OL-resolutionにもとづく論理的プログラム合成系・Pilot II

琴野　実,大村伸一,東本謙治,馬渡幸夫　　謝　章文
（京産大・理）　　（京産大・計科研）

0. はじめに

論理的プログラム合成(LPS)は，与えられたformal specification(spec)から，それに対する正当性の保証されたprogramを合成する過程で，一階述語論理の演えき能力や定義可能性などを用いる自動プログラム合成の理論である。specの主要な部分は，一階述語論理によって記述されるAxiomsとconjectureである。conjectureは目的のプログラムの入出力関係を示し，Axiomsはconjectureと基本演算・述語などとの間の関係を示す。LPSには，二つの機構が存在する。一つは，conjectureのAxiomsからの証明図を構成する機構であり，他の一つは，その証明図から目的プログラムに関する情報を抽出する機構である。情報抽出機構に関しては，謝[1]において明らかにされている。一方，証明機構の自動化のために，自動証明向きの推論規則であるresolutionをLPSでは採用する。だが，tree searchに基づくresolution-refutation methodは，treeのlevelに対して，searchの可能性がexponential orderで増加するため，極めて非効率である。さらに，LPSはすべての反例を求めることが要求され，単に一つの反例を求めれば事足りる自動証明に比して問題は複雑であり，効率の問題が極めて重大な課題となってくる。

先に報告されたLPSシステムpilot1では，情報抽出原理の実験に主眼が置かれていたため，効率に関する調査には十分な能力を持たなかった。[7]

pilot2では，効率化における問題点を明らかにするため，効率の高い反証法の一つであるOrdered Linear Resolution(OL-resolution)を用いた。具体的には，十段余りのlevelを持つ証明図の構成(MSORTなど)を目指した。OL-resolutionは；completeであり，特定のInterpretationに依存せず，implementが容易である，などの理由から選ばれた。なお，pilot2はpilot1と同じくformulaの合成機能を加えたcalculus K_1に基づく。

以下では，LPS，OL-resolutionの概説，implementにおける手法の考察，pilot2のシステム構造，および実験を通して明らかになった効率対策上の問題などについて述べる。

1. LPSとOL-resolution

1.1 LPSについて

LPSは，resolution-refutation法に基づくため，証明機構ではSpec(Conjectureの否定を含む)から矛盾(□;empty clause)を導きだす。また，情報抽出機構は，additional expression(a-exp)の規則とS-ruleと呼ばれるLPS固有の規則により，その反証図から情報を抽出する。a-expは，$[(\beta,\delta),\alpha]$の形で表わされ，vital clause(Conjectureとその子孫)に付加される。α,β,δは初期値として，それぞれφ(空集合)，入力変数の並び，出力変数の並びを持ち，推論規則の適用に従ってそのmost general unifier(mgu)がa-exp全体にかけられる。(図1.b参照)　S-ruleは，vital clause

から executable predicate を消去し，その complementary literal を a-exp の α に移すものである。このようにして，a-exp に情報が抽出される。また，S-rule を 1 回以上使った refutation を conditional refutation (c-refutation) と呼ぶ。図1では，Spec (a) から C-refutation (b) を構成し，□ の a-exp から Recursive 定義系 (c) を作る。なお，(a′) は C-refutation の resolvents である。

(a)
1. $\overline{P}(x, y)$　　x:入力変数，y:出力変数
2. $P(0, 1)$
3. $\overline{P}(x_3-1, y_3) \vee P(x_3, y_3 \times x_3) \vee x_3=0$
4. $P(x_4, fact[x_4])$　　fact: 出力関数

　　executable predicate ; =

(b)

(i) 2.　1. $[(x, y), \phi]$

　　R $\{0/x, 1/y\}$

　　□ $[(0, 1), \phi]$

(ii) 3.　1. $[(x, y), \phi]$

　　R $\{x/x_3, y_3 \times x/y\}$

4.　5. $[(x, y_3 \times x), \phi]$

　　R $\{x-1/x_4, fact[x-1]/y_3\}$

　　6. $[(x, fact[x-1] \times x), \phi]$

　　S

　　□ $[(x, fact[x-1] \times x), x \neq 0]$

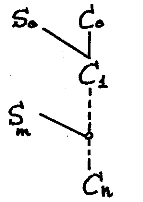
(a′)
5. $\overline{P}(x-1, y_3) \vee x=0$
6. $x=0$

(c)
$$fact(x) = \begin{cases} 1 & \text{if } x=0 \quad \text{(i)から} \\ fact(x-1) \times x & \text{if } x \neq 0 \quad \text{(ii)から} \end{cases}$$

