社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE
SS2004-16(2004-8)

関数型言語ML向け形式的検証支援システムの試作

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 〒560–8531 豊中市待兼山町 1–3
E-mail: †{saimura,okano,taniguchi}@ist.osaka-u.ac.jp

あらまし 本稿では，関数型言語 ML 向けの形式的検証支援システムの試作の概要について述べる．本システムでは，検証者はモジュールのインターフェイス定義ファイルのコメント中に，要求仕様として，前提条件 (入力の満たす性質) と後置条件 (出力の満たすべき性質) を等式論理で記述し，仕様がプログラムにおいて満たされていることをプレスブルガー文真偽判定アルゴリズムなどを用いて判定する．本システムは，実用的な関数型言語 ML プログラムを対象にした検証支援系であること，性質記述を同種の情報と共に記述することでドキュメントとしての情報を充実させること，またその開発言語としても ML を用いることで，完成したシステムを用いてシステム自体を検証するという利点があることに特徴がある．

キーワード 関数型言語，ML，形式的検証，プレスブルガー文

A verification support system for functional programming languages ML

Tetsuya SAIMURA†, Kozo OKANO†, and Kenichi TANIGUCHI†

† Graduate School of Information Science and Technology, Osaka University
Machikaneyama 1–3, Toyonaka-shi, 560–8531 Japan
E-mail: †{saimura,okano,taniguchi}@ist.osaka-u.ac.jp

Abstract This paper presents a formal verification support system for a functional programming languages ML. The system requires a designer to describe pre-conditions and post-conditions as requirements in a comment area of each interface definition file. It decides the conformance between the program and the specification via decision procedure of Presburger sentences. The proposed system has the following features;(1) it aims at one of useful functional programming languages, ML;(2) the proposed description scheme has a natural way to enrich the value of whole document products;(3) it can be an example for our proposed method.

Key words functional language, ML, formal verification, Presburger sentences

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が，形式的検証における課題として，モデル検査 [1] に基づく手法における "状態爆発問題" や，定理証明系の自動化技術 [2] の欠如 (検証者の知識や技術に頼る部分が多い) 等が挙げられる．これを克服するために大きく複雑なソフトウェアを，基本的なモジュールに分割し検証を行う手法 [3], [4] 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 [5], [6] などが考案されている．理論的な研究と共に，人の作業と計算機の作業をうまく協調させ，効率良く検証を行う研究も必要である．

研究グループでは，そのような観点から主に代数的言語を対象に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，プログラムの正しさの検証支援方法や，ハードウェア設計自動化に関する研究を行ってきた [7]．検証支援においては整数を持つプログラムの効率的検証を目的としたプレスブルガー文真偽判定ルーチンの利用 [8] などに特色があり，今までに在庫管理プログラムなどの例題に対して，その有用性を調べてきた．

本稿では，研究グループで行ってきた検証手法を用いた関数型言語 ML 向けの形式的検証支援システムを考案し，そのプロトタイプを作成したことについて述べる．本システムの特徴として，1) 関数型言語 ML を対象としていること，2)ML のインターフェイス定義ファイルにプログラムの仕様を記述すること，3) 開発言語として，同じ関数型言語 ML を用いることの三つが挙げられる．

1), 3) の背景として，関数型言語 ML は，近年は実用的な処理系が複数存在しており，ML で作られた有名な定理証明系 HOL [9] などが存在するなど，実用的プログラム言語として認

められてきていることがある．しかし，MLを対象とした，仕様の詳細化の正しさに対する検証支援系は少ない．そこで，仕様がプログラムにおいて満たされていることに対して，抽象度の高い関数型プログラム自体を用いて検証支援を行う手法を考案した．MLプログラムにおいては，モジュール毎に，2)で述べたインターフェイス定義ファイルとして，公開関数の型定義を行ったファイルが存在する．考案した手法では，そのファイルのコメント中にモジュールの要求仕様として，公開関数の入力の満たす性質(前提条件)と，入力と出力の満たすべき性質(後置条件)をXML風タグを用いて記述する．2)のねらいは同種の情報を同じ場所に記述することで見やすさを高めることと，モジュールのドキュメントとしての価値を高めることにある[10]．XMLは電子的な文書管理のためのメタ言語であり，現在は電子的なデータ交換のためのフォーマットとしてよく用いられる．XML風タグを用いることで，ドキュメントとして用いるための発展性を意識している．また，3)により，システム自体が，完成したシステムのテストケースになることが可能であり，それによりシステムの評価と，システムの保守発展的な開発に役立つと考えている．

