の両方を満たすような制約式を構築し，制約ソルバを用いて具体的なテストデータの値を求めることを特徴とする．
- 実開発案件を用いたケーススタディで，提案手法の実現性を確認した．

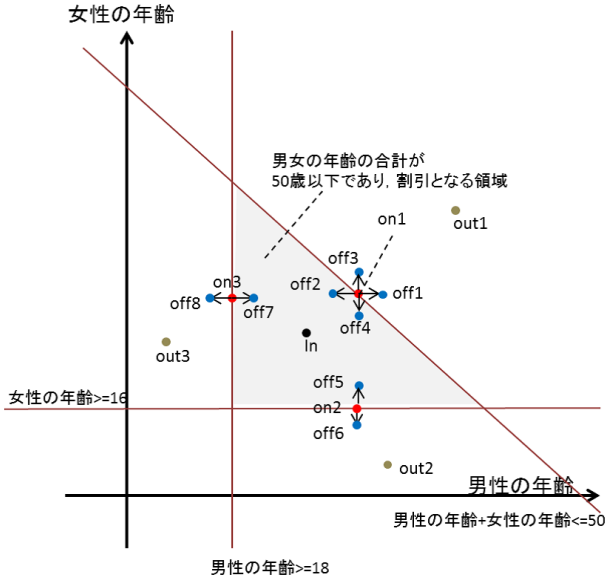

図 1 映画チケット購入機能へのドメインテスト適用例

以降，2 章では，テストデータ生成に関する関連研究について述べる．3 章で，本研究が提案する手法について紹介し，4 章では提案手法のケーススタディとその結果について述べる．そして，最後に 5 章で本論文の結論を述べる．

2. 関連研究

テストデータの自動生成に関する研究 [5] は多く行われている．本章では，本研究がスコープとするブラックボックス観点のテストデータ生成の既存手法と，本研究と技術的に関連があるホワイトボックス観点のテストデータ生成技術についてそれぞれ述べる．

2.1 ブラックボックス観点の既存テストデータ生成手法

同値分割法，境界値分析法の考え方に基づき，テストケースを網羅的に自動生成する手法としては，張らの取り組み [15] がある．この手法では，UML で記述された設計情報から，入力変数の境界条件を抽出し，同値クラスの代表値や，境界値を自動生成することができるが，変数同士に依存関係がある場合，すなわちドメインテストには対応していないという問題がある．本研究では，変数同士に依存関係のあるテスト対象システムに対しても，ドメインテストの考え方に基づき網羅的にテストケースを自動生成す

ることを目指している．

変数同士に依存関係がある場合でもテストケースを自動生成する手法としては，筆者らの過去の取り組み [11,12,14] や，藤原らの取り組み [7,16] がある．これらの手法では，ソフトウェアの設計モデルや，テストの事前条件に基づき，テストケースを自動生成している．これらの手法では，必要な条件を全て制約として集め，それらの制約を制約プログラミング [6] や制約ソルバ [3] などを用いて解くことで，変数の依存関係も考慮したテストデータの具体値生成を行っているが，特定の振る舞いを起こすテストケース，すなわち同値クラスの in ポイントを 1 つのみ生成しており，同値分割法，境界値分析法に基づいた網羅的なテストケースは生成していない．本研究の手法では，これらの手法と同じく制約ソルバを用いており，変数同士の依存関係を考慮した制約に加え，境界値に関する制約も加えることにより，ドメインテストに基づいたテストデータの具体値生成を実現している．

2.2 ホワイトボックス観点の既存テストデータ生成技術

ホワイトボックス観点のテストでは，ソースコードを対象とした単体テストにおけるテストデータ自動生成技術が多く研究されてきた [9]．ホワイトボックス観点の単体テストでは，ソースコードのカバレッジ向上が目的となるため，記号実行 [8]，Concolic Testing [10] など，これらのテストデータ生成技術は静的，動的にプログラム上の分岐条件や変数の依存関係を解析し，よりカバレッジを向上させるためのテストデータを自動生成する．変数の依存関係や，特定の分岐条件を満たす制約を集め制約ソルバを用いて解いてテストデータの具体値を求める点は本研究の技術と近いが，これらのテストデータ生成技術はホワイトボックス観点であるため，本研究がスコープとするブラックボックス観点に基づいた同値分割法，境界値分析とはテストデータ生成の考え方が異なる．

