関数型プログラミング言語向けの形式検証支援及び開発支援システムの提案

才村 徹也　　岡野 浩三　　谷口 健一
大阪大学大学院情報科学研究科
TEL: (06)6850-6606
{saimura, okano, taniguchi} @ist.osaka-u.ac.jp

Abstract

関数型プログラミング言語を対象とした形式的検証支援及び開発支援システムを提案する．モジュールのインターフェイスに対して前提条件，後置条件を等式論理の形で記述し検証を行う．上述の性質記述はソースファルのコメント中に記述する．性質記述とソースプログラムから，前提条件の下で後置条件が成り立つことを表す論理式を作成し，プレスブルガー文真偽判定アルゴリズムなどを用いて検証する．さらに図書管理システムの性質記述例について述べる．

1 はじめに

信頼性の高いソフトウェアの設計，開発において形式的検証は有用である．形式的検証における課題として，モデル検査における状態爆発問題や，定理証明系の自動化の困難さが挙げられる．これを克服するために大きく複雑なソフトウェアを，複数のモジュールに分割し検証を行う手法 [1, 2] や，既存の形式手法を複合して欠点を補い，効率的に検証を行う手法 [3, 4] などが考案されている．研究グループでは，プログラムの代数的手法に基づいた仕様記述方法，段階的設計法，検証支援方法や，ハードウェア設計自動化に関する研究を行ってきた．プレスブルガー文真偽判定ルーチンの効率的検証への利用 [5] などに特色がある．

本稿では，モジュールで構成された実用プログラムに対して， 効率的な検証支援を行う枠組みとして，関数型言語 ML を対象にした形式的検証システムを含む開発支援システムの提案を行う．

提案する検証システムでは，プログラムのモジュールごとに，モジュールの満たすべき性質 (入力に対する前提条件，前提条件のもとで入力と出力に対して成り立つ後置条件) を等式論理の形で，モジュールのインターフェイスファイル中のコメントに XML 風のタグと共に記述し，それを用いて検証を行う．記述には，入力の引数変数，関数，述語を用いる．述語は基本関数の性質を簡潔に表すために用いる．述語の定義についても，性質と同様にコメントとしてソースプログラム中に記述する．モジュールの正しさとして，モジュール内の公開関数ごとに「前提条件のもとで後置条件が成り立つこと」(検証対象式) を検証する．

関数型プログラムは，等式の集合とみなせ，代数的仕様記述との親和性が高い．性質記述とプログラム中の関数定義は，項書換え規則として扱える．上述の検証対象式に対して，そこに出現する関数等を項書換えにより展開する．得られた論理式は，Knuth-Bendix の半アルゴリズムにより書換え規則を正規化しさらに書換えを行うか，プレスブルガー文真偽判定アルゴリズムを用いてその真偽を判定する．プレスブルガー文に論理式を変換する場合は，論理式中で展開されずに残った述語の出現等を論理変数または整数変数に変換する．それらの変数についてそれ以上の解釈を入れずに真偽判定を行う．

プレスブルガー文真偽判定アルゴリズムは，関数型言語 ML を用いて実装し，実験によりその有用性を確かめた．システムの適用例題として，図書管理プログラムを設計し，性質を記述した．実用性を確かめることに適していると考える．

本稿では 2. でプレスブルガー文真偽判定アルゴリズムと作成したルーチンの概要を述べ，次に 3. で現在構築中の検証支援システムについて説明する．最後に 4. で図書管理システムの概要とその性質記述について述べる．

2 プレスブルガー文真偽判定ルーチン

2.1 プレスブルガー算術

加算を持つ整数の理論 (整数の集合上の変数，定数，$+$，$-$，$=$，$\geq$，$\leq$，$\vee$，$\wedge$，$\neg$，$\forall$，$\exists$ からなる理論) はプレスブルガー (Presburger) 算術と呼ばれ，その上の閉論理式をプレスブルガー文あるいはP文と呼ぶ，P文の真偽は決定可能である [6].

