情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-SE-165 No.8
2009/7/3

データ管理領域における プログラム検証の自動化に関する考察

矢　竹　健　朗[†1]

データ管理領域とは，図書館システム，在庫管理システム，座席予約システムなど，大量の受動的データを管理するようなシステムが属す問題領域である．本稿では，この問題領域の特性を生かしてプログラム検証を自動化する手法について考察する．本稿では特に，プログラム検証の自動化の大きな障壁として知られる不変表明の取り扱いに着目する．

Towards automating program verification in the data management domain

Kenro Yadake[†1]

The data management domain is one of the problem domains of software systems. It consists of the systems which manage a large amount of passive data such as library systems, warehouse management systems and ticket reservation systems. This paper presents an approach to automate program verification making use of the common characteristics in this domain. We especially focus on the technique to handle invariants which is known as the major obstacle against automation of program verification.

1. はじめに

データ管理領域とは，様々な情報システムの中でも，大量のデータを管理することを目的とするシステムが属す領域である．例えば，銀行システム，証券取引システムなどである．銀行システムでは大量の顧客データや口座データが管理対象となる．証券取引システムでも同様に，株式や銘柄，売買注文といった様々なデータが管理対象となる．他にも，航空券発見システムや，通販システム，在庫管理システムなど，数多くのシステムがこの領域に属す．データ管理系のシステムは社会インフラを構成しており，高信頼性が要求される．例えば、誤って口座を２重に登録してしまい前の残高データを消去してしまったり，大量に発行された株式売買注文の一部が処理可能容量を越えて消滅してしまったりといったことはあってはならない．このような欠陥は，金銭面，信用面において非常に大きな損失をもたらすこととなる．高信頼な社会インフラを実現するためには，これらのシステムの動作の正しさを保証することが必要不可欠である．

情報システムは最終的には何らかのプログラミング言語で実装される．システムの正しさをプログラムレベルで保証する手法として，プログラム検証法がある．プログラム検証とは，プログラムの動作が論理的に正しいことを定理として証明する技法であり，その基礎は Hoare 論理[1)] として 1960 年代に確立されている．プログラム検証は、システム動作の正しさを網羅的に検証できる非常に強力な手法であるが，ユーザ介入が必要であり検証コストが大きい．そのため，システム開発の迅速性が求められる産業界ではほとんど適用されていない．

そこで本研究では，問題領域をデータ管理領域に限定し，その特性を生かした自動的なプログラム検証法の構築を目指す．本研究では特に，不変表明の扱いに焦点を当てる．オブジェクト指向言語の場合，不変表明は 2 種類あるが，いずれも証明の前提条件としてユーザが与えなければならない条件であり，プログラム検証を自動化する上でネックとなっている存在である．1 つは，while 文のループ不変表明である．これは，while 文の繰り返しの過程で常に成立している条件であり，while 文の推論規則の前提として必要となる条件である．もう 1 つは，クラス不変表明である．これは，オブジェクトに関する不変表明であり，オブジェクトのライフサイクル，つまり，コンストラクタにより初期化されメソッドにより状態変化するというサイクルにおいて常に成立している性質である．不変表明を人間が理解しやすい形で自動的に求めることは難しく，通常，その発見はユーザに任せられる．

本研究では，データ管理領域において頻繁に使用される繰り返しパターンや，データ構造の特徴に着目し，不変表明を与えることなく証明可能なプログラミング言語を開発することを目指す．本稿では例題を用いてそのアイディアを説明する．

本稿の構成は次の通りである．2 節では，ループ不変表明に対処する手法として，オブジェクト列操作命令とその推論規則の導入について述べる．3 節では，クラス不変表明に対

†1 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

処する手法として，クラス不変表明を言語に織り込む手法について述べる．4 節では，今後の展望，5 節では関連研究を述べる．

2. オブジェクト列操作命令と推論規則の導入

2.1 アプローチ

while 文のループ不変表明を求めることは，プログラム検証において最も難しい作業である．実際，最大公約数を求めるプログラムや平方 2 乗根を求めるプログラムといった初歩的なプログラムでも，その発見には時間を要する．しかし，このような数学的アルゴリズムに基づくプログラムと比較して，データ管理領域における繰り返し構造は単純なものが多い．大部分は，オブジェクト列操作に関するものである．例えば,「口座列に対し順番に利息分だけ残高を加算する」といった操作や,「買い物カゴの中の各商品の値段を順番に加算し購入代金を計算する」といった操作である．このようなオブジェクト列に対する単純操作が多いのは，データ管理系のシステムは本来，大量のデータに対する定型処理を自動化することを目的に作られており，複雑なアルゴリズムにより難解な問題を解くことが目的ではないためである．