検証者は，モジュール毎に性質記述として，仕様記述のために導入した述語，ソースコード中に定義した関数を用いて，前述した前提条件，後置条件を等式論理の形で記述する．システムは「前提条件の成り立つ上で，(プログラム実行後)後置条件が成り立つこと」という論理式(検証対象式)を作成し，その中の述語や関数を定義で展開した式の真偽をプレスブルガー文真偽判定アルゴリズム等を用いて判定する．本システムの概要は，基本的に[11]において提案した形式検証支援システムのものである．検証の単位をモジュールとし，プログラムが要求仕様を満たしていることを検証するという立場において，抽象度の高いプログラム自体を検証に用いることで仕様記述によるモデル化の際の誤りや複雑さを少なくし，問題を論理式の真偽判定に帰着することで検証を半自動化することを目指している．

システムの適用例題として図書管理プログラムを考案し，一部のモジュールについて性質記述を行った．今後は，研究グループで行っていた在庫管理プログラム等の検証例題のノウハウを用いて全体の性質記述を行い，本システムに対して適用し，システムの評価と改善を行っていく予定である．

本稿では2.において提案する検証支援システムの概要と，例題を用いた性質記述例について述べる．3.では作成したプロトタイプシステムについて説明する．最後に4.において，今後の予定と検証支援システムの拡張について述べる．

2. 検証支援システムの概要

本システムでは，モジュール単位の検証を行う．モジュールに対して，検証者が性質記述を行い，システムはそれを用いて検証式を作成する．検証式は性質記述中の基本関数やプログラム中の関数定義等を用いて展開され，システムは展開後の式が真であることを確かめる．ここでは，性質記述の方法，検証式の展開，そしてプレスブルガー文真偽判定アルゴリズムの利用について説明し，最後にプログラム全体の検証について述べる．

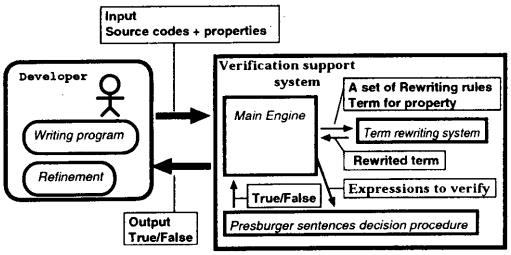

図1 Formal verification flow with the proposed system

2.1 性質記述

関数型言語MLでは，モジュールのインターフェイス定義として，公開関数のシグネチャ(関数名，型定義)を記述するファイルがある．このファイルのコメント中に，入力と出力に対応させる形で，前提条件と後置条件を等式論理の形で記述する．記述中には，論理演算，関数の入力に対応する変数，プログラム中の関数を用いることができ，また仕様記述のための基本関数，または記述を簡潔するための述語を定義し用いることもできる．基本関数は，等式論理で記述することが難しいプログラムの性質(整数やブール型以外の型や，入出力等)をうまくモデル化するために用いる．また，前提条件から後置条件を直接示すことが難しい場合はプログラムのコメント中に不変表明を行い，それを用いて検証を行う．

a) 前提条件，後置条件

前提条件は<pre>タグ，後置条件は<post>タグの中に記述する．システム中では，それぞれPre，Postという述語を用いて表す．Preは対応する公開関数の入力に対応する変数を引数として持ち，Postは Preの引数に加えて，公開関数の出力を引数として持つ．

