定理証明を用いた
システム状態の一貫性検証に関する研究

M2013SE012 渡邉大輔
指導教員：沢田篤史

1　はじめに

近年，ソフトウェアの高い信頼性を確保するための取り組みの一つとして，システムがユーザの要求などを満足しているかを論理的に検証する形式検証がある．形式検証では, ソフトウェアの仕様や設計の正しさを検証でき, 仕様書や設計書，プログラムの記述の品質を向上できる [1][2][3]. このような検証をサポートする技術として，定理証明やモデル検査というものがある．モデル検査では，仕様の範囲の中でシステムが振る舞うことを総当たり方式で検証し，定理証明では形式化したシステムの性質に対し証明を与えることでシステムが仕様を満たすことを検証する．定理証明はモデル検査と比較してシステムの状態の規模の大きさに関わらず検証を行える利点があるが，証明手順の学習や証明の記述など，証明に要するコストが高い問題がある．

本研究では，特にシステムの仕様に対する検証に着目する．そこで，本研究ではシステムの仕様をシステムの環境と状態，操作の 3 つによって構成されるものとして考える．環境の仕様記述の中にはシステムの内部構造がデータの型や関連，入出力として定義される．同様に，状態にはシステムが常に満たすべき性質が状態不変条件として定義され，操作にはその前後にシステムが満たすべき性質が事前，事後条件として各々定義されるものと考える．これらのうち，状態不変条件と事前，事後条件は，形式検証の中で記述する場合, 数学の定理や論理式の学習が必要であり, 検証経験が足りない人には記述が困難という問題がある．

本研究では，システムの仕様が満たすべき大きな性質の一つとして，状態の一貫性を考える．状態の一貫性とは，操作の仕様である事前，事後条件が状態不変条件を保持することを示す．状態の一貫性という性質は，以下のような様々なシステムが満たす性質であり，その検証はシステムの検証として重要であると考えられる．

- 銀行の口座管理システム
  口座と関係する顧客情報の削除はできない
- 自動販売機のシステム
  商品に対する設定価格が一定の数値を超えない
- ホテルの予約管理システム
  部屋を予約する際に宿泊時間や予約部屋が重複しない

本研究では，定理証明の証明，状態と操作の仕様記述に要するコストを低減する系統的方法の一つとして，定理証明を利用した状態の一貫性検証支援を目的とする．

本研究のアイデアとして，システムの状態と操作の仕様記述の簡便化を図る．その方法として，まず，システムの持つ基本データ構造と，その基本データ構造が持つ性質を各々定義する．そして，予め定義したデータ構造と性質を持つシステムに対する仕様を記述するテンプレートを作成する．作成したテンプレートに対し，システムの環境としてデータの型や関連などを適用することで，そのシステムの仕様記述が作成できると考えられる．さらに，状態の一貫性を示す検証式を仕様記述から構成し，定理証明による証明を与えることで，そのシステムの状態の一貫性を検証することもできると考えられる．

本研究ではテンプレートを構成する環境として，定理証明支援系ツールの一つである Coq[4] の証明環境を利用する．Coq は定理証明支援系として最も利用されるツールの一つであり，標準で用意されている基本的な定理を中核にユーザが柔軟に証明を記述できる．

本研究では, 作成したテンプレートを評価するにあたり, サービス間の通信の際に複数のデータ構造を扱う SOA のシステムに着目した．そこで, SOA の共通アーキテクチャに関する研究 [7] の中で構成される SOA アプリケーションプラットフォームを事例の対象とした．基本データ構造を利用するシステムに対し，対応する構造と性質から作成したテンプレートを適用することで，システムの仕様記述と状態の一貫性検証が行えること，同じ構造を持つ複数のシステムへ適用できることの 2 点を確認する．

2　システムの状態

システムの状態は，任意のシステムの状態を表す要素 (以下，変数とする) の集合，初期条件，状態不変条件，状態に対する一連の操作などからなる．本研究では，図 1 に示すように，システムの取りうるすべての変数の集合を示すシステムの状態空間のうち，状態不変条件を満たす部分を扱う．本研究で扱う状態の一貫性は，操作によって，状態不変条件を満たす変数の部分集合を示す状態空間に含まれない状態 (状態不変条件を満たさない状態) へ遷移しないことを示すものと考えている．