我々はこの点に着目して，データ管理領域における推論の自動化を目指す．具体的には，オブジェクト列に対する様々な繰り返しパターンを直接記述可能な命令群と，それらに対するループ不変表明によらない推論規則を定義する．これにより,「利息計算後，それぞれの口座の残高は利率分だけ増加している」や,「新しい商品が買い物カゴに追加されたら，購入代金はその商品の値段だけ増加する」といった推論をアトミックに行えるようにする．

2.2 例　　題

実際に例を用いて説明する．図 1 は現在開発中のデータ管理言語によって記述された，接続のタイムアウトの論理を実現するプログラムである．クラス `connection` は接続を表すクラスである．クラス `contable` は接続表を表すクラスであり，接続をリスト `connectionList` によって管理する．メソッド `decTimerAll()` は，`connectionList` に含まれるすべての接続のタイムアウト残り時間を 1 減算し，タイムアウトした（残り時間が 0 になった）接続を削除するメソッドである．この操作はオブジェクト列操作命令 `apply` と `select` を用いて実現している．命令 `apply` は，`L->apply(M())` のように使用し，オブジェクト列 `L` の各要素に対してメソッド `M()` を適用する．命令 `select` は，`L->select(P())` のように使用し，`L` の中からメソッド `P()` を満たす要素だけを取り出す．このメソッドについて契約（事前事後条件）「接続表の中に残り時間が 1 より大きい接続が存在するならば，実行後も接続表に存在する」を定義している．命令 `contains` は，`L->contains(x)` のように使用し，オブジェクト `x` が `L` の要素であるかどうかをチェックする．

```
class connection {
  private timer : num;
  public decTimer():void { timer = timer - 1; }
  public isAlive():bool { return timer > 0; }
}

class contable {
  private connectionList : connection list;
  public decTimerAll():void {
    connectionList->apply(decTimer());
    connectionList = connectionList->select(isAlive());
  } contracting {
    Forall x.
    pre : connectionList->contains(x) && (x.timer>1) && (x!=null:connection)
    post : connectionList->contains(x)
  }
}
```

図 1　サンプルプログラム

命令 `apply`, `select` に関して図 2 のような推論規則，公理を定義することができる．`apply` の推論規則は,「オブジェクト列 L の要素に対して順にメソッド $G()$ を実行した直後に要素 x がメソッド $Q()$ を満たすかどうかの真偽は，x に対し単独に $G()$ を適用した直後に $Q()$ を満たすかどうかの真偽に等しい」を意味する．例えば，接続表に含まれる接続すべてについてタイムアウト残り時間を 1 減算した直後，ある接続がタイムアウトしたかどうかの真偽は，その接続に対し単独にタイムアウト時間を減算し，タイムアウトしたかどうかの真偽と等しい．具体的に，結論部は，事前条件 P のもとで $L \rightarrow app(G())$ を実行後，オブジェクト x に関する表明 $Q(x)$ が成り立つことを意味する．2 つの前提条件は，それぞれ，x が L に含まれる場合，含まれない場合の条件である．前者は，x に対するメソッド $G()$ の適用後に結論 $Q(x)$ が成り立てばよい．後者は，P から直接 $Q(x)$ が導出できなければならない．ただし，この規則は，L に重複する要素が存在しないこと，また，$G(), Q()$ が内部で他オブジェクトのメソッドを呼ばないことを条件とする．`select` の公理は,「オブジェクト

情報処理学会研究報告
IPSJ SIG Technical Report

$$\frac{\{L \to contains(x) \wedge P\}\ x.F()\ \{Q(x)\} \qquad \neg(L \to contains(x)) \wedge P \Rightarrow Q(x)}{\{P\}\ L \to apply(F())\ \{Q(x)\}}$$

$$\{L \to contains(x) \wedge G(x)\}\ L' = L \to select(G())\ \{L' \to contains(x)\}$$