$\{t_1/v_1 \cdots t_n/v_n\}$ は mgu で v_i に t_i を代入すること，R は binary resolution，S は S-rule を適用したことを示す。また，side clause top clause にある数字は (a) にある番号の clause を使うことを示す。

[図1] factrial の合成例

1.2　OL-resolution について



[図2]

Linear Resolution の deduction tree は図2のような形である。C_i ($i=0,\cdots,n$) を center clause，S_j ($j=0,\cdots,m, m<n$) を side clause という。特に，C_0 を top clause という。

OL-resolution の推論規則を示す。(詳しくは参考文献[8] 7章)
OL-resolution では，order を center clause に入れる。order の目的は，resolved literal の候補を一意に限定することであり，order の付け方は重要ではないので，ここでは左側の literal の方が高いとする。(参考文献[8]での order とは逆である。) 一番 order の高い左端の literal を first literal と呼ぶ。L_i は literal，L_j と L_k の mgu を $\sigma = mgu[L_j; L_k]$，L_i の complementary literal を $\overline{L_i}$ とする。

- Ordered Binary Resolution [OBR]

$$\frac{L_7 \vee L_6 \vee L_5 \qquad L_4 \vee L_3 \vee L_2 \vee L_1}{(L_7 \vee L_5 \vee \boxed{L_4} \vee L_3 \vee L_2 \vee L_1)\sigma} \; ; \sigma = mgu[L_4; \overline{L_6}]$$

　　center clause ; $L_4 \vee L_3 \vee L_2 \vee L_1$
　　side clause ; $L_7 \vee L_6 \vee L_5$

- Frame Deletion [FD]

$$\frac{\boxed{L_4} \vee L_3 \vee L_2 \vee L_1}{L_3 \vee L_2 \vee L_1}$$

- Merging Right [MR]

$$\frac{L_4 \vee L_3 \vee L_2 \vee L_1}{(L_3 \vee L_2 \vee L_1)\sigma} \; ; \sigma = mgu[L_1; \overline{L_4}]$$

- Reduction [RD]

$$\frac{L_4 \vee L_3 \vee L_2 \vee \boxed{L_1}}{(L_3 \vee L_2 \vee \boxed{L_1})\sigma} \; ; \sigma = mgu[L_1; \overline{L_4}]$$

Ordered Binary Resolution では，center clause の first literal L_4 を resolved literal と

し，それにframeを付ける。side clauseのresolved literal L_6は消す。frameの付けられたliteral $\boxed{L_4}$をframed literalという。Merging Rightでは，orderの低いliteral L_1を残しL_4を消す。Frame Deletion, Merging Rightはcenter clauseに対する処理であり，Frame Deletion, Reductionは，first literalを消すのが特徴である。

1.3 Merge Sortの合成

OL-resolutionを用いたLPS procedureをMerge Sortの合成を例に示す。

Spec　Conjectureの否定；1. $\bar{R}(x,y)$　　　x；入力変数，y；出力変数

Axioms　；2. $\bar{R}(x_2,y_2)\vee S(y_2)$　　3. $\bar{R}(x_3,y_3)\vee I(x_3,y_3)$

4. $\bar{S}(y_4)\vee \bar{I}(x_4,y_4)\vee R(x_4,y_4)$

5. $\bar{I}(x_5,y_5)\vee I(cons[u_5,x_5], merge[u_5,y_5])$

6. $\bar{S}(y_6)\vee S(merge[x_6,y_6])$　　7. $I(x_7,y_6)$　　8. $S(nil)$

9. $x_9=nil\vee \bar{R}(cons[car(x_9),cdr(x_9)],y_9)\vee R(x_9,y_9)$

executable predicate ；=

ID axiom　；10. $R(x_{10}, Sort[x_{10}])$　　Sort；出力関数

ここに，$R(x,y)$は，"yはxをMerge Sortしたもの"，$S(x)$は，"xは昇順（降順）に並んでいる。"，$I(x,y)$は，"xに含まれる要素とyに含まれる要素は等しい。"である。

Conditional refutation

(a)

4.　1. $[(x,y),\phi]$

OBR　$\{x/x_4, y/y_4\}$

8.　$\bar{S}(y)\vee\bar{I}(x,y)\vee\boxed{R(x,y)}$　$[(x,y),\phi]$

OBR　$\{nil/y\}$

$\boxed{\bar{S}(nil)}\vee\bar{I}(x,nil)\vee\boxed{\bar{R}(x,nil)}$　$[(x,nil),\phi]$