3. 提案手法

本研究では，モデルベーステスト [4] の考え方に基づき，図 2 に示すように，ソフトウェアの設計情報をモデル化した設計モデルから，ドメインテストに基づいてテストモデル (テストケースの集合) を生成する．提案手法の特徴は以下の通りである．

- 業務システムの入力バリデーションやビジネスロジックに相当する設計情報を記述できる設計モデルを策定した．設計モデルは，ソフトウェアが特定の振る舞いを起こすための条件を，変数同士に依存関係がある場合でも条件式として記述することができる．この設計モデルの各条件式を境界条件とみなし，ドメインテストに基づいてテストモデルを生成する．テストモデルは in,out,on,off ポイントそれぞれに対応するテストケース (テストデータの具体的な値を含む) で構成される．
- 本手法は，ポイントごとに，「そのポイントに対応するテストケースのテストデータが満たすべき条件 (例えば，on ポイントならば着目した条件式上のポイントである必要がある)」,，「それぞれのポイントが互いに重ならないようにする条件」の全てを満たすような制約式を構築し，SMT ソルバを用いて制約を充足する具体的なテストデータの値を求める．これにより，生成した各ポイントの具体値はテストデータとして適切であり，かつポイントに対応するテストケースを実施してバグが発生した際に問題の切り分けが容易なテストデータとなる．

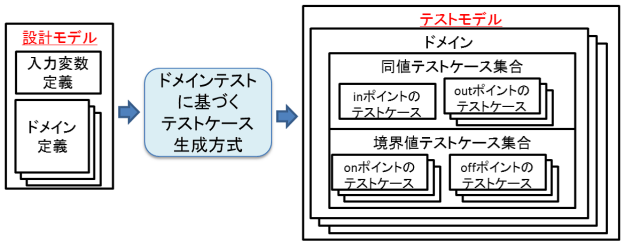

図 2 提案手法の全体像

以降の節で入力である設計モデル，出力となるテストモデル，そしてドメインテストに基づくテストデータ生成方式の詳細についてそれぞれ述べる．

3.1 設計モデル

提案手法の入力である設計モデルの定義を表 1 に示す．また，1 章で紹介した映画チケット購入機能における設計モデルの記述例を図 3 に示す．設計モデルは，入力変数定義，ドメイン定義から構成される．入力変数定義では，システムの入力となる要素を定義する．映画チケット購入機能では，図 3 における男性の年齢と女性の年齢の 2 つが入力変数である．各ドメイン定義では，ソフトウェアが特定の振る舞いを起こすための条件を条件式として複数記述できる．図 3 では，「男性の年齢+女性の年齢<=50 」，「男性の年齢>=18 」，「女性の年齢>=16 」という 3 つの条件式を記述し，これらの条件が全て満たされたとき，「映画チケットは割引き価格となる」という振る舞いが起こるという設計情報になっている．このように，設計モデルでは，現実的なアプリケーションで頻出する変数同士に依存関係があるようなビジネスロジックも表現可能である．

3.2 テストモデル

提案手法の出力となるテストモデルの定義を表 2 に示す．テストモデルは，設計モデルの各ドメインごとの in ポイント，out ポイントのテストケース群である同値テストケース集合，on,off ポイントのテストケース群である境界

ⓒ 2014 Information Processing Society of Japan

表 1 設計モデルの定義

項番	式		
1	＜設計モデル＞	::=	＜入力変数定義＞ ＜ドメイン定義＞+
2	＜入力変数定義＞	::=	＜入力変数＞+
3	＜入力変数＞	::=	InputVariableID：String
4	＜ドメイン定義＞	::=	DomainID:String ＜条件式＞+ Result:String ＜期待結果＞
5	＜期待結果＞	::=	ExpectedResult:String
6	＜条件式＞	::=	ConditionID:String ＜式＞ ＜算術比較演算子＞ ＜式＞
7	＜式＞	::=	＜項＞ ｜ ＜式＞ “＋” ＜項＞ ｜ ＜式＞ “-” ＜項＞
8	＜項＞	::=	＜因子＞ ｜ ＜項＞ “＊” ＜因子＞
9	＜因子＞	::=	＜定数＞ ｜ ＜変数＞ ｜ “(” ＜算術式＞ “)”
10	＜変数＞	::=	InputVariableId:String
11	＜定数＞	::=	ConstantValue:Integer
12	＜算術比較演算子＞	::=	“＞” ｜ “＞=” ｜ “＜” ｜ “＜=” ｜ “==”