2.2 実装と評価

実装は，SML/NJ を用いて行った．入力は一般のP文である．システムで用いることを考え，ブール変数の導入と if then else 式，含意，排他的論理和，論理比較等の演算子の追加を行った．実装に用いたP文の真偽判定法では，一般のP文に対しては Cooper のアルゴリズム [6] を用い，限定子がすべて存在子である冠頭標準形P文に対しては，高速化アルゴリズム [5] を適用する．

8ビット CPU KUE-CHIP2 を五段階 (レベル) で設計し，既に検証している [7]．レベル3からレベル4への詳細化の正しさを示す式について，今回作成したP文真偽判定ルーチンを実行した．レベル3では各命令の実行手順の決定，レベル4ではさらにプリフェッチを導入している．

トークン数 (変数，論理演算子，整数項などの総出現数) が 10〜200，変数は 1〜30 個，ブール変数 2〜20 個の式が 346 個あった．全ての式は，限定子が全て同一であり，真である．

作成したルーチンに適用した結果，全ての式の判定時間の総和は 1 秒以内であった (Pentium4 CPU 2.66GHz，メモリ 1GB)[8].

実際の検証において用いられる式を，満足のいく速度で判定できたことは，目的である検証支援システムでの有用性，関数型言語を用いたシステム作成の実用性を示す結果になったと考える．

3 形式的検証支援および開発システム

作成したP文真偽判定ルーチンを使用する形式的検証システムを現在構築中である [8]．この節では，まずシステムについて簡単に説明し，次にシステムにおいて重要な部分の設計方針について述べる．最後に，システムの将来的な課題等について述べる．

3.1 形式的検証システムの概要

このシステムの対象は，いくつかのモジュールからなる ML のプログラムである．検証者がプログラムのソースコードにコメントとしてモジュールの満たすべき性質を等式論理の形で記述し (性質記述)，それを用いてプログラムの正しさを検証する．具体的には，モジュールのインターフェイスに対して，入力の守るべき前提条件，出力の守るべき後置条件を等式論理で表し (契約による設計 [9])，それからモジュール内の公開関数ごとに「前提条件のもとで後置条件が守られること (前提条件 imply 後置条件)」という式を作成し，証明を行う (以後，この論理式を検証対象式とする)．

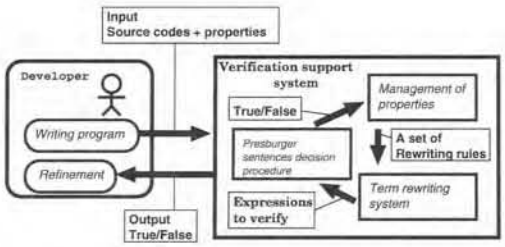

図 1. Formal verification flow with the proposed system

上述の検証対象式を直接証明することが難しい場合は，プログラム中で成り立つ性質 (不変表明) を併せてソースコード中に記述する．その場合，「前提条件が成り立つ上で，不変表明までの関数を評価した後に不変表明が成り立つこと」と「不変表明が成り立つ上で，最後まで関数を評価した後に後置条件が成り立つこと」に対応する検証対象式を作成し，証明を行う．

同様に，基本関数や補助述語の性質の定義もソースコード中に記述する．性質記述はこの基本関数や補助述語も用いて行う．

3.2 設計方針

対象としている言語は O'Caml[10](StandardML の派生言語) である．プログラムの実行速度のベンチマークでは，高い評価を得ている．

3.2.1 言語に対する制限

システムの簡単化のため，入力の O'Caml 言語に対して次のような制限を課す．(a) オブジェクト指向に関する機能を用いない，(b) 参照型は用いない，(c) 関数適用では，一階の式に限定する，(d) モジュール内でローカルなモジュール定義はない．

これは，副作用を生じるような手続き型言語的な機能の制限と簡単化のためのものであり，関数型言語としての能力は，さほど制限されるものではないと考える．

3.2.2 性質記述

O'Caml では，モジュールのインターフェイスとして，関数の入力と出力の型をファイルに記述する．このファイルのコメント中に，XML 風のタグを用いて，入力に対する前提条件，出力に対する後置条件を記述すれば，関数の入出力定義との対応が解りやすい．図 2 はソースコード中の性質記述の例である．

```
(*
整数とリストを引数に取り，整数番目の要素を取り出したリストを
返す関数pickupの型定義.
*)

val pickup : (int * 'a list ) -> 'a list
(*
<pre> (x, l) = ( x>0  and x < List.length(l) ) </pre>
<post>  (x, l, pickup) =( List.length(l) -1 == List.length(pickup(x,l)) ) </post>
*)
```