FD

7.　$\bar{I}(x,nil)\vee\boxed{\bar{R}(x,nil)}$　$[(x,nil),\phi]$

OBR　$\{nil/x_7, nil/x\}$

$\boxed{\bar{I}(nil,nil)}\vee\boxed{\bar{R}(nil,nil)}$　$[(nil,nil),\phi]$

FD,FD

$\square$ $[(nil,nil),\phi]$

(b)におけるresolventsは，一意に決定するので省略する。

(b)

9.　1. $[(x,y),\phi]$

OBR　$\{x/x_9, y/y_9\}$

$[(x,y),\phi]$

S

4.　$[(x,y), x\neq nil]$

OBR　$\{cons[car(x),cdr(x)]/x_4, y/y_4\}$

6.　$[(x,y), x\neq nil]$

OBR　$\{merge[x_6,y_6]/y\}$

2.　$[(x, merge[x_6,y_6]), x\neq nil]$

OBR　$\{y_6/y_2\}$

10.　$[(x, merge[x_6,y_6]), x\neq nil]$

OBR　$\{x_2/x_{10}, Sort[x_2]/y_6\}$

$[(x, merge[x_6, Sort[x_2]]), x\neq nil]$

FD, FD, FD

5.　$[(x, merge[x_6, Sort[x_2]]), x\neq nil]$

OBR　$\{car(x)/u_5, cdr(x)/x_5, car(x)/x_6, Sort(x_2)/y_5\}$

3.　$[(x, merge[car(x), Sort[x_2]]), x\neq nil]$

OBR　$\{cdr(x)/x_3, Sort[x_2]/y_3\}$

10.　$[(x, merge[car(x), Sort[x_2]]), x\neq nil]$

OBR　$\{cdr(x)/x_{10}, cdr(x)/x_2\}$

$[(x, merge[car(x), Sort[cdr(x)]]), x\neq nil]$

FD, FD, FD, FD

$\square$ $[(x, merge[car(x), Sort[cdr(x)]]), x\neq nil]$

Recursive定義系

$$Sort(x)=\begin{cases} nil & \text{if } x=nil \quad \text{(a)より} \\ merge[car(x), Sort(cdr(x))] & \text{if } x\neq nil \quad \text{(b)より}\end{cases}$$

2. インプリメントの思想

前章でOL-resolutionとLPSの原理を述べた。本章では，実際のインプリメントについて述べる。インプリメントは，LISP言語を用いてなされたので，LISP言語の特性上clauseのorderは，左側のliteralの方が高いとした。

2.1 Tree searching

pilot 2のtree searchingは，breadth first methodとset-of-support strategy*を結合した方法を使用している。pilot 2では，～conjectureをset-of-supportとして選ぶ。すなわち，～conjectureをtop clauseとしている。これにより，pilot 1で効率の低下の原因と思われたclash**の制限が自然に行なえる。～conjectureが複数個のclauseから構成されている場合，出力変数を含むclauseのみをtop clauseとして選択している。LPSでは，出力変数に対する情報抽出が目的であるので，出力変数を含む～conjectureをtop clauseに選べば，情報抽出に十分である。なお，～conjectureに出力変数を含むclauseが複数個あれば，それぞれのclauseをtop clauseとする複数個の証明図を生成する。また，LPS procedureは，semi algorithmであり，複数個の□の導出を行なうため，control parameterを設け，証明図の段数または，□の個数の上限を決めて停止するようにしている。

2.2 S-ruleの前処理

pilot 2では，手続きを容易にするためにS-ruleを前処理として行なっている。calculus K1では，non vital clauseに対するS-ruleの適用を禁止しているが，pilot 2では，set-of-support strategyにより，resolventsがすべてvital clauseであるので，S-ruleは前処理として行なえる。[7]

2.3 Merging Right とReduction

pilot 2では，Merging Rightの処理は，first literalと同一clause内のunifiableなliteralの間で行なわれる。pilot 1では，情報抽出の観点から，unifiableなliteralが複数個あるとき，すべての組合わせについてこの処理を行なう。[7]
Reductionについても同様のことがいえる。Merging Rightの処理は，first literalとそのunifiableなunframed literal間でなされ，Reductionの処理は，first literalとそのcomplementary framed literal間でなされる。このように，2つの処理は，framed literalであることと，complementであることを除けば同一の処理とみなせる。したがって，pilot 2では，2つの処理を同一のルーチンで行なっている。（付録A，function REDUCT-MERGE参照）

---

*) set-of-support strategyとは，充足不可能な集合に対して，充足可能な集合とそれ以外の集合に分けて，充足可能な集合間でのresolutionを禁止することにより効率を向上させる戦略である。OLRにおけるset-of-support strategyの導入は，一般にcompleteであることが示されている。[8]
**) clashとは，同一のresolventsを持つ異なるdeduction treeを構成する特定のinput clauseの集合

