情報処理学会論文誌 Vol.53 No.2 566–577 (Feb. 2012)

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

丹野 治門[1,a)] 張 暁晶[1] 星野 隆[1]

受付日 2011年6月14日, 採録日 2011年11月7日

概要：本研究は，関係データベース（DB）を用いる業務システムの機能テストを対象として，DB への参照を行う各テストケースに対し，適切な DB 初期状態を自動生成する問題を扱う．既存技術では，複数回の DB 参照，文字列の部分一致検索や，DB スキーマにおける主キーかつ外部キーの制約を扱った，複雑なビジネスロジックをテストするための適切な DB 初期状態を生成できないという問題があった．本研究では上述したような複雑なビジネスロジックを記述できる設計モデルを規定し，その設計モデルに記述された制約条件を段階的に解くことで，テストのための適切な DB 初期状態を自動生成する手法を提案する．本手法を用いると業務システムのより多くのテストケースに対して適切な DB 初期状態を生成可能である．実際の業務システム 2 件を用いた本手法の適用評価では，それぞれ全体の 72%，100% のテストケースに対して，事前状態として適切な DB 初期状態を生成することができ，手法の妥当性を確認できた．

キーワード：ソフトウェアテスト，テストデータ生成，データベース，モデルベーステスト，SMT ソルバ

Design Model Based Generation of Initial Database for Testing

Haruto Tanno[1,a)] Xiaojing Zhang[1] Takashi Hoshino[1]

Received: June 14, 2011, Accepted: November 7, 2011

Abstract: This research focuses on how to automatically generate initial test data of relational database properly for each test case, when testing enterprise systems. Existing approaches cannot generate initial database for complicated business logic such as reading database more than once, or, searching database by partial string matching, or primary and foreign key constraints on database scheme. To solve these limitations, we propose a method for initial database generation. This method adopts a design model which can describe complicated business logic given above, and from this design model, our method generates appropriate initial database, by step-by-step solving constraints extracted from the design model. The proposed method enables us to automatically generate initial database for a wide range of test cases in testing of enterprise systems. Using two real enterprise systems as case studies, we confirmed that our method properly generate initial database for 72% to 100% of the test case which needs an initial database.

Keywords: software testing, test data generation, database, model based testing, SMT solver

1. はじめに

関係 DB を用いたソフトウェアのテストを行う際には，確認したいソフトウェアの特定の振舞いを起こすために，テストケースごとに適切な DB 初期状態を用意する必要がある．

DB 初期状態を作成するには 3 つの方法がある．第 1 の方法は，手動でデータを設計し DB 初期状態を用意する方法であり，ユーザがテストケースごとに DB 初期状態を XML で記述して用意する DBUnit [17] はこれに相当する．第 2 の方法は乱数を用いて DB データを自動生成する方法である．この方法では大量の DB レコードを用意することができるので，性能に関する様々な観点を確認する際に有用であり，DBMonster [7]，疑似個人情報ジェネレータ [8]，Visual Studio Database Edition [9] などのツールがある．第 3 の方法は，旧システムの運用で用いられていた DB の一部を使用する方法である．この方法は主に改造開発の場

[1] NTT サイバースペース研究所
NTT Cyber Space Laboratories, Yokosuka, Kanagawa 239–0847, Japan
[a)] tanno.haruto@lab.ntt.co.jp

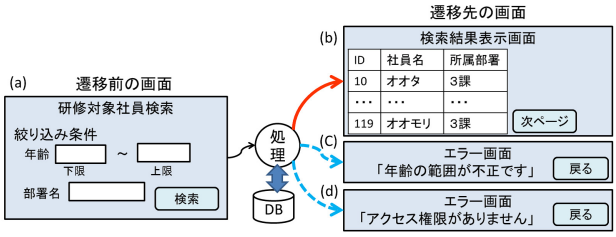

図 1 DB を用いたアプリケーションのテストの例
Fig. 1 Example: testing a database application.

合に用いられ，実運用に近い環境でテストを行うことが可能である．

しかしながら，第 1 の方法ではゼロから DB 初期状態を作成しなければならないためテスト実施者の負担が大きく，第 2，第 3 の方法で作成した場合でも，作成した DB 初期状態が特定のテストケースの事前状態として適切でないときに，DB 初期状態を手動で調整する手間が発生してしまう．

このような問題が起こる具体的な例として，**図 1** のような社内研修の対象者である社員を検索する機能のテストを行う場合を考えてみる．この機能は，研修対象である入社 3 年以内であり，かつ在籍中である社員を検索し，検索結果の人数が多いときには 100 人ずつページを区切って表示し，ユーザは年齢と部署名を指定することで絞り込み検索を行うこともできるという仕様を持つ．この仕様に対して，「検索が成功し，検索結果表示画面（図 1 (b)）に遷移する」というテストを行う場合，「入社年次が 3 年以下かつ在籍という条件と，ユーザが指定した絞り込み条件という 2 つの条件を満たす社員のレコードを 101 件以上持つ」という制約を満たす DB 初期状態が必要である．

前述した第 2，第 3 の方法で用意した DB 初期状態がこのような条件を満たすレコードを偶然持っていればそのままテストを行うことが可能であるが，持たない場合は期待するソフトウェアの振舞いを起こすことができないため，テスト実施者はレコードを適切に追加，削除することで DB 初期状態を調整するか，第 1 の方法で新たに作成する必要がある．

このように，テストケースごとに DB の主キーの制約，外部キーの制約など各種制約を崩さないように，多数のテーブルを手動で調整する，もしくは 1 から作成するのは効率が悪く，ミスも起こりやすいため，テスト実施者にとって大きな負担である．

本研究では，DB が多く用いられる業務システムのテストにおいて，特にテストケース数が多い機能テストを対象とし，テストケースごとに事前状態として適切な DB 初期状態を自動で生成する問題を扱う．

本論文では，テストケースの種類を以下のように分けた．本研究では (1) を対象とし，ソフトウェア設計情報からの DB 初期状態生成を扱う．

(1) DB 初期状態を必要とし，参照系アクセスを扱うテストケース
(2) DB 初期状態を必要とし，更新系アクセスも扱うテストケース
(3) 特定の DB 初期状態を必要としないテストケース