図 2. An example of description of properties

3.2.3 補題の扱い

性質記述で用いる補助述語の定義や，述語間の性質記述も同様にコメント中の記述として，提案したスタイルと同様に，モジュールファイルなどに記述する．このような補題は，それが出現する検証対象式において，項書換えによって，より具体的な論理式に展開される．

3.2.4 項書換え系の利用

実際に真偽判定を行う検証対象式への変換は，項書換え系を用いる．検証対象式を (1) プログラム中の関数定義，(2) 述語の性質記述を用いて項書換えを行い，関数等が展開された式を得る．さらに Knuth-Bendix の半アルゴリズムを利用し書換え規則の正規化を行うか，整数とブール変数のみを用いた P 文に変換することで，真偽判定を行う．判定ができなかったときは，検証者が性質記述を修正したり，補題を追加して再度検証を行う．

実際のプログラムに対する検証では，この枠組みの適用が難しい場合もある．以下では，そのような場合に対する設計方針を述べる．

名前空間　複数のモジュールで定義された同一の関数名が，異なった実現に対して用いられることがある．したがって，書換え規則となる関数名についてはモジュール名を補完し，名前により関数を一意に特定できるようにする．ML では，Let 式を用いて局所的定義できる．Let 式による束縛も書換え規則として用いる．Let 式を含む関数が書換え規則として用いられる度に，Let 式により定義された変数も，一意で指定できるように適切な名前をつけ書換え規則に加える．

型検査　O'Caml は型推論を行う．検証システムの項書換えにおいても，型付きの項書換えを行うべきである．現在は，O'Caml 自体の型推論による型付け機構を利用して，読み込んだ式の型付けをすることを考えている．

3.3 システムの適用範囲

このシステムの対象は，3.2.1 でも述べたように純粋な関数型プログラムである．動作主体のプログラムに対しては適用し難い．そのようなものに対してはプロセス代数など他のアプローチが有効であろう．しかし，検証が必要な，正しさが求められるモジュールでは副作用のない純関数型言語的な記述を行うことは適切であると考えられ，これらの制限をおいても，実用性はそれほど損なわれないとも考えている．

モジュールベースの検証を行うことから，大規模実用プログラムにおいても，部分的な検証において，適用がある程度可能であると考えている．

3.4 システム実装

今後は，まず 3.2. の方針に沿った検証支援システムを作成し，例題に適用することで有用性を評価する予定である．

またプログラムの修正支援として真偽判定の結果だけでなく検証者にとって有用なフィードバックを与えることを考えている．プログラムの性質の記述支援，性質記述を含むプログラムの情報のドキュメント生成等の機能なども検討している．

4 図書管理システムへの適用例

提案する検証システムの適用例として，次の図書管理システムを設計し，性質記述を行った．

4.1 概要

図書館における本の貸出，返却に伴い，本や利用者の情報に関するデータベースの更新等の図書管理を行うシステムを考える (図 3)．

図書館の本，利用者の情報はデータベースに登録される．管理システムはユーザの操作として，次のような機能を持つ．

- 本の貸出，返却
- 利用者の登録，削除
- 本の登録，削除
- データベースの更新 (返却期限を過ぎた人のリストアップ)

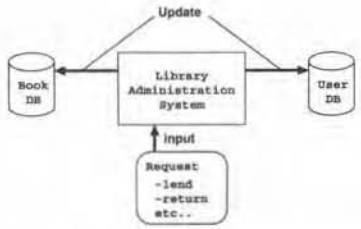

図 3. The overview of a library administration system

返却期限を過ぎた人に対しては，貸出を禁止する．返却期限を越えた場合，ペナルティリストに追加し，返却までに期限を超過した日数の間貸出禁止とする．ペナルティリスト中の人に対しては，一ヵ月毎に催促の通知を出す (システムが印刷指示を出す)．また，このシステムは開架式の図書館で用いられることを想定している．