図 2 apply の推論規則と select の公理

x が列 L の要素であり，かつ，メソッド $G()$ を満たすならば，x は L から $G()$ を満たすものだけを取り出した列 L' においてもその要素である」を意味する．

このように，オブジェクト列操作命令 `apply` と `select` について，ループ不変表明によらない推論規則，公理を定義することができる．while 文はあらゆる繰り返し処理を表現可能な最も原始的で汎用的な命令文であるが，その分，推論にはループ不変表明という付加的な条件が必要となる．しかし，繰り返しパターンごとに `apply` や `select` のような専門的な命令を導入すれば，それぞれの命令の表現力が限定される分，論理法則が明確になり，ループ不変表明のない推論規則を定義することが可能となる．実際にこれらの規則を用いてメソッド `decTimerAll()` の契約を証明することが可能である．具体的には，x,L,L',$F()$,$G()$,P,$Q(x)$ の各式をそれぞれ，`x`, `connectionList`, `connectionList`, `decTimer()`, `isAlive()`, `1<x.timer`, `x.isAlive()` に対応付ければよい．推論規則，公理の健全性に関しては定理証明器 HOL[2] で証明可能である[6]．

今後，領域分析を行い，様々な繰り返しパターンに対応する命令と推論規則を導入し，データ管理言語の表現力を高めていく予定である．

3. クラス不変表明の言語への織り込み

3.1 アプローチ

クラス不変表明とは，あるクラスからインスタンス化するすべてのオブジェクトについて常に成立している性質である．クラス不変表明の中には，あらゆるシステムに共通して出現する典型的なものがいくつか存在する．例えば，リストの単一性やオブジェクトのキー制約などである．典型的な不変表明を，検証ごとに記述，証明することはユーザにとって大きな負担である．特に，不変表明は論理式による記述が必要であり，記述段階で誤りが混入しやすい．また，不変表明の証明には帰納法が必要であり，検証コストが大きい．

そこで我々は，典型的な不変表明を予め言語に織り込んでおくことが有効であると考える．つまり，言語側で不変表明を成立させるメカニズムを実装し，ユーザが直接不変表明を記述，証明する負担を軽減する．

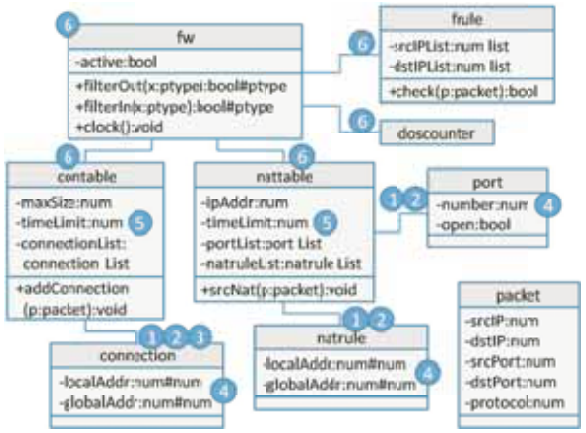

図 3 ファイアウォールシステムにおけるクラス不変表明

3.2 クラス不変表明の分類

不変表明の織り込み方はその種類に依存する．そこでまず，ファイアウォールシステム[7]の例に基づき，不変表明の分類を行った．以下に，データ管理言語に付随する表明言語（メソッド契約やクラス不変表明を記述するための言語）による記述とともに示す．また，図 3 にクラス図における対応箇所を示す．

（1）非 NULL リスト：
リストの要素に NULL オブジェクトが存在しない．例えば，`contable` クラスのフィールド `connectionList` の要素には NULL オブジェクトが存在しない．

```
Forall c. contable.connectionList->contains(c)
  implies c!=(null:connection)
```

（2）単一リスト：
リストに同一のオブジェクトが重複して含まれない．例えば，`connectionList` は単一リストである．

```
Forall c. contable.connectionList->contains(c) implies
  contable.connectionList->select(isEqualTo(c))->length()==1)
```