条件として記述する論理式は，複数のプログラムの性質の論理積で表される．その1つの性質を表す論理式のまとまりはそれぞれ<prop>タグで囲む．システムでは，全ての<prop>タグ中の論理式の論理積が条件の論理式として表されるが，システムが論理式を性質毎に管理することで表示や修正の際に利点がある．

b) 不変表明

不変表明は<inv>タグの中に記述する．不変表明は，その直前に記述された式に対する不変表明として扱われる．式中で用いられている変数と，プログラム中の関数，または基本述語を用いた等式論理の形で記述する．

c) 基本関数

基本関数の定義は<lenmma>タグの中に記述する．基本関数の中で，ブール型を返すものを特に基本述語，またその基本述語間の関係を定義したものを補題と呼ぶ．

d) その他の記述

単純な置き換え等，記述を簡潔にするために用いる記述は<aux>タグ中に記述する．

2.2 検証式の展開

モジュールの一つの公開関数に対する検証式は，前提条件，後置条件を表す述語を用いて，単純に

$$Pre(a_1, a_2, ..., a_n)\ imply\ Post(a_1, a_2, ..., a_n, out)$$

という形で表すことができる．$(a_1, a_2, ..., a_n)$ は公開関数の引数変数を表し，out は，対応する公開関数に $a_1, a_2, ..., a_n$ を適用した返り値を表す．

これを，先程の前提条件，後置条件の定義により置き換えを行い，さらにその中で用いられる基本関数，プログラム中の関数についても同様に定義により置き換えを行っていく．プログラム中の関数は実際に実行される式に展開されていき，最終的に変数と一部の基本関数のみで表される論理式に変換される．基本関数，プログラム中の関数等は，左辺から右辺への書換え規則としてみて，項書き換えの性質を用いて自動的に行う．基本的な論理演算 (例：$term \wedge false == false$, $(term = term) == true$ 等，$term$ は変数および値，関数，基本関数からなる論理式) についても書換え規則として用いて，定義による展開と同時に式の簡単化を行う．検証式によっては，Knuth-Bendix のアルゴリズムを用いて書換え規則を正規化し，さらに書換えを行うことで真偽判定まで行える場合があると考える．

2.3 プレスブルガー文真偽判定ルーチンの利用

2.3.1 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，+，−，=，≧，≦，∨，∧，¬，∀，∃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 [13].

2.3.2 P 文真偽判定ルーチン

本システムで用いる P 文真偽判定ルーチンは，論理変数を扱えるように拡張を行い，関数型言語 ML の処理系である SML/NJ を用いて実装を行った [12]．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 [13] 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 [8] を適用する．

以後，P 文と記述する際には，この拡張を行った P 文を指す．

2.3.3 検証への利用

2.2 で展開した検証式を，P 文に変換し，式中の整数変数，論理変数に任意の値を代入した式が恒真であれば真である，という立場で真偽判定を行う．

検証式中の基本述語については，P 文で扱えるように，引数を含めた出現を，リテラルの並びを変数名とした論理変数に変換する．しかし，この場合，引数の $term$ の評価を行わないために，述語間で本来成り立っていた関係に関する情報が抜け落ちる可能性があり，真偽判定が行えない場合がある．したがって，述語間に成り立つ基本的な関係式 (補題) を検証式に導入する必要がある．補題中の変数には具体的な $term$ を代入する (補題のインスタンス化)．代入値は，検証式で用いられている $term$ などから，検証者が決定する．

2.4 プログラム全体の検証

関数型プログラムの実行は，メインモジュール中のメイン関数の実行である．したがって，プログラム全体の検証とは，いくつかのモジュールを統括しているメインモジュールの検証である．このとき，用いている他のモジュールの公開関数の性質 (「前提条件 imply 後置条件」で表される論理式) は，正しいと仮定して検証式中で用いる．このように，ある程度大きなプログラムに対して，下位のモジュールの中で，明らかに正しいものや，先に検証を行ったもの，または作成途中のものなどは仮定として用いて検証を効率的に行う．

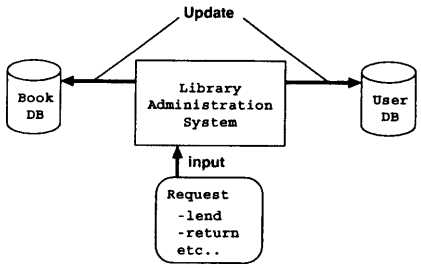

図 2 The overview of a library administration system

2.5 性質記述例

現在，例題として，図書管理システムを考案し，性質記述を行っている．ここでは，そのうちの一部の性質記述例を示す．記述には，ここでは OCaml [14](StandardML の派生言語) を用いた．

