Lispの証明システム

中西正和（慶應義塾大学工学部）

1. はじめに

Lispプログラムのためのプログラミング・ツールとしては，pretty printerやエディタ等，多くの実用的なものが存在している。ここでは，主として簡単なLispプログラムのための性質チェック・プログラムとでも言うべき道具について述べる。

この道具は，利用者の書いたLispプログラムが，構文エラーなしにLispシステムに入力されたときから働く。利用者のプログラムが意図した性質を持っているかどうかをチェックしたり，証明を出力したりする。

純Lispを対象とした証明系としては，Boyer-Moore Pure Lisp Theorem Prover[1][2]が有名であり，実用的であると思われるが，さらに現実のプログラムに利用することを目指して，本システムが考えられた。

システムの名前を仮にCANDYと名づける。CANDYは，Lispで書かれ，Lisp言語のプログラムをデータとして扱う関数の集合として定義される。

2. システムの利用

まず，簡単に利用法について述べる。システムには，次の3つの種類の関数が含まれる。

1) 宣言関数
2) 証明系
3) 出力関数

宣言関数は，チェックすべき利用者プログラムがどれであるかを指摘したり，特別な性質をシステムに知らせる役割を持つ。例えば，チェックしたいEXPR関数が $f_1, f_2, \cdots, f_n$ であったときは，次のような形式を評価させる。

(getfunc f_1 f_2 ⋯ f_n)

また，証明に必要な諸性質をシステムに知らせるための関数もいくつかある。

(declareprop p_1 p_2 ⋯ p_m)

は，$p_1, p_2, \cdots, p_m$ が命題あるいは述語としてプログラム中で利用されていることを宣言した形式である。これは，証明中の場合わけのときに利用される。

関数の代数的性質や補助定理等は，assumptionとして，次の形で入力することができる。

(assume a b)

ここでaはパターンであり，証明中にaのパターンをbに置きかえてもよいことを表わしている。パターン中の変数は，次の形式を評価して宣言する。

(declarepatvar v_1 v_2 ⋯ v_n)

例えば，次のように書くことができる。

```
(DECLAREPATVAR X Y)
(ASSUME (CAR (CONS X Y)) X)
```

条件付きassumptionは，次のような形式で宣言する。

(condassume pattern condition value)

conditionは，パターン変数を含む命題であり，パターンマッチングが行なわれてパターン変数に値が代入されたときに命題が評価されるが，これが真になるときのみ，パターン・マッチングは成功したとみなされる。

証明を行なうためには，証明すべき「文」を入力する。文は，次の形を持つ．

(*arrow a b E)

ここで，aとbは 値を表わす式（CANDY形式と呼ぶ）であり，Eは状況である。すなわち，これは「Eの状況では，aの値がαならばbの値もαである」と読む。以後，これを次のように書く。

$a \Rightarrow b; E$

証明は，次の形の形式を評価することで得られる。

(prove s)

ここでsは文である。

3. 利用の例

2つの階乗の関数を定義して，その関係を調べることを考えよう。まず，FとGが次のようなEXPR関数であるとする．

```
F<= (LAMBDA (X) (COND ((ZEROP X) 1) (T (TIMES X (F (SUB1 X))))
    ))
G<= (LAMBDA (X Y) (COND ((ZEROP X) Y) (T (G (SUB1 X) (TIMES
          X Y))) ))
```

getfuncによってFとGを宣言した後，次のものを評価する。

(PROVE (*ARROW ((F ?) X) ((G ?) X 1) NIL))

これは「『Fを何回か（?回）適用した後，(F X)の値がαになったとしたら，Gが何回か（?回）適用されて(G X 1)の値がαとなる。ただし，どのような状況下であっても』ということが証明できるか」とでも読めばよい。

この証明で必要なことは，TIMESに関する性質等，いろいろあるが，予めそれらを知ることは困難であるから，とにかく証明を試みる．仮定が欠落しているので，証明は不可能であり，証明系はbacktraceを行なう。利用者はこれにより，欠落条件を知ることができる。

証明系は，会話的に利用する。欠落している仮定は，適当な時点で与えて，再び証明を試みる。

4. 証明系

証明のアルゴリズムは，[3]の体系を基盤としている。しかし，[3]の体系がLKのsequentと同様の左辺，右辺に複数のformulaを持つ式を扱っているのに対し，CANDYでは左辺，右辺はそれぞれ1つだけのformulaを持つ．

公理

次の形の文は公理である。公理はこれ以上変換できない。

1) $\bot \Rightarrow A; E$
2) $A \Rightarrow A; E$

ここで⊥は，undefined constant であり，Aは任意の CANDY 形式を表わす．Eは環境であるが，この場合，Eはどのような環境でもよい．

環境

環境を次のように書く。

$\langle m_1/p_1\rangle, \langle m_2/p_2\rangle, \cdots, \langle m_k/p_k\rangle$

ここで，p_i は命題であり，m_i はその値である。環境は左から右へ探すこととする。環境に新しい要素が加わるときは，左端に入る。$\langle m/p\rangle$ が E に加えられるとき，新しい環境を次のように書く。