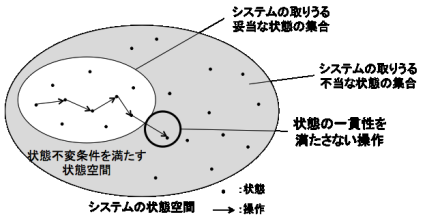

図 1　状態不変条件を満たす状態空間

3　状態の一貫性検証のテンプレート

3.1　テンプレートの構成

本研究では，状態の一貫性を検証するテンプレートは図 2 のように，本研究で考えたシステムの仕様に即して構成する．形式仕様記述言語である Z[5] の仕様を構造化して記述する記法と，VDM[6] の状態不変条件，事前条件，事後条件を構文として記述する記法をもとにテンプレートは作られるので，十分に仕様を記述できると考えられる．

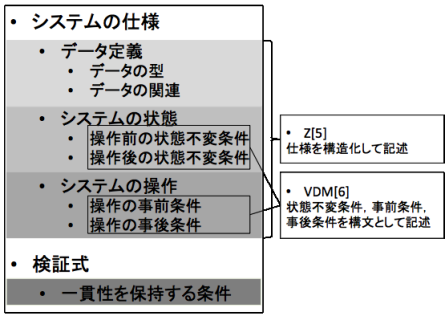

図 2　検証のテンプレートの構成

3.2　検証式

システムの状態の一貫性を示す検証式は，3.1 節で定義した状態と操作の仕様から，以下に示すように 4 つの条件から構成されると考えられる．このことから，検証式は一つの命題として考えることができ，証明を与えることで状態の一貫性を示す定理として利用できると考えられる．

操作前の状態不変条件∧操作の事前条件∧操作の事後条件
− > 操作後の状態不変条件

4　基本データ構造と性質の定義

4.1　基本データ構造

本研究では，システムが扱う基本データ構造として，以下の 4 つを考える．

- Queue
  データを先入れ先出しのリスト構造で保持するデータ構造．主にプリンタの出力処理やプロセス管理など，入力順にデータを処理するシステムで利用される．
- 2 項関係
  2 つのデータに関係を持たせて管理するデータ構造．主に一方のデータにアクセスすることで関係するもう一方のデータを扱う辞書的なシステムで利用される．
- Tree
  ノード間の関係を利用してデータを管理するデータ構造．主に階層構造を持つデータの管理システムや検索システムに利用される．
- Set
  データを順序なく管理するデータ構造．

4.2　基本データ構造の性質

本研究では，4.1 節で定義した基本データ構造の性質を図 3 のように抽出し，以下のように定義した．

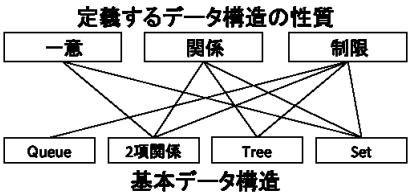

図 3　基本データ構造とその性質

- 一意
  各データに重複が無いこと
- 関係
  − 写像
  1 つのデータが他のデータと対応
  − 1 対 1 対応
  データが互いに 1 つに対応
- 制限
  − 上限
  データの容量がメモリ容量を上回らないこと
  − 下限
  データの容量が 0 を下回らないこと

5　テンプレートにおける仕様記述

3 章で示したテンプレートの構成に対し 4 章で定義したようなデータ構造の性質を適用することで，その構造と性質を持つシステムの仕様記述は作成される．その例として，2 項関係のデータ構造を持つシステムに対し一意の性質で仕様を記述したテンプレートの仕様記述の一部を図 4 に示す．図 4 における四角で囲まれた箇所がシステムの環境に依存する記述箇所であり，四角の箇所に対しデータの型を当てはめることでシステムの仕様記述を作成できる．

```
Definition Invariant (s:SetofData) :=
 forall x y:Data,
  In Data s x ∧ In Data s y ->
   fst x <> fst y.
Definition PreInvariant :=
 forall s:SetofData, Invariant s.
Definition PostInvariant :=
 forall s':SetofData, Invarinat s'.
Definition PreCondition :=
 forall (s:SetofData) (x:Data) (key:Key),
   In Data s x -> fst x <> key.
Definition PostCondition :=
 forall (s s':SetofData) (key:Key) (value:Value),
  s' = Add Data s (key,value).
```

図 4　テンプレート：仕様記述部分

6　事例検証

5 章で挙げたデータ構造とその構造の持つ性質から作成したテンプレートを評価する目的で事例検証を行う．本研究では，サービス間の通信の際に複数のデータを扱う SOA のシステムに着目した．事例として，SOA の共通アーキテクチャに関する研究 [7] の中で構築された SOA アプリケーションプラットフォーム (以下，プラットフォームとする) を対象とした．