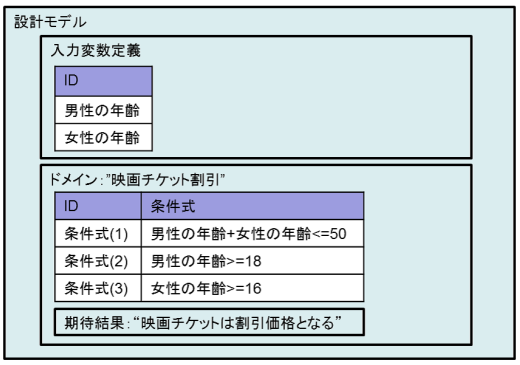

図 3 映画チケット購入機能の設計モデル

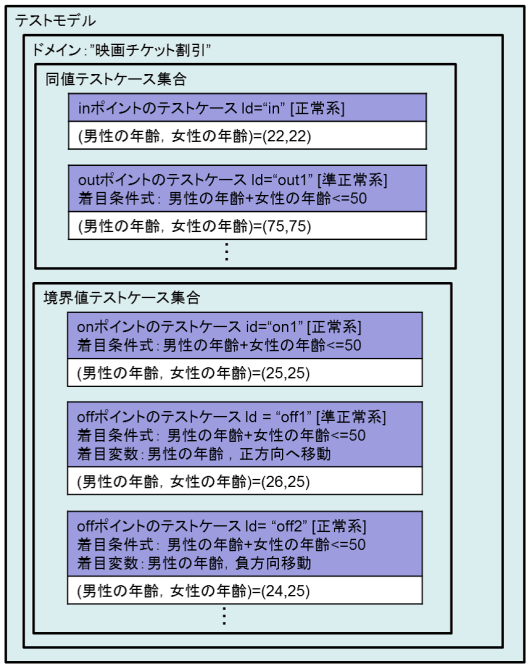

図 4 映画チケット購入機能のテストモデル

値テストケース集合で構成される．映画チケット購入機能におけるテストモデルの出力例を図 4 に示す．このテストモデルにおけるテストケースは，それぞれ図 1 の各ポイントに対応しており，ドメインテストの考え方に基づき網羅的なテストケースを生成している．加えて，各ポイントは互いに重ならないことが保証されている．例えば，図 4 の境界値テストケース集合における Id が"on1" の on ポイントは，着目している条件式が「男性の年齢+女性の年齢<=50 」であり，テストデータの具体値である「(男性の年齢，女性の年齢)=(25,25) 」は「男性の年齢+女性の年齢=50 」という式を満たすポイントであるため，着目している条件式における on ポイントである条件を満たし，かつ他の条件である「男性の年齢>=18 」，「女性の年齢>=16 」も満たすため，適切な on ポイントの具体値となっている．加えて，この on ポイントは他の境界値と重ならないポイントとなっており，テストでバグが発生した場合，問題の切り分けが容易になっている．例えば，(男性の年齢，女性の年齢)=(18,32) は，「(男性の年齢，女性の年齢)=(25,25) 」と同様に，「男性の年齢+女性の年齢<=50 」に着目したときの on ポイントである条件は満たしているが，同時に「男性の年齢>=18 」 に着目したときの on ポイントであることの条件も満たしているため，複数の境界値が重なるポイントとなってしまう．提案手法では，このようなポイントを生成しないようにしている．

3.3 ドメインテストに基づくテストケース生成方式

提案手法では，設計モデルからドメインテストに基づいて網羅的にテストケースを自動生成する．以下，各ポイントを生成するときに共通に用いる定義の説明を行った後，各ポイントの生成方式についてそれぞれ述べる．

3.3.1 準備

本節では，各ポイントを生成するときに共通に用いる定義などを述べる．

- 設計モデルにおいて，入力変数が m 個，条件式が n 個あるとき，それぞれの条件式を $E_1(x_1,...,x_m),...,E_n(x_1,...,x_m)$，もしくは単に