（3）リスト長限界：
リスト長に上限がある．例えば，`connectionList` の長さはフィールド `maxSize` の値

を超えない．

```
contable.connectionList->length()<=contable.maxSize
```

(4) キー制約：

オブジェクトリストの要素について，あるフィールドの値がその要素を特定するためのキーとなっている．例えば，リスト portList に含まれる port オブジェクトは，フィールド number により一意に識別可能である．

```
Forall p1 p2. nattable.portList->contains(p1) &&
  nattable.portList->contains(p2) &&
    p1!=p2 implies p1.number!=p2.number
```

(5) 定数オブジェクト：

システムの初期化時から常駐し続け，決して NULL にならないオブジェクト．例えば，fw オブジェクトにリンクする contable オブジェクト，nattable オブジェクト，frule オブジェクト，doscounter オブジェクトはシステム初期化時に生成され，その後ずっと存在し続ける．

```
contable!=(null:contable) && nattable!=(null:nattable) &&
frule!=(null:frule) && doscounter!=(null:doscounter)
```

(6) フィールド限界：

フィールドのとりうる値に範囲がある．例えば，cotable クラスのフィールド timer の値は，nattable クラスのフィールド timer の値を超えない．

```
contable.timer <= natable.timer
```

3.3 織り込みの例

例えば，不変表明 (1) を織り込むには次のようにすればよい．まず，構文レベルでは，ユーザがラベル non-null によって不変表明を宣言できるようにする．

```
private non-null connectionList : connnection List = Nil;
```

このラベルにより，フィールド connectionList は非 NULL リストであることが宣言される．一方，言語側では，このラベルが付与された場合，リスト connectionList に対するプリミティブ命令をリストの非 NULL 性を満たすように意味付けする．具体的には，connectionList に対する要素追加命令 add （L->add(x) のように使用する）を,「引数のオブジェクトが NULL でないときに限りリストに追加する」と意味付けする．これにより，ユーザが connectionList に対してどのようにアクセスしたとしてもリストへの NULL オブジェクトの混入を防ぐことができ，不変表明を保証できるようになる．

このような織り込み手法により，ユーザはラベルを付加するだけで不変表明の成立を保証でき，従来のように不変表明をプログラムにアノテーションとして記述したり，帰納法により証明したりする必要がなくなる．

不変表明の織り込みは，これ以外にも，言語の公理系を強化できるという利点がある．現在，言語の意味論はオブジェクト指向理論（定理証明器 HOL 上に実装したオブジェクト指向の公理系[7]）により公理的意味論として定義している．不変表明を言語に織り込むことにより，公理系にその不変表明を新たな公理として追加することができる．そして，追加された公理と既存の公理からさらに新しい定理を導出することができる．結果として，公理系に含まれる定理数を増やすことができ，公理系を強化することができる．

例えば，(1) に示した不変表明は，次の公理として公理系に導入することができる．

```
|- !(fw:fw) (c:connection) (s:store).
     let ct = fw_get_contable fw s in
     let l = contable_get_connectionList ct s in
       MEM c l ==> connection_ex c s
```

この公理は,「connectionList の要素は NULL でない」を意味する．

また，公理系には次の公理が存在する．

```
|- !c x s. connection_ex c s ==>
     (connection_get_timer c (connection_set_timer c x s) = x)
```

この公理は,「NULL でない connection オブジェクトについて，フィールド timer を x に更新し，直後に参照すると x が得られる」を意味する．

これら 2 つの公理から次の定理を導出することができる．

```
|- !fw x s.
     let ct = fw_get_contable fw s in
     let l = contable_get_connectionList ct s in
     let c = HD l in
       0 < LENGTH l ==>
       (connection_get_timer c (connection_set_timer c x s) = x)
```

この定理は,「connectionList の長さが 0 でないとき，その先頭要素について，フィールド timer を x に更新し，直後に参照すると x が得られる」を意味する．

この定理の導出には数ステップの証明を要する．つまり，実際の検証において，その分だけ証明ステップ数を削減できることになる．勿論，公理の導入は，公理系の健全性を維持す