(1) は様々な種類のシステムで用いられる頻度が高く [15]，自動化の効果が顕著であるため，既存技術である程度試みられている領域であるが，実用性を考えると不十分な点があり，詳しくは 2 章で述べる．(2) は更新により，DB の状態が刻々と変わるため，DB 初期状態の生成にあたって，DB のライフサイクルを考慮する必要があるため，(1) とは異なった手法や工夫が必要になると考える．本研究では (2) を対象としない．(2) に対応した DB 初期状態生成の既存研究としては Emmi らの試み [14] があるが，これは，ソースコードのカバレッジ向上を目的としており，DB アクセスを行うソフトウェアを実際に実行させながら，DB 初期状態，入力値を調整していくという手法をとっており，設計情報のみからテストケースごとの適切な DB 初期状態を生成する本研究の趣旨とは異なる．(3) は，DB 初期状態のバリエーションを必要としない箇所であり，任意の DB（空でもよい）を用意しておけばテストが可能なケースで，本研究は (3) を対象としない．前述のとおり，本論文では，DB を用いたアプリケーションで特に多く用いられる参照系アクセスを対象とする．そのため，本論文における適切な DB 初期状態の定義は，「DB スキーマを満たし，かつ DB 参照の検索条件に合う一定件数のレコードを含むもの」となる．

本論文の貢献点は以下のとおりである．

- モデルベーステストの考え方に基づいた既存のテストケース生成手法の設計モデルを拡張し，DB スキーマや DB 参照の検索条件を記述可能にし，DB 参照を行う業務システムの仕様を記述可能にした．提案手法の設計モデルは，業務システムで頻繁に用いられ，かつ従来技術では扱うことのできなかった制約を記述できることを特徴とする．制約は，複数回の DB 参照，DB 検索条件における文字列の部分一致比較，DB スキーマにおける主キーかつ外部キーの制約を含む．
- 提案手法では，DB 初期状態の生成を制約充足問題として扱い，SMT を用いて解く．SMT では，整数の連立不等式や論理式を入力として扱うため，DB レコードや文字列といったデータ構造や，主キー，外部キーといった制約は直接扱うことはできない．そこで，提案手法のアルゴリズムでは，設計モデルから抽出した制約の充足を 3 ステップの部分問題に分割し，DB レコード件数，文字列長という配列の長さに相当する変数の制約を先に SMT が入力として扱える制約に帰着して解くよう工夫しており，テストケースの事前状態として適切な DB 初期状態を求める．

- 提案手法をツールとして実装し，実際の業務システム2件に適用することにより手法の妥当性を確認した．本手法が特徴とする複数回のDB参照，文字列部分一致，主キーかつ外部キー制約は実システムでも多く使われていることが確認でき，これらの制約を扱えるようにしたことは有効であった．

2. テストケースに合うDB初期状態を作成する既存技術の問題点

テストケースそれぞれに合ったDB初期状態作成を支援する既存研究には主に2種類の手法がある．それぞれの手法とその問題点について説明する．

AGENDA [10] では，テストケースごとに，DB初期状態が満たすべき事前条件をユーザが記述し，テストを実施する前に，使用するDBがこの事前条件を満たしているかどうかをチェックできる．この手法の利点は，複数のテストケースに対し1つのDB（たとえば，実際の運用で用いるDBの一部）を用意すればよい点である．しかしながら，この手法ではテストケース数が増えてくるにつれて，事前条件が満たされないことが増えてくるため，失敗が増え，結局手作業での修正が増えてしまう．また，条件に合わない場合，DBデータ生成ツール [7], [8], [9] を用いてDBを生成し直し，事前条件を満たすまでチェックを繰り返す方法もあるが，この方法ではすべての制約を満たすDB初期状態が生成できる可能性があるのか，何回の生成とチェックが必要なのか分からない．

Willmorらの研究 [11] では，ユーザがテストケースごとに事前，事後条件としてDB初期状態が満たすべき制約を記述し，その事前，事後条件に合うように，自動でDBレコードを追加，削除することでDBを調整する．また，藤原らの研究 [13] では，事前，事後条件の制約を満たすようなDB初期状態をテストケースごとに生成する．これらの手法を用いるとAGENDAのような手作業の修正が必要ない．しかし，業務システムのテストを対象にしたとき，文献 [11], [13] の手法では，次のような問題点がある．

- 事前，事後条件だけでは，DB参照を複数回行う複雑なビジネスロジックに対応できない．そのため，図1においてアクセス権限などをDBに問い合わせて確認してから，条件を満たすレコードの検索処理を行う，といった処理の振舞いを確認するテストケースに対応できない．
- 業務システムで頻繁に用いられるような制約のうち対応できないものがある．DBスキーマにおいて「主キーかつ外部キーの制約」は，リソース効率を良くするために1つのテーブルを2つに分割する際によく用いられるが，この制約に対応できない．また，文字列の部分一致比較や四則演算といった制約にも対応できない．

まとめると，Willmorらの手法や藤原らの手法では，対応しない条件に遭遇したとき，テストケースで必要とされるDB初期状態の条件を十分に記述できない場合があり，このような場合，テストケースの事前状態として適切なDB初期状態を生成することができない．

本研究では上にあげる問題点を解決し，業務システムに対する，より多くのテストケースに対して，テストの事前状態として適切なDB初期状態を自動生成することを目指す．

3. 提案するDB初期状態生成手法

本研究では，モデルベーステスト [5] の考え方を用い，評価対象システムの設計情報をモデル化した**設計モデル**から，テストケースとそのテストケースの事前状態として適切なDB初期状態の生成を行う．提案する手法の特徴を以下に示す．

- 提案手法が入力とする設計モデルは，張ら [4] の提案した設計モデルを拡張し，**処理フロー**，**入力定義**，に加えて**DBスキーマ**を記述することができ，処理フローでは，DB参照を複数回行う複雑なビジネスロジックの振舞いを記述できるため，DB参照を行う業務システムを記述することが可能である，DB参照を行うときの検索条件には文字列の比較や四則演算などを用いた多様な条件が記述できる．入力定義は処理フローで用いるユーザからの入力，設定ファイルやシステムから得る入力とその定義域を記述できる．DBスキーマでは，主キー，外部キーといった各種制約を記述可能である．
- 本手法が出力するテストケースは処理フローの各**経路**に対応しており，DB初期状態と入力値から構成される．設計モデルから「適切なDB初期状態」とそれとペアをなす入力値を作るための条件を抽出し，それらの条件を制約充足問題に帰着する．この問題を複数ステップからなる部分問題に分割し，それらを順次，制約ソルバを用いて解く．

これらの特徴により，業務システムにおいて，複雑なビジネスロジックの振舞いを確認するための様々なテストケースに対して事前状態として適切なDB初期状態を生成することが可能である．

図2に手法の全体像を示す．ビジネスロジックの分岐経路を網羅的にテストするため，処理フローから，Decision/Condition Coverage [2] を満たす経路を既存技術 [3] を用いて抽出し，経路ごとにテストケースを生成し，テストケースに含まれるDB初期状態と入力値を生成する．以降，本手法で扱う設計モデル，テストケースの定義，そしてDB初期状態の生成アルゴリズムについて詳しく述べる．

表 1 設計モデルの定義
Table 1 Definition of the design model.