表 2 テストモデルの定義

項番	式		
1	＜テストモデル＞	::=	＜ドメイン＞+
2	＜ドメイン＞	::=	DomainId:String ＜同値テストケース集合＞ ＜境界値テストケース集合＞+
3	＜同値テストケース集合＞	::=	＜ in ポイントのテストケース＞ ＜ out ポイントのテストケース＞+
4	＜境界値テストケース集合＞	::=	＜ on ポイントのテストケース＞ ＜ off ポイントのテストケース＞+
5	＜ in ポイントのテストケース＞	::=	Id:String ＜テストデータ＞
6	＜ out ポイントのテストケース＞	::=	Id:String ＜テストデータ＞ ＜着目条件式＞
7	＜ on ポイントのテストケース＞	::=	Id:String ＜テストデータ＞ ＜着目条件式＞
8	＜ off ポイントのテストケース＞	::=	Id:String ＜テストデータ＞ ＜着目条件式＞ ＜着目変数＞ ＜移動方向＞
9	＜着目条件式＞	::=	ConditionID:String
10	＜移動対象入力変数＞	::=	InputVariableID:String
11	＜移動方向＞	::=	“ 正 ” ｜ “ 負 ”
12	＜テストデータ＞	::=	＜入力変数と具体値＞+ ＜期待結果＞
13	＜期待結果＞	::=	“ 正常 ”｜“ 準正常 ”
14	＜入力変数と具体値＞	::=	＜入力変数＞ ConstantValue:Integer

$E_1, ..., E_n$ と表す．

- 式変換関数として，$Edge(E)$ を条件式における不等号の演算子 $(>=, <=, >, <, =)$ を等号 $=$ へ変換する関数と定義する．例えば $Edge$(男性の年齢 + 女性の年齢 <= 50) は男性の年齢 + 女性の年齢 = 50 となる．また, $Shift(E, x_i,\ op\ 1)$ を E における変数 x_i を $(x_i op\ 1)$ へ置き換える関数，すなわち $E(x_1, ..., (x_i\ op\ 1), ...x_m)$ へと変換する関数として定義する．
- $Shift$, $Edge$ の順に適用する合成関数を $EdgeShift(E, x_i, op1)$ と表記する．例えば，$EdgeShift$(男性の年齢 >=18, 男性の年齢 , +1) は (男性の年齢+1) = 18 となる．
- $Shift(E, x_i, \pm 1)$ は，$Shift(E, x_i, +1)$ *or* $Shift(E, x_i, -1)$ と置き換えるものとする．
- ある着目した条件式 E の on ポイントもしくは off ポイントである制約を表す関数 $OnOrOff(E)$ を $Edge(E)$ *or* $EdgeShift(E, x_1, \pm 1)$ *or ... or* $EdgeShift(E, x_m, \pm 1)$ と定義する．

以降の節では，これらの定義を用いて各ポイントの生成方式を具体的に説明していく．

3.3.2 in ポイントの生成方式

in ポイントは，設計モデルにおいて記述された期待結果となるような同値クラスにおけるポイントであり，他のポイント (on, off ポイント) には重ならないように選ぶ必要がある．そのため，in ポイントの具体値は以下の 2 つの制約を充足するように生成する必要がある．

- (a) 設計モデルにおける全ての条件式を満たす．
- (b) 設計モデルにおける各条件式の境界 (on,off) ポイントとならない．

3.3.1 節の定義を用いると，in ポイントの満たすべき制約 (a),(b) は以下のように設計モデルにおける条件式 $E_1, ..., E_n$ から導出できる．

- (a) E_1 *and* , ..., *and* E_n
- (b) $not(OnOrOff(E_1)$ *or* , ..., *or* $OnOrOff(E_m))$

(a),(b) の制約を *and* でつないだものを制約ソルバへ入力すれば，具体的な解が得られる．

例えば，図 3 で示した映画チケット購入機能の設計モデルから，(a) の制約を導出する場合，「男性の年齢+女性の年齢 <= 50 and 男性の年齢 >= 18 and 女性の年齢 >= 16」という制約を導出する．この制約と (b) の制約を，制約ソルバを用いて解き，図 4 の in ポイント (男性年齢，女性年齢)=(22,22) という，in ポイントとして適切であり，かつ on,off ポイントと重ならないテストデータの具体値を得る．

3.3.3 out ポイントの生成方式

out ポイントは，各条件式についてそれぞれ条件式を満たさないようなポイントを作成するため，以下の (c)，in ポイントにおける (b) という両方の制約を満たす必要がある．