2.5.1 図書管理システムの概要

本と利用者の情報をデータベースに持ち，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 (図 2)．システムは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)

図書管理システムのモジュール構成を図 3 に示す．

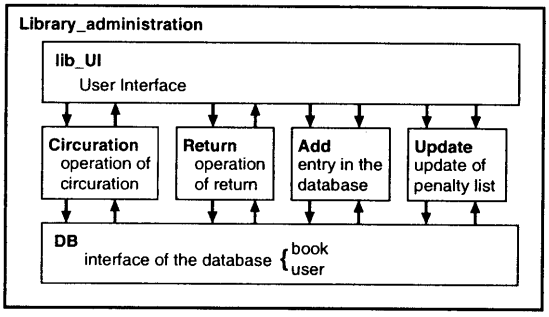

図 3 Modules of the library administration system

2.5.2 貸出モジュールの性質記述例

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

a) 貸出モジュールの処理

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 “lendable” 以外の場合は貸出不可．処理終了
- 本の状態を “checked_out” 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 “lendable” 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

b） 準備：基本関数の導入

データベースに関する処理は，関係代数を用いて表した．関係代数を用いるために導入した基本関数の一例を次に示す．

- Union(R1, R2)

データベース R1 と R2 の和を返す．

- Join (R1, R2, p(X1, X2))

データベース R1 の属性 X1 とデータベース R2 の属性 X2 のデータが述語 p を満たすエントリの自然結合を返す．

- Pr(Xlist, R)

データベース R における，属性リスト Xlist 中の属性の列からなるデータベースを返す．

- isEmpty(R)

データベース R が空 (NULL) であるとき，真となる述語．

以下は，関係代数の基本関数ではないが，記述の便宜上に設けた基本関数である．

- EachPr (p(X), R)

データベース R 中の任意のエントリについて，その属性 X のデータが述語 p を満たすときのみ，真となる述語．

- DBofList(list)

ML におけるリスト型を引数にとり，そのリスト要素をエントリとして持つデータベースを返す．

- Delete(R)

データベース R から勝手な一つのエントリを削除したデータベースと削除されたエントリとの組を値として返す．OCaml の組から要素を取り出す関数 fst と snd を用いると，fst(Delete(R)) が削除後のデータベース，snd(Delete(R)) が削除対象となったエントリの値をそれぞれ表す．ML におけるリスト型の操作とデータベースの操作の対応のために用いる．

```
<lemma>
prim1(db1,db2) ==
not( isEmpty(db1) ) imply
    ( Union(db1,db2) = db1 imply
        ( Union ( fst( Delete(db1) ), db2) = fst( Delete
(db1) ) )
</lemma>
```

図 4 A lemma for relational database

図 4 は，上に挙げたような基本関数の定義または関係を表す補題の一つである．この補題 prim1 は「データベース db1 が db2 の部分集合なら，db1 から一つのエントリを削除したデータベースも db2 の部分集合である」という関係データベースの性質を表している．

c） 貸出モジュールに対する性質記述

貸出モジュールの公開関数 lend_book の型定義に対して行った性質記述の一部を図 5 に示す．

ここでは，lend_book の入力 (本リストと利用者 ID) 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．

```
module type Circulation =
Sig
Val   lend_book : (book list, user)  ->  (book list);
(**
<pre>   Pre(b,u)==
not (isEmpty(Iset)) and isRegistered(u) </pre>
<post> Post(b,u,out) ==
<prop>isSubSet(Iset,Oset)</prop>
<prop>EachPr(state= “lendable” , PreOset)</prop>
<prop>EachPr(state= “checked_out” and user=u, PostOset)
</prop>
<prop>EachPr(state= “checked_out” and
state= “unlendable” , PreIset-PreOset)</prop>
...
</post>
<aux>
def out lend_book(b,u);
def Iset DBofList(b); def Oset DBofList(out)
def PreOset Join(PreBDB,Oset,*code=code);
...
</aux>
**)
```