項番	式		
		DB スキーマ	
1	⟨DB スキーマ⟩	::=	⟨ テーブル定義 ⟩*
2	⟨ テーブル定義 ⟩	::=	TableId:String ⟨ カラム ⟩+
3	⟨ カラム ⟩	::=	ColumnId:String ⟨ 型と定義域 ⟩ ⟨ カラムの種類 ⟩
4	⟨ 型と定義域 ⟩	::=	IntegerType ⟨ 整数定義域 ⟩ | StringType ⟨ 文字列定義域 ⟩
5	⟨ 整数定義域 ⟩	::=	MinValue:Integer MaxValue:Integer
6	⟨ 文字列定義域 ⟩	::=	MinLength:Integer MaxLength:Integer
7	⟨ カラムの種類 ⟩	::=	Normal | PK | (FK ⟨ 参照先カラム ⟩)|(PKFK⟨ 参照先カラム ⟩)
8	⟨ 参照先カラム ⟩	::=	TableId:String ColumnId:String
		処理フロー	
9	⟨ 処理フロー ⟩	::=	⟨ ノード ⟩+ ⟨ エッジ ⟩*
10	⟨ ノード ⟩	::=	NodeId:String Text:String NextEdgeId:String*
11	⟨ エッジ ⟩	::=	EdgeId:String NextNodeId:String (⟨ ガード条件 ⟩? | ⟨DB 検索条件 ⟩?)
12	⟨DB 検索条件 ⟩	::=	FromTableId:String+ ⟨Where 句条件群 ⟩ ⟨ 結果件数条件 ⟩
13	⟨Where 句条件群 ⟩	::=	⟨And 制約群 ⟩
14	⟨ 結果件数条件 ⟩	::=	Count (FromTableId:String) ⟨ 整数比較演算子 ⟩ ResultCount:Integer
15	⟨ ガード条件 ⟩	::=	⟨And 制約群 ⟩
16	⟨And 制約群 ⟩	::=	⟨ 制約 ⟩+
		入力定義	
17	⟨ 入力定義 ⟩	::=	⟨ 入力値 ⟩+
18	⟨ 入力値 ⟩	::=	⟨InputValueId:String⟩ ⟨ 型と定義域 ⟩

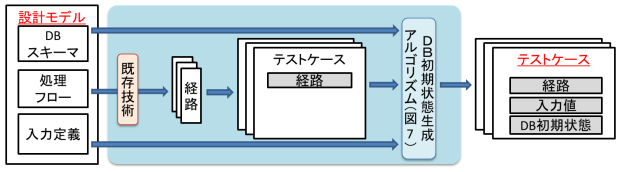

図 2 提案手法の概要
Fig. 2 Overview of our proposed method.

3.1 設計モデル

本手法が入力とする設計モデルの定義を**表 1** に示す．処理フローはノードとエッジで構成される有向グラフによりビジネスロジックを記述する設計情報である．ノードは処理の内容を，エッジはある処理から次の処理への遷移をそれぞれ表す．エッジには，DB 検索条件とその検索条件を満たすレコードの件数に対する制約や，入力値が満たすべきガード条件を記述することができる．DB 検索条件に含まれる Where 句条件群とガード条件には**表 2** に示すように，文字列の部分一致比較，四則演算，内部結合といった様々な条件を扱うことができる．DB スキーマでは，主キー，外部キーの制約に加え，主キーかつ外部キーという制約も扱うことができる．また，本手法では処理フローの 1 経路において，複数回の DB 参照に対応している．

このように，表 1 の設計モデル上では DB を用いるアプリケーションの複雑なビジネスロジックも記述できるようになっている．

具体的な例として，図 1 で紹介した社内研修対象者検索機能の設計モデルを**図 3**，**図 4**，**図 5** に示す．図 3 の処理フローでは，入力値に対して年齢下限が年齢上限以上になっていないかのチェック（図 3 (2)）を行った後に，アクセス権限をチェック（図 3 (4)）し，アクセス権限があるユーザならば社員検索（図 3 (6)）を行うようになっている．ガード条件，DB 検索条件の Where 句条件群では表 2 の制約を用いた条件を記述している．また，図 5 の DB スキーマでは，社員の情報の格納された社員基本情報テーブルや，付随する情報の格納された社員付随情報テーブルなどが定義されており，これらのテーブルには主キーかつ外部キーといった制約が記述されている．

3.2 テストケース

本手法が出力するテストケースは経路，DB 初期状態，入力値の 3 者で構成される．DB 初期状態は DB スキーマを満たし，かつ DB 参照の検索条件（条件には複数の入力値を使用する場合もある）に対して，条件に合う件数のレコードを含むものとなる．そして，すべての入力値は，経路中のガード条件と入力定義における定義域を満たすものとなる．以上により，テストケースの事前状態として適切な，DB 初期状態と入力値の組となっている．

具体的な例として，図 1 において，検索結果表示画面へ遷移する経路（図 3 における (1)，(2)，(4)，(6)，(7) の経路）に対応するテストケースを**図 6** に示す．図 6 のテス

表 2 Where 句条件とガード条件で記述できる制約の定義
Table 2 Definition of the constraints used in the WHERE clauses and the guard conditions.

項番	式		
1	〈 制約 〉	::=	〈 整数制約 〉 | 〈 文字列制約 〉
2	〈 整数制約 〉	::=	〈 整数算術式 〉 〈 整数比較演算子 〉 〈 整数算術式 〉
3	〈 整数算術式 〉	::=	〈 整数項 〉 | 〈 整数算術式 〉 "+" 〈 整数項 〉 | 〈 整数算術式 〉 "-" 〈 整数項 〉
4	〈 整数項 〉	::=	〈 整数因子 〉 | 〈 整数項 〉 * 〈 整数因子 〉 | 〈 整数項 〉 / 〈 整数因子 〉
5	〈 整数比較演算子 〉	::=	">" | ">=" | "<" | "<=" | "==" | "! ="
6	〈 整数因子 〉	::=	ConstantValue:Integer | 〈 入力変数参照 〉 | 〈 文字列長参照 〉 | 〈 カラム参照 〉 | "(" 〈 整数算術式 〉 ")"
7	〈 文字列長参照 〉	::=	Length (〈 文字列因子 〉)
8	〈 文字列制約 〉	::=	〈 文字列因子 〉 〈 文字列比較演算子 〉 〈 文字列因子 〉
9	〈 文字列比較演算子 〉	::=	SubString | EqualsString | NotSubString | NotEqualsString
10	〈 文字列因子 〉	::=	ConstantValue:String | 〈 入力変数参照 〉 | 〈 カラム参照 〉
11	〈 入力変数参照 〉	::=	VariableId:String
12	〈 カラム参照 〉	::=	TableId:String ColumnId:String