6.1　SOA アプリケーションプラットフォーム

SOA の共通アーキテクチャに関する研究 [7] の中で構築された，コンポーネント間のサービス通信をサポートするプラットフォーム．その主な構成は図 5 に示す通りである．プラットフォームはサービス間で通信のイベントが発生した際に，送信先のサービスの状態に依存しないステートレスな通信を実現する．その実現のために，通信のイベントや送信先のサービス名，送信先のサービスの状態や場所などをデータベースで関連付けて管理する必要がある．そのデータ管理に，複数のシステムが特定のデータ構造を参照する．

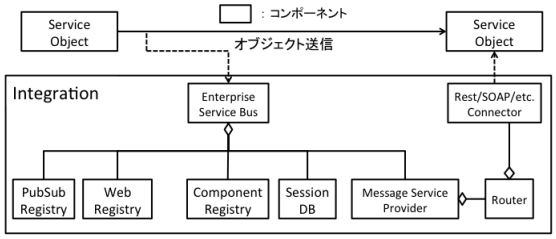

図 5　SOA アプリケーションプラットフォーム [7]

6.2　テンプレートを適用するシステム

プラットフォームの中のシステムで，本研究で定義したデータ構造を参照して構成されているものに対し，テンプレートを適用して検証を行う．適用するシステムは以下の 3 つであり，それらが管理するデータとデータ構造との対応関係をまとめたものを以下の表 1 に示す．

- MessageQueue
- WebRegistry
- SessionDB

表 1　各システムのデータ構造の対応

Queue		Value	
MessageQueue		Packet	
2 項関係	Key	Value	
WebRegistry	Name	Url	
SessionEntrySet	Name	SessionID	

6.3　テンプレートの適用

事例に対する適用例として，6.2 節で対象としたシステムの中の WebRegistry に対して，2 項関係のデータ構造と一意の性質から作成した 5 章のテンプレートを適用する．WebRegistry で扱われるデータは表 1 の通りである．また，WebRegistry のデータ構造は図 6 に示す通りである．

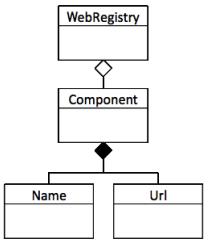

図 6　WebRegistry のデータ構造

以上に示すデータ構造から，各データを 5 章で示したテンプレートへの適用方法に則って当てはめる．それぞれのデータとテンプレートの対応と適用した結果として作成される仕様記述を以下に示す．

- SetofData <=> WebRegistry
- Data <=> Component
- Key <=> Name
- Value <=> Url

```
(* データ構造 *)
Definition Name := Set.
Definition Url := Set.
Definition Component := Name * Url.
Definition WebRegistry := Ensemble Component.


(* システムの状態の仕様記述 *)
(* 状態不変条件 *)
Definition Invariant (s:WebRegistry):=
 forall x y:Component,
  In Component s x /\ In Component s y -> fst x <> fst y.

(* 操作前の状態不変条件 *)
Definition PreInvariant :=
 forall s:WebRegistry, Invariant s.

(* 操作後の状態不変条件 *)
Definition PostInvariant :=
 forall s':WebRegistry, Invariant s'.


(* システムの操作（データの追加）の仕様記述 *)
(* 事前条件 *)
Definition PreCondition :=
 forall (s:WebRegistry) (x:Component) (key:Name),
    In Component s x -> fst x ≠ key.

(* 事後条件 *)
Definition PostCondition :=
     forall (s s':WebRegistry) (key:Name) (value:Url),
       s' = Add Component s (key,value).
```

以上に示すように，本研究で作成したテンプレートに対し，データの型などのシステムの環境を当てはめることで，そのシステムの仕様記述を作成できる．また，各仕様記述から状態の一貫性を保持することを示す検証式を構成している．検証式には証明を与えていることから，仕様記述の作成と同時にその検証も行うことができる．

7　評価

事例検証の結果として，Queue のデータ構造と制限の性質から構成したテンプレートは MessageQueue に，2 項関係のデータ構造と一意の性質から構成したテンプレートは WebRegistry，SessionDB にそれぞれ適用可能であることが確認できた．この結果から，本研究で提案したテンプレートは同じデータ構造を持つ複数のシステムに対する仕様を記述できることが確認できた．また，テンプレートでは仕様記述から構成した状態の一貫性の検証式に対し証明を与えているので，適用した複数のシステムの仕様の検証も可能である．以上のことから，本研究で作成した状態の一貫性検証のテンプレートは，共通する構造と性質を持つシステムの仕様記述の作成，および証明に要するコストの低減に有効であると考えられる．