ここで，加算ができるデータ (日付など) は整数として扱い，その他は文字列としてモデル化する．利用者データベースは，属性集合 {利用者コード，利用者名，住所，借本リスト，貸出禁止期間} を持つデータベースである借本リストは，{図書コード，返却期限} のエントリの集合である．本データベースは，属性集合 {図書コード，ISBN 番号，本のタイトル，本の状態，貸出者コード} を持つデータベースである．貸出者コードは，現在この本を借りている利用者の利用者コードである．本の状態として {貸出可，貸出中，貸出禁止} がある．

図書管理システムのモジュール構成を図 4 に示す．

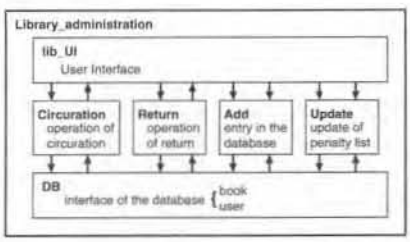

図 4. Modules of the library administration system

4.2 貸出モジュール

ここでは，貸出モジュール (Circuration) に注目し，その性質記述の例について述べる．

4.2.1 貸出処理

貸出処理の手順は次のようにする．入力として，利用者 ID と本 ID 集合が与えられるとする．

1. ペナルティリストと利用者 ID を照合，リストにあれば貸出不可．処理終了
2. 入力の本 ID について
   - 本 ID から本データベースに問い合わせ，本情報を得る
   - 本の状態が "貸出可" 以外の場合．貸出不可．処理終了
   - 本の状態を "貸出中" に変更
3. 利用者 ID から利用者データベースに問い合わせ，情報を得る
4. 利用者の貸出リストに，先程 "貸出中" にした本を追加する (本 ID，返却日を 1 エントリとする)
5. データベースを更新

O'Caml により記述した貸出モジュールの公開関数である貸出関数は図 5 のようになった．

```
module Circulation = struct
...
let lend_one_book uid bid = let book = DB.getBook bid in
    if book.state = "lendable" then (
        let user = DB.getUser uid in
        let date = limitDate() in
        let list = user.books in
        let list' = (bc,book.title,date)@list in
        let rst = DB.editUser uid {usercode=user.usercode;
                                   name=user.name;address=user.address;
                                   books=list';penalty=user.penalty} in
        if (fst rst) then ( let rst = DB.editBook bc {libcode=bc;isbn=book.isbn;
                                                      title=book.title;borrow=uc; state="lending"} in
                            if (fst rst) then (true,"You can keep the book")
                            else (DB.editUser uc book))
        else (false,(snd rst)))
    else (false,"This book cannot be lent");;

let rec lend_books uid list = lend_books_itr uid list []
and  lend_books_itr uid books lendable =
        if books == [] then lendable
        else
          let bid = List.hd(books) in
          let rst = lend_one_book uid bid in
          (if (fst rst) then lend_books_itr uid (List.tl(books)) (bid::lendable)
           else lend_books_itr uid (List.tl(books)) lendable) ;;
...
end
```

図 5. Circulation module

4.2.2 貸出モジュールの性質記述

貸出モジュールのインターフェイスとなる公開関数の満たすべき性質を，プログラム中の関数，定義した述語を用いた論理式で表す．以下一つの公開関数の性質記述について述べる (図 6)．

前提条件では，関数の入力の変数を引数とし入力の満たす条件を記述し，後置条件では入力の変数と返り値となる関数値が満たすべき条件を記述する．記述中には，記述を簡潔にし，論理式で表現しにくい処理を表現するため，述語を定義して用いる (性質記述と同様にソースコード中に記述)．

ここでは，貸出モジュールの入力が有効であれば，データベース上の貸出処理を正しく行うという性質について記述した．

前提条件では，処理が正常に行われる条件として，利用者 ID がペナルティリストにないこと (述語 havePenalty) をそれぞれ述語を交えた論理式で

表した (述語の定義は同様にソース中のコメントとして記述する).

- 入力の利用者 ID，本 ID が有効であること (データベースに対応するエントリがある，述語 isUser，isBook が対応)
- 利用者にペナルティがないこと (述語 havePenalty)

後置条件として，貸出処理が正しく行われたことを次のように表した．後置条件は，関数の入力と関数の返り値を表す ret_val を引数とした述語を用いる．またデータベースを表すために，関係代数を用いる．処理前，処理後の本データベースを preBDB, postBDB で表し，利用者データベースも同様に preUDB, postUDB で表す.