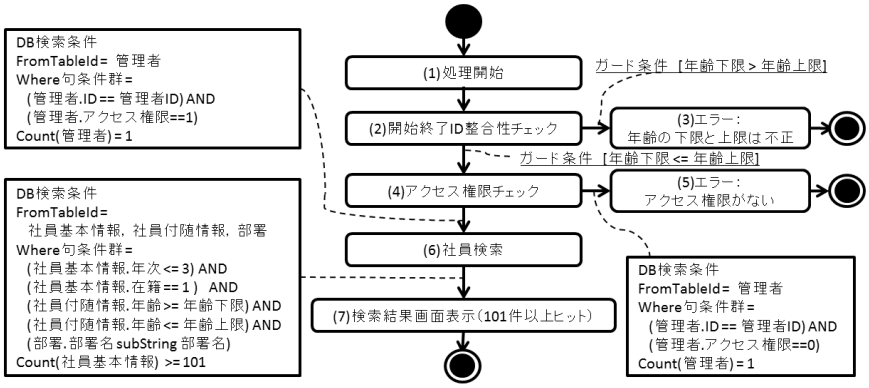

図 3 設計モデル：処理フローの例
Fig. 3 Example: the process flow in the design model.

VariableId	型	定義域
年齢下限	IntegerType	MinValue = 18, MaxValue = 120
年齢上限	IntegerType	MinValue = 18, MaxValue = 120
部署名	StringType	MinLength = 0, MaxLength = 16
管理者ID	IntegerType	MinValue = 50, MaxValue = 60

図 4 設計モデル：入力定義の例
Fig. 4 Example: the input definition in the design model.

トケースの DB 初期状態は，社員基本情報テーブル，管理者テーブルといった直接参照されるテーブルのほかに，部署テーブル，社員付随情報テーブルといった DB のリレーションを満たすために必要なテーブルのレコードも存在する．

3.3 DB 初期状態と入力値の生成アルゴリズム

DB 初期状態と入力値の生成アルゴリズムについて説明する．Samuel ら [6] や，張ら [4] は，処理フローから経路に基づきテストケースを抽出し，入力値を変数と見なし，経路中のガード条件を満たすように値を求めている．本手法では，この考え方を拡張し，**表 3** に示すとおり DB 初期状態，入力値の両方を変数と見なしてこれらの値を求める．本手法では設計モデルから抽出した制約（以後，**制約群**と呼ぶ）と表 3 で示すような変数（以後，**変数群**と呼ぶ）を，変数群に対する連立制約式と見なし，以下の方針に従って解く．

- 整数に対する制約は連立不等式として扱い既存の制約ソルバである Satisfiability Modulo Theories の Arithmetic Theory（以降，単に SMT と呼ぶ）を用いて解を求める．
- テーブルはレコードの配列として扱う．配列構造は一般的には可変長であるため，そのまま連立不等式の解法へと帰着して SMT で解くことはできない [12]．そこで，まずレコード数を配列の長さに関する制約として表現し，個々のレコードが保持するデータよりも先に SMT を用いてレコード数を決定する．次に，レコード数を新たな制約として加え，再度 SMT を用いることでレコードの保持するデータを決定する．
- 文字列は文字の配列として扱う．テーブルと同様に，文字列長は配列の長さに関する制約として表現し，文

TableId = 社員基本情報

ColumnId	型	定義域	カラムの種類
ID	IntegerType	Min = 0, Max = 2000	PKFK 社員付随情報.ID
名前	StringType,	MinLength = 0, MaxLength = 16	Normal
年次	IntegerType	Min = 1, Max = 60	Normal
在籍	IntegerType	Min=0, Max=1	Normal
部署ID	IntegerType,	Min = 0, Max = 1000	FK 部署.ID

TableId = 社員付随情報

ColumnId	型	定義域	カラムの種類
ID	IntegerType,	Min = 0, Max = 2000	PK
年齢	IntegerType,	Min = 18, Max = 120	Normal

TableId = 管理者

ColumnId	型	定義域	カラムの種類
ID	IntegerType,	Min = 0, Max = 60	PK
社員情報アクセス権限	IntegerTYpe,	Min = 0, Max = 1	Normal

TableId = 部署

ColumnId	型	定義域	カラムの種類
ID	IntegerType	Min = 0, Max = 1000	PK
部署名	StringType,	MinLength = 0, MaxLength = 16	Normal

図 5 設計モデル：DB スキーマの例
Fig. 5 Example: the DB scheme definition in the design model.

TestCase01
経路
PathId = "(1) (2) (4) (6) (7)"
入力値
年齢下限= 18
年齢上限= 18
部署名 = "b"
管理者ID = 50

DB初期状態

社員基本情報テーブル

ID	名前	年次	在籍	所属部署ID
10	"a"	1	1	0
11	"a"	1	1	0
…	…	…	…	…
118	"b"	1	1	0
120	"c"	1	1	0

101件

社員付随情報テーブル

ID	年齢
10	18
11	18
…	…
119	18
120	18

101件

管理者テーブル

ID	社員情報アクセス
50	1

部署テーブル

ID	部署名
0	"b"

図 6 テストケースの例
Fig. 6 Example: the test case.

字列の内容よりも先に SMT を用いて文字列長を決定する．文字列を構成する文字は数が多いため，1 つ 1 つについて制約を SMT で記述すると，解を効率良く求められないので，SMT を用いずに後述する独自の手法で決定する．

これらの方針により，本手法では**図 7** で示すように，Step1 で SMT を用いてレコード件数変数の値を求め，Step2 で SMT を用いて文字列長変数と整数変数の値を求め，Step3 で文字変数の値を求める，という段階的な解法で DB 初期状態と入力値を求める．提案手法が特徴とする，複数回の DB アクセス，主キーかつ外部キーの制約は図 7・Step0 の (6) で，文字列部分一致の制約については，図 7・Step2 の (12) で，それぞれ制約充足問題の制約として考慮している．

以降の節で，図 7 における Step0，1，2，3 についてそれぞれ詳しく説明する．

3.3.1 Step0・設計モデルから変数群，制約群を抽出

Step0 は以下の手順（図 7 の (1)～(6)）からなる．

(1) DB スキーマを基に DB 初期状態変数を作成し変数群に追加する．
(2) 入力定義を基に入力変数を作成し変数群に追加する．
(3) DB スキーマからカラムの定義域を抽出し制約群に追加する．
(4) 入力定義から入力値の定義域を抽出し制約群に追加する．
(5) 経路からガード条件，DB 検索条件を抽出し制約群に追加する．
(6) 主キー，外部キーの制約を満たすために，レコード件数変数に関する制約を追加する．