- (c) 設計モデルにおける着目した条件式 E を満たさず，他の条件式は満たす．

3.3.1 節の定義を用いると，条件式 E_1 に着目した out ポイントの満たすべき制約 (c) は以下のように設計モデルにおける条件式 $E_1, ..., E_n$ から導出できる．

- (c) $not(E_1)$ *and* E_2 *and* , ..., *and* E_n

(c),(b) の制約を *and* でつないだものを制約ソルバへ入力すれば，E_1 に着目したときの out ポイントの具体的な解が得られる．条件式 $E_2, ..., E_n$ のそれぞれに着目した out ポイントも同様に導出可能である．

3.3.4 on ポイントの生成方式

on ポイントは，設計モデルにおける各条件式の境界線上となるポイントである．また，各 on ポイントは他の条件式に着目したときの境界 (on,off) ポイントと重ならないよ

ⓒ 2014 Information Processing Society of Japan

うに選択する必要があるため，on ポイントは以下の制約を満たす必要がある．

- (d) 設計モデルにおける着目した条件式 E 上のポイントとなっており，E 以外の条件式を全て満たす．
- (e) 設計モデルにおける着目した条件式 E 以外の境界 (on,off) ポイントとならない．

3.3.1 節の定義を用いると，条件式 E_1 に着目した on ポイントの満たすべき制約 (d),(e) は以下のように設計モデルにおける条件式 $E_1, ..., E_n$ から導出できる．

- (d) $Edge(E_1)$ *and* E_2 *and* $, ...,$ *and* E_n
- (e) $not(OnOrOff(E_2)$ *or* $, ...,$ *or* $OnOrOff(E_n))$

(d),(e) の制約を *and* でつないだものを制約ソルバへ入力すれば，具体的な解が得られる．条件式 $E_2, ..., E_n$ のそれぞれに着目した on ポイントも同様の考えで網羅的に抽出可能である．

3.3.5 off ポイントの生成方式

off ポイントは，設計モデルにおける各条件式の各変数を正負いずれかの方向へ 1 平行移動させた式上のポイントであるため，以下の (f) と， on ポイントにおける (e) の両方の制約を満たす必要がある．

- (f) 設計モデルにおける着目した条件式 E，E において着目した変数 x に関して，E を x 軸の正 (もしくは負) 方向へ 1 移動させた式の上のポイントとなっており，E 以外の条件式を全て満たす．

3.3.1 節の定義を用いると，条件式 E_1 の変数 x_1 に着目し正方向へ 1 移動させた off ポイントの満たすべき制約 (c) は以下のように設計モデルにおける条件式 $E_1, ..., E_n$ から導出できる．

- (f) $EdgeShift(E_1, x_1, -1)$ *and* E_2 *and* $, ...,$ *and* E_n

(f),(e) の制約を *and* でつないだものを制約ソルバへ入力すれば, 具体的な解が得られる．x_1 の負方向，変数 $x_2, ..., x_m$，他の条件式 $E2, ..., En$ にそれぞれに着目した off ポイントも同様の考えで網羅的に抽出可能である．

4. ケーススタディ

本研究の目的は，設計情報からドメインテストの考え方に基づき，テストケースを網羅的に自動生成し，これまでの手動作成作業を自動生成で代替し，手動作成の労力を削減することである．そのため，以下の観点で簡単なケーススタディを題材に評価を行った．

- データ自動生成率：提案した自動化手法で適切なテストデータがどれだけ生成可能かを測定．
- 労力の比較：提案手法で生成したテストケースと，手動作成したテストケースを作成をした場合の労力がどれだけ違うかを比較．今回はテスト設計を行う合計の所要時間を測定することで労力の比較を行った．
- 手動代替効果：提案手法で作成したテストケースは手動作成を代替できるかどうか確認するため，提案手法で作成したテストケースが手動作成したテストケースのどれだけをカバーしているかどうかを調べた．

以降，評価方法，評価結果，考察をそれぞれ述べる．

4.1 評価方法

以下のような手順で評価を行った．

- 手順 1:実開発案件の設計書から入力変数同士に依存関係がありドメインテストが適用可能な評価用の題材を選択する．
- 手順 2:評価用題材からドメインテストの考え方に基づき手作業でテストケース集合 ($\mathbb{M}_{tc}$) を作成する．
- 手順 3:評価用題材から提案手法の設計モデルを作成し，そこから自動でテストケース集合 ($\mathbb{A}_{tc}$) を生成する．