$\langle m/p\rangle : E$

左から右へ E を探した結果，$\langle m/p\rangle$ が見つかることを，

$E(m/p)$

と書く。

CANDY 形式

CANDY 形式は，文の左辺，右辺にある形式であるが，これは"値"を表わすものと解釈される。Lisp の式 e は，次のように CANDY 形式 e′ に変換される。

i) e がアトムで EXPR 関数でなければ，$e' = e$ である。

ii) e がアトムで EXPR 関数ならば，

$e' =$ (e (DIFFERENCE *N* 1))

ここで *N* は，適用回数を表わす特別な変数（アトム）である。

iii) e がラムダ式 (LAMBDA (v_1 v_2 … v_n) d) ならば，

$e' =$ (GAMMA *N* (v_1 v_2 … v_n) d')

iv) e が (e_1 e_2 … e_n) の形で e_1 が LAMBDA でなければ，

$e' = (e_1' \; e_2' \; \cdots \; e_n')$

例えば，FAC が EXPR 関数で，次の定義が与えられているとしよう．

```
(LAMBDA (X) (COND ((ZEROP X) 1)
                  (T (TIMES X (FAC (SUB1 X)))) ))
```

(getfunc FAC) を評価すると，次のような，対応する CANDY 形式が作られる．

```
(GAMMA *N* (X) (COND ((ZEROP X) 1)
                     (T (TIMES X ((FAC (DIFFERENCE *N* 1))
                                  (SUB1 X)))) ))
```

これは，CEXPR property として FAC に与えられる．

CANDY 形式は，次のように解釈される。

i) e′ がアトムなら，これは e の値である。

ii) e′ が ((fn m) e_2' … e_n') ならば，e′ は (fn e_2 … e_n) の値で，fn が m 回以下呼ばれている。

m が"?"であることは，いくつかはわからないが，ある m が存在することを表わしている。

簡単化

文の中のCANDY形式は，証明の過程で，新しい環境ができる度に簡単化される。

(COND $(p_1\ e_1)(p_2\ e_2)\cdots(p_n\ e_n)$)

は，EがE(t/p_1)に変化したときはe_1に簡単化される。ここでtは真を表わす。EがE(f/p_1)に変化したときは，(COND $(p_2\ e_2)\cdots(p_n\ e_n)$)に簡単化される。fは偽を表わす。

(((GAMMA i $(v_1\ v_2\ \cdots\ v_n)\ e)\ m)\ a_1\ a_2\ \cdots\ a_n$)

は，eに簡単化されるが，このとき，現在の環境に次のような新しいエントリがつけ加えられる。

$\langle a_n'/v_n\rangle, \cdots, \langle a_1'/v_1\rangle, \langle m'/i\rangle$

ここで各a_i'とm'は，a_iとmの簡単化された形である。eは，この新しい環境下で簡単化される。

仮定の適用

定理や公式の形での仮定の適用は重要であるが，汎用のパターン・マッチング機能を用いようとすると遅くなるので，できる限り限定した形を用いる。

(assume $(f\ a_1\ a_2\ \cdots\ a_n)\ e$)

の宣言で，eの中に$v_1, \cdots v_k$というパターン変数が存在するとき，fnに次のようなPATTERN propertyがつけ加えられる。

$((z1\ z2\ \cdots\ zn)(a_1\ a_2\ \cdots\ a_n)\ e)$

文の中に$(f\ c_1\ c_2\cdots\ c_n)$の形が現われると，a_iとc_iのマッチングが行なわれ，すべてがマッチしたところでeに置きかわる。ラムダ式に対応する(GAMMA …)の形式のマッチングのときも，同様になる。すなわち，上の式でfが(g m)の形のときは，次のようなPATGAMMA propertyがgに与えられる。

$(m\ (z1\ z2\ \cdots\ zn)(a_1\ a_2\ \cdots\ a_n)\ e)$

分解

簡単化も仮定の適用もできない場合は，次の分解規則が適用される。

i) 文の左辺に条件式

(COND $(p_1\ e_1)\cdots(p_n\ e_n)$)

がある場合，文 $a \Rightarrow b; E$ は次の2つの文に分解される。

$a \Rightarrow b; \langle t/p_1\rangle : E$

$a \Rightarrow b; \langle f/p_1\rangle : E$

どちらの文も証明されなければならない。

ii) 文の右辺に条件式

(COND $(p_1\ e_1)\cdots(p_n\ e_n)$)

がある場合，次のどちらかを証明する。

$a \Rightarrow b; \langle t/p_1\rangle : E$

$a \Rightarrow b; \langle f/p_1\rangle : E$

選択は利用者によりなされる。