DB の参照整合性制約を満たすために，外部キーの参照先のレコードが少なくとも 1 つは必要になる．そのため，手順 (6) では経路中の DB 検索条件の FromTableId に対応するテーブル変数 t_{src} から外部キー，主キーの参照関係でたどることのできるテーブル変数 t_{target} に対して，「$Count(t_{target}) >= 1$」という制約を追加する．また，外部キーの参照元のカラムが主キーの制約も持っていた場合は，最低，参照元のレコードの数だけ外部キーの参照先のレコードが必要になるため，外部キーの参照元，参照先のテーブル変数 $t1$，$t2$ に対し，「$Count(t1) <= Count(t2)$ レコード件数変数」という制約を追加する．たとえば，図 5 の社員基本情報テーブルと，社員付随情報テーブルがこの関係に相当する．なにも制約が与えられていない DB レコード件数変数 t については，テストケースの事前状態として不要なテーブルのレコードが生成されないようにするため，「$Count(t) == 0$」の制約を生成し制約群に加える．

3.3.2 Step1・レコード件数変数の値を決定

Step1 は以下の手順（図 7 の (7)～(10)）からなる．

(7) レコード件数変数に関する制約 $\mathbb{C}_t$ を制約群から選択する．
(8) $\mathbb{C}_t$ を SMT で解きレコード件数変数の値を決定し，$\mathbb{C}_t$ を制約群から除く．
(9) 参照整合性制約を満たすために，主キー，外部キーであるフィールドに対して制約を追加する．
(10) カラムに対する制約を DB レコードのフィールドに対する制約として展開して制約群に追加し，カラムに対する制約は制約群から除く．

手順 (8) では，DB レコード件数変数に関する制約を連立不等式と見なし SMT に与え解を求める．研修対象社員検索システムで図 6 のテストケースを生成する場合だと，「$Count($ 管理者 $) == 1$」，「$Count($ 社員基本情報 $) >= 101$」，「$Count($ 社員付随情報 $) >= 1$」，「$Count($ 部署 $) >= 1$」，「$Count($ 社員基本情報 $) <= Count($ 社員付随情

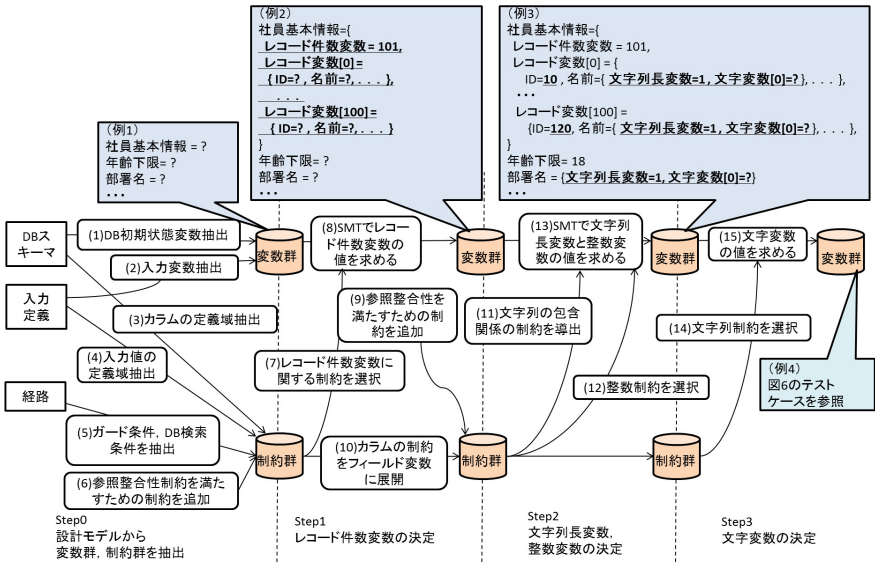

図 7 DB 初期状態生成アルゴリズムの全体像
Fig. 7 Initial database generation algorithm.

表 3 テストケースと変数群
Table 3 Definition of the test case and the variables.

項番	式		
1	⟨ テストケース ⟩	::=	PathId:String ⟨DB 初期状態変数 ⟩ ⟨ 入力変数 ⟩+
2	⟨DB 初期状態変数 ⟩	::=	⟨ テーブル変数 ⟩+
3	⟨ テーブル変数 ⟩	::=	TableId:String ⟨ レコード件数変数 ⟩ ⟨ レコード変数 ⟩*
4	⟨ レコード件数変数 ⟩	::=	⟨ 整数変数 ⟩
5	⟨ レコード変数 ⟩	::=	Index:Integer ⟨ フィールド変数 ⟩+
6	⟨ フィールド変数 ⟩	::=	ColumnId::String (⟨ 文字列変数 ⟩ |⟨ 整数変数 ⟩)
7	⟨ 入力変数 ⟩	::=	InputValueId:String (⟨ 文字列変数 ⟩ |⟨ 整数変数 ⟩)
8	⟨ 文字列変数 ⟩	::=	⟨ 文字列長変数 ⟩ ⟨ 文字変数 ⟩*
9	⟨ 文字列長変数 ⟩	::=	⟨ 整数変数 ⟩
10	⟨ 整数変数 ⟩	::=	Value:Integer
11	⟨ 文字変数 ⟩	::=	Index:Integer Value:Char

報)」という 5 つの式からなる連立不等式を SMT で解く．

手順 (9) では，主キーである各フィールドが確実に一意になるようにするため，n 件のレコード変数を持つテーブル変数の，主キーであるカラムに対応するフィールド変数 $pk_1,\ldots,pk_n$ に対して，IntegerType の場合は「$pk_1 < pk_2$」,...,「$pk_{n-1} < pk_n$」，StringType の場合は「pk_1 $subString$ c_1」,...,「pk_n $subString$ c_n」($c_1,\ldots,c_n$ は互いに異なる定数文字列)，という制約を新たに生成し制約群に追加する．また，外部キーによる参照元のフィールド変数と参照先のフィールド変数の値が等しくなるようにするため，外部キーであるカラムに対応するフィールド変数 fk に対して，参照先のテーブル変数のフィールド変数 pk を任意に選び（参照元が主キーかつ外部キーのカラムに対応していた場合は，それぞれ別々のフィールド変数 pk を選ぶ）IntegerType ならば「$fk == pk$」，StringType ならば「fk $equalsString$ pk」という制約を新たに生成して制約群に追加する．

手順 (10) では，DB 検索条件，DB スキーマにおけるカラムの定義域から抽出したカラム c への制約「c $Operator$ x」を，カラム c に対応する各フィールド変数 $f_1,\ldots,f_n$ への制約「f_1 $Operator$ x」,...,「f_n $Operator$ x」として展開し，展開した制約を制約群へと追加する．研修対象社員検索システムで図 6 のテストケースを生成する場合だと，「社員基本情報. 年次 <= 3」という制約を，「社員基本情報. レコード変数 [0]. 年次 <= 3」,...,「社員基本情報. レコード変数 [101]. 年次 <= 3」という制約へと展開する．