図 5 A signature with properties of Circulation module

前提条件では，処理が正常に行われる条件として，次の性質を，それぞれ基本関数 (述語) を用いて記述した．

- 入力の本集合が空でないこと
- ユーザ ID がユーザデータベースに登録されていること

記述中の Iset，Oset は，本 ID のみをカラムとして持つデータベースを表し，<aux>タグ中にそれに関する記述がある．

後置条件として，貸出処理が正しく行われたことを示すために，例えば次のような性質を用いる．説明のため，以下利用者 ID を u，入力の本 ID 集合を I，返り値の本 ID 集合を R，I と R の差集合を S で表す．また，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．

- R 中の本は全て postBDB 上で u により貸し出された状態になっている (postBDB 上で，R に対応する本の状態が “checked_out”，貸出者 ID が u となっている)
- R 中の本は全て，preBDB 上で貸出可能である (本の状態が “lendable”)
- R は，I の部分集合となっている
- 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

図 5 の記述においては，簡潔な記述のために，引数の変数を lend_book 関数に適用した値は Out を，本 ID が Out のエントリの ID と同じ値を持つ処理前のデータベースは preOset を用いて表している．postOset も preOset と同様に定義され，Out に対応する処理後のデータベースを表す．

d） 検証式の展開

図 6 のように，等式論理で記述した基本述語やプログラム中の関数を左辺から右辺への書換え規則とみて，項書換えを行うことにより，最終的に変数や基本関数，またはそれを引数として持つ基本関数からなる論理式を得る．ここで，前述した Knuth-Bendix アルゴリズム，または P 文真偽判定アルゴリズムを適用する．

P 文真偽判定を用いる場合は，検証式中の基本述語は図 7 のように論理変数に変形し，具体的な *term* を代入した関連する補題の論理式を追加する必要がある．

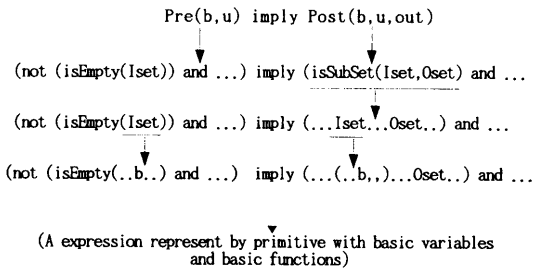

図 6　Term rewriting process

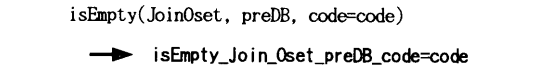

図 7　Transformation of a term into a propositional variable

3. プロトタイプの実装

実装には OCaml を用いた．OCaml 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．OCaml はオブジェクト指向を採り入れた実用的な関数型言語であるが，2. で述べたシステムでは，純粋な関数型言語としての性質を利用しているため，システムの実装には注意が必要である．OCaml 言語は純粋な関数型言語ではない機能も有しているが，それを使わずにプログラムすることも可能である．

最も単純な機能を持ち，OCaml 言語にいくつか制限をおいたサブクラスに対する検証支援システムを作成した．以下では，システムを実装した際の方針と検証の手順について述べる．

3.1 設計方針

対象は実装に用いた OCaml である．ただし，システムの簡単化のため，入力の OCaml 言語に対して次のような制限を課す．

（1） モジュール内での局所的なモジュール定義はない
（2） match 式，for 式，while 式等の一部の構文を用いない
（3） 関数は原則として非カリー化されているものとする
（4） オブジェクト指向に関する機能を用いない
（5） 参照は用いない

オブジェクト指向に関する機能を省いたのは，用いる手法が関数型言語に対するものであり，言語固有のオブジェクト指向機能に対する拡張は本質的ではないと判断したためである．検証手法では純粋な関数型言語の性質を利用しているので，副作用を許す参照型変数を用いると検証が難しくなる．しかし実用プログラムにおいては，参照型変数が使われると考えられるので，今後，対応について検討する必要がある．

しかし，これらの制限を置いても，検証支援システムの実用プログラムに対する有用性はそれほど損なわれないと考える．これらの制限は，言語の表現能力を制限をするが，関数型言語としての能力にはさほど影響を与えない．(2) については，if 式と再帰関数を用いれば同じ意味を持つプログラムは実装可能である．また，検証支援システムの対象として，大きなソフトウェアプログラム全体ではなく，その中の検証する必要のあるモジュール群を考えている．そのような部分に対する適用では，プロトタイプにおいても実用性は十分あると考える．