上述した手順で作成したテストケースを用い，テストケース $\mathbb{A}_{tc}$ について,「データ自動生成率」を確認し，$\mathbb{M}_{tc}$ と $\mathbb{A}_{tc}$ の作成所要時間を比較することで「労力の比較」を行う．また，$\mathbb{A}_{tc}$ のテストケースが，$\mathbb{M}_{tc}$ のテストケースのうちどれだけをカバーしているかで「手動代替効果」を確認した．テストケース同士が一致しているかどうかは，各ポイントの具体値を求めるときの元になった領域が同一であるかで判定した．例えば,「男性の年齢+女性の年齢 <=50」という条件式の on ポイントであれば,「(男性の年齢，女性の年齢)=(25,25)」と「(男性の年齢，女性の年齢)=(26,24)」は同一であるとみなす．

評価用題材としては，実開発案件 2 件 (ネットワーク制御システムの管理機能 A と，スケジューラ機能 B) におけるビジネスロジック，入力バリデーションに関する設計書から 3 件の題材を選び，この 3 件と 1 章の例題，計 4 件について評価を行った．4 件の例題の概要を以下に示す．

- 題材 (1)：機能 A 入力バリデーションにおける「リソース容量 (FROM)< リソース容量 (TO)」の条件
- 題材 (2)：機能 A ビジネスロジックにおける「開始時刻 (FROM)< 終了時刻 (TO)」の条件
- 題材 (3)：機能 B 入力バリデーションにおける「開始日付 (FROM)< 終了日付 (TO)」の条件
- 題材 (4)：1 章で述べた映画チケット割引機能へ変数を 1 つ追加し，少し複雑にした題材

提案手法は整数型のみを対象としているため，題材 (2) は，時刻を時分秒へと分割し，題材 (3) は日付を年月日へと分割して扱った．

テストケース自動生成のためのプロトタイプツールを作成した．プロトタイプツールはテキスト形式の設計モデルを入力とし，3.3 節の方式に基づきテストケース (in,out,on,off ポイント) を網羅的に自動生成する．制約を充足するテストデータの具体値を生成するための制約ソルバとしては，SMT ソルバである CVC4 [1] を用いた．

ⓒ 2014 Information Processing Society of Japan

表 3 テストデータ自動生成率

題材	(1)	(2)	(3)	(4)	合計
自動生成率	73%	71%	71%	100%	75%
	11/15	30/42	45/63	18/18	104/138

表 4 手動と自動の所要時間比較 (単位は分)

手動

題材	条件式整理	候補決定	具体値決定	期待結果決定	合計
(1)	3	25	30	5	63
(2)	5	46	40	11	102
(3)	10	71	52	14	147
(4)	5	62	64	19	150
合計	23	204	186	49	462

自動

題材	条件式整理	モデル記述	ツール実行	合計
(1)	3	3	2	8
(2)	5	6	5	16
(3)	10	7	9	26
(4)	5	3	3	11
合計	23	19	19	61

4.2 評価結果

4.2.1 テストデータ自動生成率

テストデータ自動生成率の評価結果を表 3 に示す．テストデータ自動生成率は全体では 75% (104/138) となった．設計モデルにおける条件式から導出した制約式を制約ソルバで解くことができ具体値が生成できたテストデータは，全て適切な値になっていることが確認できた．生成できなかった 25% は制約ソルバで解が存在しなかった (UNSAT となった) テストケースである．

4.2.2 労力の比較

ドメインテストに基づくテストケース作成を手動で行った場合と，自動で行った場合の所要時間を表 4 に示す．4 つの題材の合計で，手動作成の場合は 462 分，自動生成の場合は 61 分となった．

手動では，はじめに設計書の情報を元にドメインテストを適用する上で必要な条件式の整理を行い (表 4 の条件式整理)，次に整理した着目する条件式に基づいて網羅的に in,out,on,off の候補を決定 (表 4 の候補決定) し，その後，各ポイントごとに具体値の決定 (表 4 の具体値決定) を行い，最後に期待結果 (正常系か準正常系) の決定 (表 4 の期待結果決定) を行っており，候補決定，具体値決定に特に時間がかかっている．提案手法を用いた自動生成では，条件式を整理する部分は手動と共通であり，整理した条件式をプロトタイプツールの入力であるテキスト形式の設計モデルとして記述し (表 4 のモデル記述)，その設計モデルからテストケースを自動生成 (表 4 のツール実行) を行っている．提案手法を用いた自動生成では，手動で時間がかかっていた作業が自動化されているため，テストケース作成の労力が大きく削減できていることが確認できた．