8　考察

8.1　他のテンプレートの作成

事例検証では SessionDB のデータ構造の一部に対しその構造と一意の性質から構成したテンプレートを適用し検証を行った．しかし，SessionDB には他に 1 対 1 対応の関係を持つ構造部分も存在する．その箇所については，本研究で定義した関係の性質の一つである 1 対 1 対応の性質を利用して構成するテンプレートを作成，適用することで仕様記述を作成，検証できると考えられる．

8.2　作成できるテンプレート

本研究で提案したテンプレートは基本データ構造で扱うデータと各データ構造の性質から構成される．つまり，テンプレートはデータ構造と性質の組み合わせの数だけ作成可能であると考えられる．本研究で定義した基本データ構造とその性質からは，その関係を示す図 3 の線の本数から，9 個のテンプレートが作成可能と考えられる．

8.3　他のデータ構造の検証

本研究では，基本データ構造として，Queue と 2 項関係と Tree，および Set を定義した．しかし，他の基本データ構造として，Stack なども存在する．Stack は Queue と同様に一時的にデータを保持し，先入れ後出しの要領でデータを管理するデータ構造なので，制限の性質を持つと考えられる．よって，制限の性質を利用して作成したテンプレートを適用することで検証できると考えられる．

8.4　形式的仕様記述言語との連携

本研究では，定理証明支援系ツールの証明環境を利用して仕様記述および検証のテンプレートを作成した．しかし，ツールの証明環境の中には記述が困難な定理や実装されていない定理が存在する．そこで，関連研究 [8] のように，形式仕様記述言語を利用することで，適切な証明環境における仕様記述と証明が可能になり，より正確な仕様記述を作成できると考えられる．

9　関連技術：Z

検証対象のシステムを適切に抽象化して表現できる，形式仕様記述言語の一つ．特徴として，各仕様を構造化して分けて定義できる「スキーマ」というものを利用する．スキーマを利用することで状態の仕様や操作の仕様を個々にまとめて定義することができる．また，データ型の定義は集合と関数に基づいて表現される．以上の特徴から，Section という記述で仕様記述を区切ることができ，同様のデータ型の定義を持つ Coq とは 8.4 節で考察した連携における相性が良いと考えられる．

10　おわりに

本研究は，ソフトウェアの品質向上の技術である形式検証における，定理証明の証明と仕様記述に要するコスト低減を目的として，定理証明を利用した状態の一貫性検証を支援した．支援方法として，システムのデータ構造とその性質から構成される状態の一貫性検証のテンプレートを作成した．共通のデータ構造と性質を持つシステムの仕様記述，および状態の一貫性の検証に対するコストの低減が可能であることを SOA の事例 [7] に適用することで確認した．今後の課題として，本研究で提示した Queue や 2 項関係などの基本データ構造だけでなく，他のデータ構造やその性質の定義，および検証を行うことが挙げられる，

参考文献

[1] A.Hall, “ Seven Myths of Formal Method, ” IEEE Software, Vol.7, No.5, pp.11-19, 1990.

[2] D.Graigen, S.Gerhart, T.Ralston, “ An International Survey of Industrial Applications of Formal Methods, ”National Technical Information Service, Vol.1, 1993.

[3] D.Graigen, S.Gerhart, T.Ralston, “ An International Survey of Industrial Applications of Formal Methods, ” National Technical Information Service, Vol.2 1993.

[4] The Coq Community, “ The Coq Proof Assistant, ” http://coq.inria.fr.

[5] Mike Spivey, “ The Z Notation: a reference manual, ” http://spivey.oriel.ox.ac.uk/mike/zrm/index.html.

[6] SCSK 株式会社, “ VDM information web site, ” http://www.vdmtools.jp/.

[7] 江坂 篤侍，野呂 昌満，沢田 篤史, “ SOA アプリケーションプラットフォームのプロダクトライン化に関する研究, ” 情報処理学会研究報告，ソフトウェア工学研究会報告，Vol.2014，No.13，pp.1-8，2014.

[8] 来間 啓伸，Burkhart Wolff，David Basin，中島 震, “ 仕様記述言語 Z と証明環境 Isabelle/HOL-Z, ” コンピュータソフトウェア, Vol.24, No.2, pp.21-26, 2007.