基本的に，検証するプログラムに関連するモジュールファイルは全て読み込まれることとする．読み込まれていないモジュールの関数が出現する場合，検証がうまく行われない (検証式の真偽判定が失敗する) 場合がある．本来，そのような場合には，2.4 で述べたように，その関数に関係する性質を用いることで検証を行う．OCaml に標準で提供される基本ライブラリ等についても，性質記述を導入するか，ソースコード自体を読む方が良いか，プロトタイプを用いて実験を行うことで調整することを考えている．

プロトタイプでは，項書き換えによる検証式の展開と，Knuth-Bendix による書換え規則の完備化，P 文真偽判定アルゴリズムの利用といった基本的な機能を実装する．

3.2 検証の手順

インターフェイスにはコマンドインターフェイスを用いた．以下の手順を繰り返し用いてモジュールに対する検証を行う．

e） モジュールの読み込み

モジュール定義，モジュールのインターフェイス定義ファイルを読み込む．

OCaml はトップレベル (OCaml のインタプリタ実行環境) でファイルを読み込み実行させる場合と，コンパイルして実行ファイルとして実行する場合がある．また，モジュール等は，コンパイルを行いオブジェクトファイルとして用いることができる．“filename.ml” というファイルをコンパイルすると Filename という名前のモジュールが生成されファイル中の関数は Filename モジュール中の関数となる．インターフェイス定義ファイル “filename.mli” が存在すれば同時にコンパイルされる．逆にトップレベルで用いるファイルでは，モジュールを用いたい場合は，ファイル中で明示的にモジュール定義，モジュールインターフェイス定義を記述する必要がある．

本システムでは，ユーザはまずトップレベル用のファイルか，コンパイル用ファイルかを選択する．コンパイル用ではモジュール名を入力すれば，その名前を持つ.ml ファイルと.mli ファイルを読み込み，ファイル中の関数定義は，入力されたモジュール中の関数として扱う．トップレベル用ファイルの場合は，ファイル名そのものを入力する．システムは，ファイル中で定義されたモジュール，モジュールインターフェイスを読み込み，モジュール定義外で定義された関数は Toplevel という仮想的なモジュール中の関数として扱う．

f） 検証対象モジュールの選択

読み込んだモジュールの中から，検証を行うモジュールを選択する．性質記述を行った公開関数が複数ある場合は，検証を行う公開関数も選択する．システムはその前提条件と後置条件から検証式を作成する．

g） 検証式の展開 (項書換え)

作成された検証式は，システム中に読み込まれたプログラムにおける関数，前提条件，後置条件と共に記述された基本関数等を書換え規則とする項書き換えの適用により展開される．項書き換えは，書換えが起こらなくなるまで，もしくは一定回数の書換えが起こるまで行う．

書換え後の式を表示し，ユーザはそれを見て，Knuth-Bendix アルゴリズムを用いて書換え規則を完備化してから再度書換えを行うか，P 文真偽判定アルゴリズムを用いるか選択する．書換え後の式が True または False の場合，検証は終了となる．

h） P 文真偽判定

検証式は P 文に変換する為 2.3.3 で述べた変換を行う．整数変数は，全称演算子で束縛し，式中の整数変数，論理変数が任意の値において検証式が成り立つことを検証する．このとき，補題のインスタンスを追加する必要がある場合は，式を表示した後，ユーザは，読み込んだ基本関数から追加する補題を選択し，その代入値を入力する．

P 文真偽判定ルーチンにより，結果は True または False または充足可能として返される (論理変数を持つように拡張されているため，充足可能の場合も生じる)．検証失敗時 (結果が False または充足可能の場合) は，繰り返し，補題の入力を行い真偽判定を行うか，検証を中断しプログラム及び性質記述の修正を行う．