表 5 手動と自動のテストケース比較

題材	共通	手動のみ	手動合計	自動のみ	自動合計	自動化率	有効率
(1)	13	2	15	2	15	87%	87%
(2)	34	12	46	8	42	74%	81%
(3)	51	18	69	12	63	74%	81%
(4)	9	7	16	9	18	56%	50%
合計	107	39	146	31	138	73%	78%

4.2.3 手動代替効果

表 5 に手動代替効果の評価結果を示す．自動化率は自動作成のテストケースが手動作成のテストケースのうちどれだけをカバーできたか (共通/手動合計) を示しており，合計では 73%であった．また，有効率は自動生成のテストケースが手動作成のテストケースとどれだけ重なっているか (共通/自動合計) を示しており，合計では 78%であった．

4.3 考察

4.3.1 テストデータ自動生成率

テストデータ自動生成率に関しては，今回，制約ソルバで具体値が求まらなかった一律に自動生成不可とみなしたが，生成できなかったケースを精査したところ，制約を充足する解がそもそも存在しないので，手動で作成も不可であるケースがあるとわかった．また，今回選択した題材では存在しなかったが，生成するポイントのとりうる領域が狭い場合，「他のポイントと重ならないようにする」という制約があると解が存在しなくなるというケースがあることもわかった．例えば，$x = 1$，$x = 2$ という 2 つの条件式があった場合，$x = 1$ に着目した on ポイントと，$x = 2$ に着目したときの off ポイントは必ず重なるため，提案方式では具体値を生成することができなくなってしまう．このような場合には，「他のポイントと重ならないようにする」という制約を除いた上で，再度，制約ソルバによる具体値生成を試みるなど，多少制約を緩和してテストデータを作成するという改良が考えられる．

4.3.2 労力の比較

手動作成の場合，2 変数については図 1 のように図示しながら考えるなどが可能であるが，3 変数以上については図示することも困難であり，各ポイントを網羅的に作成するのは労力のかかることであったが，提案手法では，それらを全て自動で行うため，大きな労力削減につながったと考えられる．

4.3.3 手動代替効果

手動作成のみのテストケース (自動では生成していないテストケース) は，in ポイントを手動で複数作成していたために生じたテストケースであった．in ポイント，out ポイントの数については，ドメインテストで明確には定められておらず，選択の仕方にはある程度バリエーションがあるため，手動と自動でこのような差異がでたと考えられる．どのような選択の仕方がよりテストとして有効である

ⓒ 2014 Information Processing Society of Japan

かは，実際のバグ検出能力にどれだけ違いがあるかなど，様々な観点からより議論を深めていく必要がある．また，テスト実施の自動化が進みあまりコストがかからないのであれば，より多くのポイントを選択し全てのテストを行うという方向性もある．

自動生成のみのテストケース (手動では作成していないテストケース) は，off ポイントについて全変数を各変数ごとに正負方向に 1 移動させたものを網羅的に生成していたために生じたテストケースであり，手動では一部変数のみについてしか off ポイントを作成していなかったためこの差異が発生した．ドメインテストでは，off ポイントは各変数に着目して作成するよう定められているため，手動作成では，off ポイントの作成が漏れていたと考えられる．そのため，提案手法ではこのような漏れがなく網羅的に off ポイントを生成できたので，網羅性という点では，自動化の方がより信頼性の高いテストケースを生成できたといえる．

手動作成と自動生成で共通に存在するテストケースにおいて，in ポイントの具体値に違いが見られた．手動作成では，in ポイントは領域の真ん中あたり (例えば，$1 <= x <= 100$ だったら，$x = 50$) の具体値を作成していることが多かったが，自動生成では，今回用いた制約ソルバの解法アルゴリズムの影響と考えられるが，on,off ポイントの除いた領域の端にある値 (さきほどの例ならば $x = 3$) という具体値が生成された．人が見た時の in ポイントとしてのわかりやすさを考えると $x = 50$ の方が望ましいが，同値分割法における同値クラスの代表値を生成するという in ポイントの性質を考えると $x = 3$ のような値であっても，テストの目的は達成できるため問題ないと思われる．