るために保守的な方法で行う必要がある．つまり，公理を直接導入するのではなく，add 命令の定義から導出する必要がある．

今後，他の種類の不変表明についても言語に織り込む手法を検討していく予定である．

4. 今後の課題

今後はまず，本稿で述べた技術を導入したデータ管理言語を開発する．意味論は定理証明器 HOL に実装したオブジェクト指向理論上に定義し，ホーア論理の公理や健全性を証明する．また，検証システムとして，データ管理言語に対する検証条件生成器，及び，検証条件自動証明器を実装する．

次に，検証システムを実例に適用し，検証能力の評価を行う．評価点としては，データ管理言語の記述能力でどの程度のシステムが記述可能であるか，また，検証システムによりどの程度検証が自動化できるかの２点を考える．実例としては，まずこれまで扱ってきたファイアウォールシステムを対象とする．このシステムについては ObjectLogic[7] を使って検証を行ったが，リスト操作に関する推論に大きなコストがかかっていた．今回，新たに実装する検証システムにより，どの程度検証効率が改善されるのかを評価する．また，これ以外にも様々な事例について検証実験を行い，検証システムの記述能力，検証能力を強化していく．

さらに，検証システムの付加機能として，データ管理言語から Java コードを自動生成する機能を実装する．データ管理言語は実際のプログラミング言語よりも一段階抽象レベルにおいてシステムの動作論理の記述，検証を行うための言語である．これを Java コードに変換する機能を付加することにより，実際の開発に導入しやすい言語とする．Java コードへ変換する際には，データ管理言語におけるリストを，コレクションやデータベースなど，システムの規模に応じて変換先を切り替えられるようにする．また，変換の正当性を保証する手法についても検討を行う．

5. 関 連 研 究

プログラム検証の自動化を目指した研究は数多い．その多くが，while 文のループ不変表明を自動生成することを目指したものであり，様々な観点からの自動化手法が提案されている．例えば，プログラムの実行時の変数値を観測し不変表明を推測しようというもの[3]，while 文の中身が多項式の代入文と条件文のみで構成されているプログラムを対象として自動生成を行うもの[4]，証明の失敗ケースを手掛かりに生成される不変表明の推測精度を高めようとするもの[5] などである．本研究のアプローチは，問題領域に特化した繰り返し命令を導入し，最初からループ不変表明によらない推論規則を定義している点が特徴である．これにより，不変表明の自動生成という冗長なステップを踏むことなく，アトミックな推論を可能とする．また，本来 while 文や for 文で記述される繰り返し構造を，領域専門の高級な命令を用いて 1 行で記述することが可能である．これによりユーザの記述量を減らすとともに直観的な記述を可能としている．

6. お わ り に

本稿では，データ管理領域においてプログラム検証の自動化率を向上させる手法を 2 つ説明した．いずれもプログラム検証のネックとなる不変表明に扱いに着目した手法である．1 つは，オブジェクト列に対する繰り返しパターンを直接記述するための命令とループ不変表明によらない推論規則を定義する手法である．もう 1 つは，典型的なクラス不変表明を言語に織り込む手法である．これらの手法によりプログラム検証におけるユーザ介入性が大幅に削減されることが期待できる．今後は，データ管理言語，検証システムの実装，実験，評価と順次行っていく予定である．

参 考 文 献

1) Hoare, C.A.R. An Axiomatic Basis for Computer Programming. Communications of ACM, 12. 1969.
2) The HOL system. URL: http://hol.sourceforge.net/.
3) Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. The Daikon system for dynamic detection of likely invariants by Science of Computer Programming, vol. 69, no. 1–3, Dec. 2007, pp.35-45.
4) E.Rodriguez-Carbonell and D. Kapur. Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations. In International Symposium on Symbolic and Algebraic Computation 2004 (ISSAC'04), July 2004.
5) A. Ireland and J. Stark. On the Automatic Discovery of Loop Invariants. In Proceedings of the Fourth NASA Langley Formal Methods Workshop, NASA Conference Publication 3356, 1997.
6) 矢竹健朗，片山卓也：オブジェクト列に対するループ文の推論規則，第 5 回ディペンダブルソフトウェアワークショップ DSW2007, 予稿集, pp.57-63.
7) 矢竹健朗, 片山卓也：事例研究：定理証明によるファイアウォールサーバモデルの検証, コンピュータソフトウェア, Vol.26, No.1(2009), pp.111-126.