4. 今後の展開

今回作成したプロトタイプシステムに対して，ある程度の例題を適用し，その有用性の評価を行う．また，それにより得られた知見を基にシステムの改善を行う．

4.1 例題への適用

2.5.2 で述べた図書管理システムを例題として考えている．研究グループでは，在庫管理システムのような良く似たシステムの性質記述を行ってきており，そのノウハウを生かしながら記述が行えると考えている．

4.2 検証支援システムの機能拡張

4.2.1 型の概念

関数型言語において，全ての式は型を持ち，OCaml 言語においても静的に型推論を行い，型チェックを行う．検証システムの項書換えにおいても，型の概念を付加しプログラムとして正しい項書換えを行う方が望ましい．現在は，OCaml 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．

4.2.2 入力クラスの拡張

システムの単純化のため制限したものと，参照型等の手続き型言語に近いものへの拡張に分けられる．前者は，単純に時間をかけることで拡張は可能であると考えているが，後者については現在の検証手法の単純な適用では対応していないことから，対処法及び検証手法の拡張を考える必要がある．

4.2.3 開発支援機能の考案

検証支援システムを核とした開発支援機能の考案を行っていく．現在は，一部の性質記述を自動生成する機能により検証者の負担を軽減すること，検証失敗時に，修正箇所を指摘する等のシステムのフィードバックの充実等を考えている．前者では，関数型言語特有の記述や，プログラムよく使われる関数や型に対する仕様記述や基本関数について，例題等を通して探査する必要があると考えている．

また，性質記述は XML 風タグを用いて記述していることから，性質記述をドキュメントとして取り出すことは容易にできる．性質記述や検証結果等をプログラム (モジュール) の情報として見やすい形で可視化することは意義があると考えている．

5. おわりに

本稿では，関数型言語 ML 向けの形式的検証支援システムの試作の概要について述べた．本システムは，実用的なプログラミング言語である ML を対象とすること，インターフェイス定義ファイルのコメント中に仕様を記述することでソースファイルのドキュメントとしての情報を充実させること，開発言語に ML を用いることでシステム自身の検証をシステムで行うという利点があることなどに特徴がある．記述した仕様がプログラムにおいて成り立つことを，プログラム自体を用いて論理式に帰着し，プレスブルガー文真偽判定ルーチン等を用いて半自動で検証を行う．今後は，図書管理システムなどの例題にプロトタイプシステムを適用し，その評価を行う予定である．

文　献

[1] E. M. Clarke, O. Grumberg, D. A. Peled, "Model Checking," MIT press, Cambridge, 1999.

[2] J. Harrison, M. Aagaard, "Fast Tactic-based Theorem Proving," 13th International Conference Theorem Proving in Higher Order Logics, LNCS1869, pp.252–266, Portland, Oregon, Aug.2000.

[3] 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, Portland, Oregon, May 2003.

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

[5] 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, Helsinki, Finland, Sept.2003.

[6] 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, New Orleans, Louisiana, June 2002.

[7] K.Okano, Y.Kitahama, A.Kitajima, T.Higashino, K.Taniguchi, "Formal Verification of CPU in Laboratory Work," Proceedings of the 2001 International Conference on Microelectronic Systems Education(MSE 2001), pp.32–36, Las Vegas, Nevada, June 2001.

[8] 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プレスブルガー文の真偽判定の高速化手法"，情処論，Vol.38, No.12, pp.2419–2426, 1997.

[9] M. J. C. Gordon, T. F. Melham, "Introduction to HOL: a theorem proving environment for higher order logic," Cambridge University Press, New York, 1993.

[10] H. D. Foster, A. C. Krolnik, D. J. Lacey, "Assertion-Based Design, 2nd Edition," Kluwer Academic publishers, Boston, 2004.

[11] 才村徹也，岡野浩三，谷口健一，"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"，ソフトウェアシンポジウム論文集 2004, pp.53–57, June 2004.

[12] 才村徹也，岡野浩三，谷口健一，"関数型言語 ML によるプレスブルガー文真偽判定ルーチンの開発"，信学技報，Vol.103, No.708, pp.7–12, Mar.2004.

[13] D. C. Cooper, "Theorem Proving in Arithmetic without Multiplication," Machine Intelligence, No.7, B. Meltzer, D. Michie, pp.91–99, Edinburgh University Press, Edinburgh, 1972.

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