対話型Ｌｉｓｐプログラム検証システムの試作

7L-4

松倉　隆一　伊藤　貴康
(東北大学工学部)

１．はじめに

プログラムの検証に関する研究が様々な言語を対象として行われてきた。Ｌｉｓｐについてもこれまでにいくつかの研究があり、実際に計算機上で検証システムが実現されているものもある。筆者らはCartright[1]のシステムを参考にして、第一階論理に基づくTyped Pure Lispの対話型検証システムを試作したので報告する。

２．試作システムの概要

試作システムは第一階論理を基にしてTyped Pure Lispプログラムの性質を対話的に検証するものである。

２．１ Typed Pure Lisp

Typed Pure Lispのデータタイプは図１のように定義されている。Lispにタイプ付けを行う事はプログラミング上からは好ましいこととは言えないが、タイプの導入はプログラムの検証には有効である。すなわち、

(1)　プログラム検証におけるタイプチェックの活用。

(2)　対話型検証を進める上でのタイプ情報の有効利用。

試作システムはTyped Pure Lispに対する検証システムであるが、実用性を高めるためにタイプ推論の機能を具備させタイプ付きでないPure Lispに対しても適用できるように機能拡張を図っている。図２にはTyped Pure Lispの基本関数をあたえた。

２．２ 対話型Lispプログラム検証の考え方

システムの対象となるLispプログラムの性質は等号を含む第一階の論理式（これをステートメントと呼ぶ）で表現する。Gentzen流で書かれた推論規則が後述の表１のような対話型コマンドとして適用される形態になっている。このシステムの１つの特徴はLisp関数及び論理記号に対する簡約化能力を持っていることで、ユーザーの指示により規則を適用した後で自動的にステートメントの簡約化を行う。簡約化規則の例を図３に示す。このほかに、証明済みの定理をシステム内に構築し、簡約化規則として他の証明に利用する機能も一部実現されている。

２．３ 試作システムの基本構成

試作した検証システムの構成を図４に示す。システムはSymbolics3670上でCommon Lispによって実現されている。プログラムの規模はソースリストで約3500行である。システムは常にユーザーと対話をしながら処理を行っている。このシステムには３つの実行モード、システムモード・プログラムモード・証明モードがある。実行モードは制御部により制御されている。

２．４ 構文解析部

このモジュールはシステムモードでコマンドPROGRAMによって起動される。プログラムモードになる。このモードで

(1)　プリミティブなデータタイア
atom型　Typed Lispのデータオブジェクト　ただし次の４つのオブジェクトを除く。
NIL, ZERO, TRUE, FALSE型　Lispのオブジェクト NIL, ZERO, TRUE, FALSEに対応するタイア。
(2)　プリミティブなオブジェクトから構成されるタイア
①　列挙型　データオブジェクトの有限集合として定義されるタイア。
TYPE BOOL = (TRUE FALSE)
②　構造型　簡単なタイアからC($S_1:T_1,\ldots,S_n:T_n$)の形で定義されるタイア。Cはこのタイア名、Siはselector関数、T_iはその要素のタイア。
TYPE PAIR = PAIR(ATOM1:ATOM, ATOM2:ATOM)
③　結合型　いくつかのタイプの和集合で表されるタイア
$T_1;\ldots;T_n$で表しT_iは定義されたタイアのサブタイアという。
TYPE EXT-PAIR = NIL;PAIR
④　再帰型　再帰的な構造型を含む結合型。再帰的な構造型とは，構造型の要素のタイアが，再帰型として定義されるタイアと同一のもの。
TYPE NATNUM = ZERO;SUC(PRED:NATNUM)
(3)　any型　全てのタイアの結合型。

図１　Typed Pure Lispのデータタイア

(1)　基本関数　（関数名、引数のタイア、値のタイア）

equals	any x any	bool	２個の引数が同じ値ならばTRUEそうでなければFALSE
substr	any x any	bool	第１引数が第２引数の副構造*ならばTRUEそうでなければFALSE
and	bool x bool	bool	２個の引数が共にTRUEならばTRUEそうでなければFALSE
or	bool x bool	bool	２個の引数の少なくとも一方がTRUEならばTRUEそうでなければFALSE
not	bool	bool	引数の値がTRUEならばFALSE, FALSEならばTRUE
:T	any	bool	引数のタイアがTであるときTRUEそうでなければFALSE
constructor関数	$C(S_1:T_1,\ldots,S_n:T_n)$		引数のタイアはT_1x..xT_n 関数の返す値のタイアC
selector関数	S_i		引数のタイアはC，関数の返す値のタイアT_i