以下利用者を u, 入力の本集合を I，返り値の本集合を R，I と R の差集合を S で表す.

- R 中の本が全て,postBDB 上で u により貸し出された状態になっている (postBDB 上でその本の状態が "貸出中"，貸出者が u(の利用者コード) となっている．また postUDB 上で u の借本リストの中にその本のエントリがある．それぞれ述語 onLoan)
- R 中の本は全て，preBDB 上で貸出可能である (本の状態が "貸出可"．述語 lendable)
- R は，I の部分集合となっている (述語 isSubList)
- S 中の本は全て，preBDB，postBDB のどちらの上でも貸出できない状態である
- 貸出処理に関係のない部分のデータベースは変更されない (preBDB は，R の本以外のエントリについて postBDB と等しい，preUDB は，利用者 u 以外のエントリについて postUDB と等しい，述語 exBookDB, exUserDB,eqDB を用いる)

```
module type CirculationSig =
struct
...
val  lend_books: (int * int list * int) -> int list
(**
<pre>  (user, books, preBDB, preUDB) =
        (isUser(user, user, preBDB) and isBook(book, preBDB) and
          havePenalty(user, preUDB ) ) </pre>
<post>  (user, books, ret_val, preBDB, preUDB, postBDB,postUDB) =
  ( onLoan(books, user, postBDB, postUDB)  and isLendable(books,preBDB)
    isSubList(ret_val, books)  and (not (isLendable(subList(books, ret_val), preBDB)))
    and not (isLendable(subList(books, ret_val), postBDB)) and
    eqDB(exBookDB(books, preBDB), exBookDB(books, postBDB)) and
    eqDB(exUserDB([user], preUDB), exUserDB([user], postUDB))
 </post>
**)
...
end
```

図 6. A signature with properties of Circulation module

5 おわりに

本稿では，形式的検証を含むソフトウェア開発の枠組みとして，関数型プログラミング言語を対象とした形式的検証支援及び開発支援システムを提案した．ここではモジュールのインターフェイスを記述したファイルに，性質を記述し，ソースプログラムの情報と併せて用いて検証を行う．そのシステムの主幹エンジンであるブレスブルガー文真偽判定ルーチンは ML にて作成した．また図書管理システムの性質記述の具体例について述べた.

参考文献

[1] S. Chaki, E. Clarke, A. Groce, S. Jha, H. Veith, "Modular verification of software components in C," *Proceedings of the 25th international conference on Software engineering*, pp.385–395, 2003.

[2] F. Xie, J. C. Browne, "Verified Systems by Composition from Verified Components," *Proceedings of the 9th European software engineering conference*, pp.277–286, 2003.

[3] R. D. Jeffords, C. L. Heitmeyer, "A Strategy for Efficiently Verifying Requirements Specifications Using Composition and Invariants," *Proceedings of the 9th European Software engineering conference*, pp.28–37, 2003.

[4] S. Hazelhurst, O. Weissberg, G. Kamhi, L. Fix, "A hybrid verification approach: getting deep into the design," *Proceedings of the 39th conference on Design automation*, pp.111–116, 2002.

[5] 森岡澄夫，柴田直樹，東野輝夫，谷口健一, "全ての変数が存在記号で束縛された冠頭標準形ブレスブルガー文の真偽判定の高速化手法", **情報処理学会論文誌**, Vol.38, No.12, pp.2419–2426, 1997.

[6] D. C. Cooper, "Theorem Proving in Arithmetic without Multiplication," *Machine Intelligence*, No.7, pp.91–99, 1972.

[7] 北嶋暁，森岡澄夫，島谷肇，東野輝夫，谷口健一, "代数的手法を用いた CPU KUE-CHIP2 の段階的設計の正しさの自動証明", **電子情報通信学会論文誌**, Vol.J79-D-I, No.12, pp.1017–1029, 1996.

[8] 才村徹也，岡野浩三，谷口健一, "関数型言語 ML によるブレスブルガー文真偽判定ルーチンの開発", **電子情報通信学会技術報告**, vol.103, No.708, pp.7–12, 2004.

[9] B. Meyer, "Object-Oriented Software Construction," *Prentice-Hall*, 1988.

[10] "Objective Caml," `http://pauillac.inria.fr/ocaml/`.