Step1 が終了するとテーブル変数のレコード変数の値が決まるため，変数群は図 7 の例 1 の状態から例 2 のような入力変数とテーブル変数の各フィールド変数が未決定の状態となる．

3.3.3 Step2・文字列長変数，整数変数の値を決定

Step2 は以下の手順（図 7 の (11)〜(13)）からなる．

(11) 文字列変数の包含関係に基づき文字列長変数に関する整数制約 $\mathbb{C}_l$ を生成し制約群に追加する．

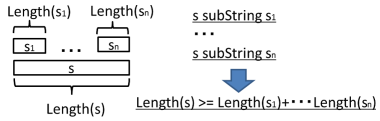

図 8 文字列の包含関係の制約を導出

Fig. 8 Deriving new constraints from the relationship among strings.

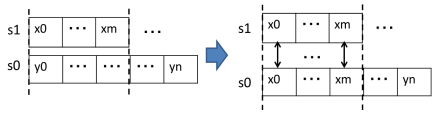

図 9 文字変数の値を決める

Fig. 9 Deciding the value of each character.

(12) 制約群からすべての整数制約 $\mathbb{C}_i$ を選択する（$\mathbb{C}_i$ には $\mathbb{C}_l$ も含まれる）．

(13) $\mathbb{C}_i$ を SMT で解き文字列長変数，整数変数の値を決定し，$\mathbb{C}_i$ を制約群から除く．

部分文字列の文字列長の和が，それらを含む文字列の文字列長を超えないようにするため，手順 (11) では，**図 8** に示すある文字列変数 s に部分文字列として $s_1, \ldots, s_n$ を含んでいたとき文字列変数どうしの包含関係に基づき「$Length(s) \geq Length(s_1) + \ldots + Length(s_n)$」という整数制約を導出し，制約群へ加える．文字列どうし x，y が等しい場合は $Length(x) == Length(y)$ となる．たとえば，図 3 から手順 (5) で抽出し，手順 (10) で展開した「部署. レコード変数 [0]. 部署名 subString 部署名」という制約からは「$Length$(部署. レコード変数 [0]. 部署名) $\geq Length$(部署名)」という制約を導出する．

Step2 が終了すると，すべての文字列長変数，整数変数の値が決まるため，変数群は図 7 の例 2 の状態から例 3 に状態に変わり，未決定なのは文字列変数を構成する文字変数のみとなる．

3.3.4 Step3・文字変数の値を決定

Step3 は以下の手順（図 7 の (14)〜(15)）からなる．

(14) 制約群から文字列制約 $\mathbb{C}_s$ を選択する．

(15) $\mathbb{C}_s$ に基づき文字変数の値を決定し，$\mathbb{C}_s$ を制約群から除く．

手順 (15) では，**図 9** のように，ある文字列 $s0$ と別の文字列 $s_1, \ldots, s_n$ の相関関係があるとき，部分一致や等価（表 2 の $subString$, $equalsString$）であれば，それぞれの位置を揃えてから，対応する文字が同じになるよう文字変数の値を決定し，部分一致や等価の否定（表 2 の $notSubString$, $notEqualsString$）の場合は，1 カ所文字が異なるよう文字変数の値を決定する．

Step3 が終了すると，制約群が空になり変数群のすべての値が決定し，図 7 の例 3 の状態から例 4 の状態に変わり，図 6 のようにテストケースの事前状態として適切な DB 初期状態と入力値が求まる．

4. 評価

提案手法に対する評価観点および評価結果を示し，考察を述べる．

4.1 観点

本研究では，業務システムにおけるより多くのテストケースに対してテストケースの事前状態として適切な DB 初期状態を自動生成することを目指している．そのため，評価観点は以下の 2 点である．

- DB 参照を行うテストケースのうちどれだけに対して DB 初期状態を生成できたか．
- 生成した DB 初期状態はテストケースの事前状態として適切であるか．

4.2 方法

前章で解説した提案手法に基づいて DB 初期状態生成ツールを作成した．整数制約を解く際（3.3.2 項，3.3.3 項）に用いる SMT に基づく制約ソルバとして，Z3 [16] を用いた．

本手法は DB への参照系アクセスを行うテストケースを対象としているため，2 つの業務システム（以後，A，B と呼ぶことにする）のうち参照系アクセスが多い機能をそれぞれ選び評価に用いた．これら 2 つの業務システムについては，以下に述べるように，DB 参照の検索条件や DB 定義の複雑さが異なるものを意図的に選択した．

- A はネットワーク機器管理システムのオペレータ向けの Web フロントエンドであり，IP アドレス，サービス名などを指定してサービス情報の検索を行うなど，比較的単純な検索処理を主に行うシステムである．
- B はスケジューラソフトであり，日時の範囲を指定して検索を行ったり，ユーザごとにスケジュールが管理されていたりと，A よりも検索処理や DB 定義が複雑なシステムである．

本手法のプロトタイプツールでは XML 形式の設計モデルを入力とする．そのため，A，B の設計書から既存技術 [3] を用いて処理フローの経路を抽出した後，手作業で XML 形式の設計モデルを記述し，さらに A，B の設計書から DB スキーマと処理フローの経路中の検索条件に相当する設計情報を抽出して設計モデルに書き加えた．この設計モデルを入力として DB 初期状態生成ツールに DB 初期状態を生成させ，それが正確に経路を通るもの，すなわちテストケースの事前状態として適切なものになっているかを確認した．なお，作業はすべて筆者と異なる業務システム開発経験者 1 名が行った．DB 初期状態生成ツールを動作させた環境は CPU が Intel Core i7 3.0 GHz，メモリは 6 GB，OS は Windows Vista Ultimate SP2 である．

試作したツールではデータ型として，整数型，文字列型，

表 4 テストケースの分類
Table 4 The number of the test cases in category.

項番	分類	件数	
		A 4 テーブル，11 画面	B 12 テーブル，12 画面
(1)	DB 初期状態を必要とし，参照系アクセスを扱うテストケース	74	83
(2)	DB 初期状態を必要とし，更新系アクセスも扱うテストケース	7	4
(3)	特別な DB 初期状態を必要としないテストケース	147	292
	テストケースの合計	228	379

表 5 評価結果
Table 5 Evaluation results.

	A	B
提案手法が対象とするテストケースの合計	74	83
DB 初期状態生成率	100% (74/74)	72% (60/83)
生成時間	0.1 s/テストケース	0.1 s/テストケース

表 6 本手法が特徴とする機能の使用頻度
Table 6 The frequency in use of the features which characterize our proposed method.