*副構造　ある構造が他の構造の部分構造となっているとき、副構造であるという。たとえば、ZEROはSUC(ZERO)の副構造。

(2)　ユーザー定義関数
function function-name($P_1:T_1,\ldots,P_n:T_n$):T = ξ
function-nameは識別子，P_iは引数，T_iは引数のタイア，Tは関数の返す値のタイア，ξは式でありP1,...,Pn以外の変数を含まない，式は次のように定義される。
①　if $\xi_1$thenξ_2 else ξ_3　　但しξ_1 ξ_2 ξ_3は式
（ξ_1の値がTRUEのときξ_2FALSEのときξ_3）
②　case　$\xi$of　$T_1:\xi_1;\ldots;T_n:\xi_n$　　但しξのタイアは$T_1,\ldots,T_n$の列挙型、結合型、再帰型。ξ_1は式。T_iをインデックスと呼ぶ。
（ξのタイアがT_iのときξ_iを返す）
③　関数の呼び出し
$f(\xi_1,\ldots,\xi_n)$、not(ξ)、ξ:T
④　プリミティブなデータオブジェクト

図２　Typed Pure Lisp の関数

if TRUE then ξ1 else ξ2 =>ξ1
if FALSE then ξ1 else ξ2 =>ξ2
if undefined then ξ1 else ξ2 =>undefined

図３　if then elseに関する簡約化規則

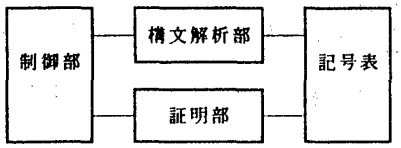

図４　試作システムの構成

An Interactive Lisp Program Verifier Based on First Order Logic
Ryuichi MATSUKURA, Takayasu ITO
TOHOKU UNIV.

プログラムが入力されると、ただちに構文解析・タイプチェック・停止性チェックが行われ、問題がなければシステム内に登録される。

(1)構文解析　プログラムのシンタックスをチェックし、エラーが生じた場合にはメッセージを返す。

(2)タイプチェック　プログラムのタイプチェックを次の規則に従って行う。

1 if-then-else文の第1引数のタイプはbool型またはそのサブタイプでなければならない。

2 case-of文の第1引数のタイプはインデックスのタイプの結合したものでなければならない。

3 関数の引数のタイプは、その関数で定義されているタイプのサブタイプでなければならない。

Lisp関数の引数に対する簡単なタイプ推論を実現しており、また、ユーザーによるタイプ付けと融合させることもできる。タイプはある程度まで推論できるので、システムに推論させてユーザーがそれを補うこともできる.

(3)停止性チェック　プログラムの停止性はユーザーの指定により行うことにしている。停止性の証明は、関数の定義における再帰呼び出しの引数が、整列順序になっていることを示せばよいわけであるが、簡単な場合を除いてシステムに自動的に行わせることは困難である。停止性証明を対話的に行いやすくするためにメニューを用意しつつある。

2．5 証明部

証明部はシステムモードにおいてコマンドPROVEで起動される。証明モードのコマンドは第一階論理の推論規則に対応している。推論規則はステートメントを1個ないし2個以上のステートメントに書き換えるもので、コマンドではユーザーが使いやすいようにユーザーが指定すべきことは、できるだけ少なくなるように設計した。このため一部のコマンドは、推論規則を簡略したものであったり、システムがパラメターを補ったりしている。証明モードのコマンドによりステートメントは次のように評価される。証明コマンド処理は、はじめに引数のシンタックスをチェックし、適用条件を調べ、問題がなければステートメントを書き換える。このとき、シンタックスのチェックでは、構文解析部の機能を部分的に利用している。パターン照合は、証明コマンドINSTANTIATE LEMMAにおいて帰納仮定、補助定理の変数に対する適当な代入をさがす処理をしている。最後に簡約化は書き換えられたステートメントを簡約化規則により評価し制御部に値を返す。