5. 結論

本研究では，ドメインテストに基づき，設計モデルからテストケースを網羅的に自動生成する手法を提案した．入力バリデーションやビジネスロジックを記述できる設計モデルを策定し，そこから互いのポイントが重ならないよう in，out，on，off ポイントに対応するテストケースを網羅的に自動生成する手法を提案した．本手法は，ポイントごとに適切な値となるための制約と互いのポイントが重なり合わないようにする制約を満たすような制約群を条件式から導出し，SMT ソルバを用いて具体的なテストデータの値を求めることができる．実開発案件，例題を用いたケーススタディでは提案手法の実現性を確認することができた．今後は，より多くの適用案件で評価を行い有効性を確認していきたい．

参考文献

[1] CVC4. http://cvc4.cs.nyu.edu/web/.
[2] Lee Copeland, 雅彦宗. はじめて学ぶソフトウェアのテスト技法. 日経 BP 社, 日経 BP 出版センター (発売), 2005.
[3] Leonardo de Moura, Bruno Dutertre, and Natarajan Shankar. A tutorial on satisfiability modulo theories. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification*, Vol. 4590 of *Lecture Notes in Computer Science*, pp. 20–36. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-73368-3_5.
[4] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. A survey on model-based testing approaches: a systematic review. In *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, WEASELTech '07, pp. 31–36, New York, NY, USA, 2007. ACM.
[5] Jon Edvardsson. A survey on automatic test data generation. In *Proceedings of the Second Conference on Computer Science and Engineering in Linköping*, pp. 21–28. ECSEL, October 1999.
[6] Barry O'Sullivan Frederic Benhamou, Narendra Jussien. *Trends in Constraint Programming*. ISTE, 2010.
[7] Shoichiro Fujiwara, Kazuki Munakata, Yoshiharu Maeda, Asako Katayama, and Tadahiro Uehara. Test data generation for web application using a uml class diagram with ocl constraints. *Innovations in Systems and Software Engineering*, Vol. 7, pp. 275–282, 2011. 10.1007/s11334-011-0162-3.
[8] James C. King. Symbolic execution and program testing. *Commun. ACM*, Vol. 19, No. 7, pp. 385–394, July 1976.
[9] Alessandro Orso and Gregg Rothermel. Software testing: A research travelogue (2000-2014). In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pp. 117–132, New York, NY, USA, 2014. ACM.
[10] Koushik Sen, Darko Marinov, and Gul Agha. Cute: a concolic unit testing engine for c. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, ESEC/FSE-13, pp. 263–272, New York, NY, USA, 2005. ACM.
[11] 石川太一郎, 高田眞吾, 丹野治門, 生沼守英. Concolic testing を用いた web アプリケーションに対するテストデータ生成に関する研究. 情報処理学会研究報告ソフトウェア工学, Vol. 2014-SE-183, No. 1, pp. 1–8, mar 2014.
[12] 丹野治門, 星野隆, Koushik Sen, 高橋健司. Concolic testing を用いた結合テスト向けテストデータ生成手法の提案. 情報処理学会研究報告ソフトウェア工学, Vol. 2013-SE-182, No. 6, pp. 1–8, oct 2013.
[13] 丹野治門, 張暁晶, 星野隆. 設計モデルを利用したテスト用データベース自動生成手法. 情報処理学会論文誌, Vol. 53, No. 2, pp. 566–577, feb 2012.
[14] 丹野治門, 張暁晶, 生沼守英. ソースコード生成を利用した結合テスト向けデータベース生成手法の提案. Technical Report 25, jul 2014.
[15] 張暁晶, 星野隆. 設計モデルを用いたテスト項目抽出とテストデータ生成手法. 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, Vol. 109, No. 41, pp. 37–42, may 2009.
[16] 藤原翔一朗, 宗像一樹, 片山朝子, 前田芳晴, 大木憲二, 上原忠弘, 山本里枝子. 1b-3 smt solver を利用した web アプリケーション用テストデータの生成 (テスト・検証, 一般セッション, ソフトウェア科学・工学, 情報処理学会創立 50 周年記念). 全国大会講演論文集, Vol. 72, No. 1, pp. 1–281–282, mar 2010.

ⓒ 2014 Information Processing Society of Japan