	A	B
総数	74	60
DB 参照を 2 回以上行う	39/74 (47%)	39/60 (65%)
主キーかつ外部キーのカラムを持つ DB テーブルへアクセスする	28/74 (38%)	0/60 (0%)
文字列部分一致比較を行う	12/74 (16%)	10/60 (17%)

テーブル型のみを扱っており，その他のデータ型を扱うことができない．そこで，列挙型については文字列型として，その他の基本データ型は整数型で扱うことで代用した．

4.3 結果

今回扱った A，B それぞれのテストケース（設計モデルの処理フローにおける 1 経路が 1 テストケースに対応する）を 1 章で述べたテストケースの分類に基づいて**表 4** のとおり分けた．表 4 (1) は提案手法が対象とする DB 初期状態を必要とし，参照系アクセスを行うテストケースである．表 4 (2) は DB 初期状態を必要とし，更新系アクセスも扱うテストケースであり，1 章で述べたように今回は対象としない．表 4 (3) は特別な DB 初期状態を必要としないテストケースであり，DB 接続エラー，外部ファイル取得失敗エラー，セッションエラー，不正な入力値が入力されたときの検証が大部分を占めた．これらは任意の DB を用意しておけば実施可能なテストケースであるため，本手法では特に対象としない．表 4 (1) の評価結果を**表 5** に示す．

A では，本研究が対象とするテストケースの（表 5 の 74 件）のうち 100%に対してに対して本手法が適用可能であり，テストケースの事前状態として適切な DB 初期状態が生成できた．*B* では，本研究が対象とするテストケースの（表 5 (1) の 83 件）のうち，72%（60 件）に対して本手法が適用可能であり，事前状態として適切な DB 初期状態が生成できた．DB 初期状態生成の実行時間は，1 テストケースあたり 0.1 秒程度であり，1,000 KL 規模のソフトウェアの結合テストの全テストケース 58,000 件（結合テストでは，1 KL あたりのテストケース数は約 58 件 [1]）すべてに適用すると考えても 2 時間弱ほどとなり，夜間バッチ処理として運用すれば問題ないレベルである．

また，本手法が特徴とする制約の使用頻度を**表 6** に示す．*A* で，DB 初期状態が生成できたテストケース 74 件のうち，DB 参照が 2 回以上行われるテストケースが 47%，主キーかつ外部キーの制約を持つテーブルのレコードが必要となるテストケースは 38%，文字列の部分一致を用いて検索が行われるテストケースは 16%存在した．*B* で，DB 初期状態が生成できたテストケース 60 件のうち，2 回以上行われるテストケースが 65%あり，文字列の部分一致を用いて検索が行われるテストケースは 17%あった．

4.4 考察

4.4.1 対応できた制約とできなかった制約

評価に用いた *A*，*B* のテストケースは，システム・ユーザのアクションによってシステムがある画面から次の画面へ遷移するという単発のインタラクションが正しく動作するかという観点で作成されていた．そのため，最初にアクセス権限などのチェックを行ってから次に本来の目的である検索を行うというパターンが多く，このとき参照するのはそれぞれ別のテーブルであったため，複数回 DB アクセスに対応している本手法で多くのテストケースに対し適

切な DB 初期状態を生成することができた．また，DB スキーマにおける主キーかつ外部キーの制約，文字列の部分一致を行う検索条件といった本手法の特徴も，表 6 のように出現頻度から考えると有効であった．

B では，A に比べて複雑な DB 定義を設計情報として持っており，「ユーザと，ユーザごとの予定」といった 1 対多のリレーションを，DB 定義上で，複合キーと主キーかつ外部キーの制約を併用して表しており，本手法を適用することができなかった．このことが，今回の評価において，DB 初期状態の生成率が A で 100%，B で 70%という結果の差違につながった．1 対多のリレーションを表現するときに用いる複合キー制約への対応は今後の課題といえる．

4.4.2 バックトラック

本手法では，3.3.2 項における DB レコード件数の決定，3.3.3 項における文字列長の決定において，バックトラックが必要となる可能性がある．しかし，バックトラックが必要となる場面は稀であり，今回の評価で試行した範囲内でも，バックトラックをしなくても十分な DB 初期状態が生成できた．

3.3.2 項では，SMT で DB レコードの件数をある値に決め打ち，次の処理に進んでいる．そのため，たとえば，あるテーブルにユニークな整数型のカラムが含まれており，その定義域が 1 以上，10 以下となっているときに，決め打ちでそのテーブルのレコード件数が 100 と決定された場合，そのカラムに対応する各レコードのフィールドに対して一意な値が生成できなくなるため，バックトラックでレコード件数を決め直す必要がある．しかし，今回使用した SMT ソルバ・Z3 は解の候補が複数あった場合に，絶対値が最小である値を選ぶ仕様になっているため，このようなことは起こらない．

3.3.3 項の文字列長の決定では，たとえば設計情報として文字列 x, y, z について，「z $subString$ x, z $subString$ y」という文字列の包含関係に関する制約と，「$Length(x) == 1, Length(y) == 1, Length(z) == 1$」という文字列長に関する制約が同時に存在したとき，これらの文字列長に関する制約を「$Length(x) + Length(y) <= Length(z)$」と決め打つと，解が存在しなくなるため，バックトラックを行い代わりに「$Max(Length(x), Length(y)) <= Length(z)$」といった制約を用いて，文字列長を決定し直す必要がある．しかし，文字列は Web 画面のフィールドであり入力できる文字列の長さの自由度は高いことが多く，このような状況になることは稀であると考えられる．

仮にバックトラックが発生するようなことがあった場合，各テストケースの DB 初期状態の生成時間にタイムアウトを設けることにより，限られた時間でなるべく多くの DB 初期状態を生成するなどの対処法がある．

4.4.3 生成時間

各テストケースは，その目的によって必要な量の DB レコードを用意するだけで十分である．たとえば，ログイン処理，アクセス権限のチェックなら 1 件，検索結果ならば十数件程度のレコードが事前状態としてあればよいというパターンが多かった．本手法では大量にレコードを生成するツール [7], [8], [9] と異なり，必要のないテーブルのレコードは生成せずテストケースごとに必要最低限のレコードのみを生成しているため，DB 初期状態の生成時間は実用的な範囲内に収まったと考えられる．