なお，よく使われるコマンドTYPE, INDUCTに対応する推論規則を紹介する．（他の推論規則は文献(1)を参照）

推論規則はステートメントを書き換える規則である．ステートメントは一般に$a_1 \hat{} a_2 \hat{} .. \hat{} a_m => B$の形をしているので，$A=\{a_1, a_2, ..., a_m\}$と表すことにする．（^はandを示す）

(1)Type split rule

$$\frac{AU\{e:T_1\}=>B, \ldots, AU\{e:T_n\}=>B,\ A=>e:T}{A=>B}$$

eは$T_1, ..., T_n$を結合したタイプをもつ．コマンドではeを指定する．

(2)Induction rule

$$\frac{AU\{\forall z_1, ..., z_n[A_1 U\{substr(e_1, e)\}=>B_1\}=>B,\ A=>e:T}{A=>B}$$

式eの変数を$z_1, ..., z_n$で置き換えたものがe1で，この置換をA, BにするとA_1, B_1になる．$\forall z_1, ..., z_n[A_1 U\{substr(e_1, e)\}=>B_1$は帰納仮定であり，コマンドではeを指定する．

コマンド	引数	機能
システムモードコマンド		
PROGRAM		プログラムモードに入る
PROVE	theorem	証明モードに入り，theoremを証明する
BYE		システムを終了する
プログラムモードコマンド		
DEFTYPE	type-def	タイプ定義文の構文解析を行い、登録する
DEFUN	func-def	関数定義文の構文解析を行い、登録する
証明モードコマンド		
EQSUBST	arg*	条件部の等式を利用して、式の書換えをする
TRANSUB		条件部にsubstr(e1, e2), substr(e2, e3)の論理式があるときsubstr(e1, e3)を加える
CONSEQUENCE	formula	ステートメントA=>Bに対して2個のステートメントA=>formula, A^formula=>を生成する
DELETION	arg*	不要な論理式を削除する
REPLACE	expr var	式exprを変数varに置き換える
TYPE	expr	式exprのタイプにより場合分けをする
SPLIT	formula	論理式formulaの真偽の場合に分ける
CONSTRUCT	var	構造型タイプの変数varをconstructor関数を用いた形式に置き換える
INDUCT	expr	式exprが再帰型の時、帰納仮定を生成する
INSTANTIATE		帰納仮定を具体化する
LEMMA	statement	補助定理を用いてゴールを簡単にする
BACK**		現在のゴールの前のゴールに戻る

*複数の引数を指定できる　**BACKは証明規則ではない。

表1　検証システムの主なコマンド

1. 不要な論理式の削除、式の変数への置き換え
2. 最も内側の関数の展開　①再帰関数のとき　帰納仮定を生成し展開　to 3
   ②非再帰関数のとき　そのまま展開する　to 1
3. パターン照合し、帰納仮定により書き換える。
   パターン照合しなければ、補助定理等を用いて書き換え仮定を利用する
4. 等式を利用して式を書き換える　to 1

図5　証明の一般的な戦略

3．証明の構成法

試作した検証システムを使って証明を行うには、Lispプログラムの性質についての相当な知識が必要である。証明の戦略にも技巧的なものが要求されるが、簡単な場合には図5のような証明戦略を適用することによって、解決できることが多い。問題の性質に応じた証明戦略を用意しておき、システムがユーザーに証明の方向を示唆することができるならば、対話的証明の環境はたいへん良くなると思われ、このような機能をProof Editorという形で実現する事を検討中である。

4．まとめ

Lispプログラムの性質を証明するシステムの概要と実現法について述べた。対話的証明を支援するメカニズムの開発は今後の課題である。

謝辞　日頃熱心に討論頂く松山隆司助教授をはじめとする研究室の方々に深く感謝致します。

本研究を行うにあたり、文部省科学研究費一般研究(A)60420035および特定研究(1)61102003の補助を受けた。

参考文献

(1)R. Cartwright: User Defined Deta Type as an Aid to Verifying, Automata Languages and Programming(1976) pp228-256

(2)伊藤、松山：推論ソフトウェアの構成、電気通信学会誌（掲載予定）