4.4.4 生成した DB 初期状態の質的側面

本手法により自動生成した DB 初期状態は，手動で用意する DB レコードに対して，以下のような点で優位性があると考えられる．

- 各テストケースに対して必要最低限のレコードのみを生成しているため，バグが発生した際に原因を特定しやすくなる可能性がある．手動の場合は，多くのレコードが用意された DB の中から，それらのレコードの一部をテストに使用することが多いため，1 度バグが発生すると，多くのレコードから原因を特定するのに手間がかかる．
- テストデータ作成の属人性が排除できる．今回用いた SMT ソルバである Z3 は，同じ制約式に対しては必ず一意の解を得ることができるため，提案方式では設計モデルが同じであれば，どのような環境でも同じ DB 初期状態を得ることができる．

また，現時点では以下のような不足点も考えられる．

- 生成した DB 初期状態は，検索条件に一致するレコードしか入っていないため，本来条件に合わないレコードが誤って検索結果に含まれることがないかの確認を行うことはできない．手動で行う場合には条件に一致しないレコードを最低 1 つ入れておくことも多い．このような確認も行うことのできる DB 初期状態を生成するためには，レコード件数を決定する際に件数を増やしておき，増やした分のレコードについては，検索条件の not をとった制約を追加したうえで，生成を行うなどの工夫が考えられる．
- 生成した DB 初期状態における各レコードのフィールドに入る値のバリエーションが少ない．Z3 は解の候補が複数あった場合に，絶対値が最小である値を選ぶ仕様になっているため，結果的に図 6 の例のように，多くのフィールドが同じ値になることが多い．テストデータとしては，境界値などを含む値のバリエーションが存在しているとより有用であるので，境界値の制約などを追加したうえで制約ソルバで解くなどの改良が考えられる．

これらに対応するのは今後の課題である．

5. 結論

本論文では，DB へ参照系アクセスを行う機能性テスト

を対象とし，テストケースごとに事前状態として適切なDB初期状態を，ソフトウェアの設計モデルから自動で生成する手法を提案した．

提案手法では，モデルベーステストの考え方に基づいた既存のテストケース生成手法の設計モデルを拡張し，DBスキーマやDB参照の検索条件を記述可能にし，DB参照を行う業務システムを記述可能にした．提案手法の設計モデルは，業務システムで頻繁に用いられ，かつ従来技術では扱うことのできなかった制約を記述できることを特徴とする．制約は，複数回のDB参照，DB検索条件における文字列の部分一致比較，DBスキーマにおける主キーかつ外部キーの制約を含む．提案手法のアルゴリズムでは，設計モデルから抽出した制約の充足を3ステップの部分問題に分割し，DBレコード件数，文字列長という配列の長さに相当する変数の制約を先にSMTが入力として扱える制約に帰着して解くよう工夫している．提案手法をツールとして実装し，実際の業務システム2件に適用することにより手法の妥当性を確認した．本手法が特徴とする制約は実システムでも多く使われていることが確認でき，これらの制約を扱えるようにしたことは有効であった．

今後は，より多くのテストケースにおいてDB初期状態が生成できるように，設計モデルで扱える制約を増やし，適用範囲を拡大していきたい．

参考文献

[1] 情報処理推進機構ソフトウェア・エンジニアリング・センター：ソフトウェア開発データ白書2009，日経BP出版センター (Sep. 2009).

[2] 情報処理推進機構ソフトウェアエンジニアリングセンター：ソフトウェアテスト見積りガイドブック—品質要件に応じた見積りとは，オーム社 (Sep. 2008).

[3] 丹野治門，張　暁晶，星野　隆：結合テストにおけるテスト項目自動生成手法の提案と評価，信学技報，Vol.110 of SS2010-34, pp.37–42 (2010).

[4] 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，信学技報，Vol.109 of SS2009-7, pp.37–42 (2009).

[5] Neto, A.C.D., Subramanyan, R., Vieira, M. and Travassos, G.H.: A survey on model-based testing approaches: A systematic review, *Proc. 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, pp.31–36 (2007).

[6] Samuel, P. and Mall, R.: Slicing-based test case generation from uml activity diagrams, *SIGSOFT Softw. Eng. Notes*, Vol.34, No.6, pp.1–14 (2009).

[7] DBMonster, available from ⟨http://dbmonster.kernelpanic.pl/⟩.

[8] PPDGen：疑似個人情報ジェネレータ，入手先 ⟨http://www.start-ppd.jp/⟩.

[9] Visual studio database edition, available from ⟨http://www.microsoft.com/japan/msdn/vstudio/⟩.

[10] Chays, D., Dan, S., Frankl, P.G., Filippos, I.V. and Elaine, J.W.: A framework for testing database applications, *Proc. 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp.147–157 (2000).

[11] Willmor, D. and Embury, S.M.: An intensional approach to the specification of test cases for database applications, *Proc. 28th International Conference on Software Engineering*, pp.102–111, ACM (2006).

[12] Edvardsson, J.: A survey on automatic test data generation, *Proc. 2nd Conference on Computer Science and Engineering in Linkoping*, pp.21–28, ECSEL (1999).

[13] 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：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，情報処理学会創立50周年記念（第72回）全国大会講演論文集 (2010).

[14] Emmi, M., Majumdar, R. and Sen, K.: Dynamic test input generation for database applications, *Proc. 2007 International Symposium on Software Testing and Analysis*, pp.151–162 (2007).

[15] Chen, S., Ailamaki, A., Athanassoulis, M., Gibbons, P.B., Johnson, R., Pandis, I. and Stoica, R.: TPC-E vs. TPC-C: Characterizing the New TPC-E Benchmark via an I/O Comparison Study, *SIGMOD Record'10*, Vol.39, No.3 (Sep. 2010).

[16] Z3, available from ⟨http://research.microsoft.com/en-us/um/redmond/projects/z3/⟩.

[17] DBUnit, available from ⟨http://dbunit.sourceforge.net/⟩.

丹野 治門 （正会員）

2007年電気通信大学電気通信学部情報工学科卒業．2009年同大学大学院電気通信学研究科情報工学専攻博士前期課程修了．同年日本電信電話株式会社入社．2008年未踏ユース・スーパークリエータ認定（情報処理推進機構），2009年山内奨励賞（情報処理学会）．プログラミング言語，デバッガ，ソフトウェアテスト自動化に関する研究開発に従事．

張 暁晶

2005年九州大学工学部電気情報工学科卒業．2007年同大学大学院システム情報科学府情報工学専攻博士前期課程修了．同年日本電信電話株式会社入社．2010年善吾賞（テスト技術者振興協会）．モデルに基づくソフトウェア開発支援技術に興味を持ち，主にテスト自動化に関する研究開発に従事．

星野 隆 （正会員）

1989 年電気通信大学電気通信学部通信工学科卒業．1991 年同大学大学院電気通信学研究科電子情報学専攻博士前期課程修了．同年日本電信電話株式会社入社．データベース，データ流通方式，ソフトウェア開発技術に関する研究開発に従事．