2.4 Axiom table と OL-resolution

Ordered Binary Resolutionの処理は，center clauseのfirst literalに関してのみ行なわれ，side clauseは常にinput clauseより選ばれる。したがって，center clauseのfirst literalとcomplementaryなliteralを含むclauseを，input clauseの集合より捜し出す必要がある。このため，pilot 2では，つぎのようなtable (Axiom table) を作り，処理の効率化と手続きの容易化を計っている。

たとえば，1. $\bar{P}$, 2. $\bar{P}\vee\bar{Q}\vee P$, 3. $\bar{P}\vee Q\vee P$, 4. P のような4つのclauseが与えられたとしよう。このとき，各clauseについて，それに含まれるliteralがそれぞれfirst literalとなるようにliteralの位置を回転させ，literalの順序の異なるclauseを作り出す。たとえば，clause 2は，$\bar{P}\vee\bar{Q}\vee P$, $\bar{Q}\vee P\vee\bar{P}$, $P\vee\bar{P}\vee\bar{Q}$ のような順序の異なるclauseになる。つぎに，このようなclausesをfirst literalのliteral symbol (Pまたは$\bar{P}$)に注目し，non executable predicateとその否定ごとにまとめ図3のようなAxiom tableを得る。

実際の処理では，Axiom tableをliteral symbolとclausesとの対リストにしたassociation listにすることによりside clauseを容易に取り出せるようにした。

Axiom tableにおけるclauseのorderは入力されたものと異なるが，これらのclausesは直接center clauseとしては用いられないので，このような処理が行なえる。

pilot 2では，まずS-ruleの前処理を行ない，その後にFrame Deletion, Merging Right とReduction，Ordered Binary Resolutionの処理を繰返す。Merging Right, ReductionとOrdered Binary Resolutionの処理は，同一のcenter clauseに対して行ない，それぞれのresolventsを求め，情報のとりこぼしをなくしている。

Predicate symbol	clauses
P	$P\vee\bar{P}\vee\bar{Q}$, $P\vee\bar{P}\vee Q$, P
$\bar{P}$	$\bar{P}$, $\bar{P}\vee\bar{Q}\vee P$, $\bar{P}\vee Q\vee P$
Q	$Q\vee P\vee\bar{P}$
$\bar{Q}$	$\bar{Q}\vee P\vee\bar{P}$

[図3]

3. pilot 2のシステム概説

ここでは，データ表現，システム構造を示し，pilot 2の主なモジュールを簡単に説明する。pilot 2の主要モジュールのプログラム・リストを付録に載せておく。使用言語は，INTERLISPの中のCLISPと呼ばれるAlgol-likeな記法が可能な言語である。

3.1 データ表現

pilot 2で取り扱うデータの中心は，clausesである。入力specにおいて，formulaは，clause formに変形されているとする。clauseは，つぎのリスト形式で表現している。

(hist body inexp outexp condexp)

histは，input clauseにおいて，clause番号，axiom table中のclauseにおいて，clause番号とfirst literalのinput clauseでの位置を示す番号との対リスト，resolventにおいては，deduction treeを構成するclauseのhistと，Reduction (R), Merging Right (M), Frame deletion (F)などの操作情報との並びであり，履歴情報を示す。

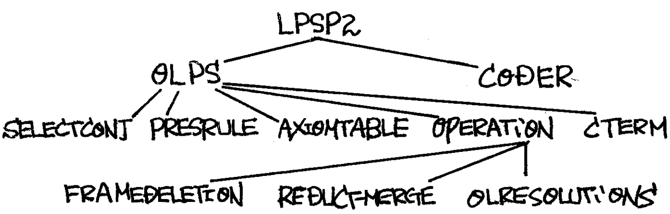
bodyは，literalの並びで，clause本体を表わす。特に□のときには，bodyをNILとする。
inexp, outexp, condexpは，それぞれvital clause中のa-exp(すなわち，β, γ, α)を示す。axiom clauseでは，inexpによってbody中の変数リストを示し，他はNILとする。literalは，肯定，否定に応じて，(P $t_1 \cdots t_n$)，および，(NEG P $t_1 \cdots t_n$)と表わされる。ここでt_iは，term，Pは，述語記号を表わす。framed literalは，literalの前に，FRAMEという記号を挿入して表わす。たとえば，

((-5 (3.1) F (R.3) (M.2)) ((NEG P X) (FRAME R (F X1))) (X) (X1) NIL)

は，resolvent $\bar{P}(x) \vee \boxed{R(F(x))}$ が，-5番のtop clauseと(3.1)のclauseとの間で，ordered-binary resolutionが行なわれ，つぎにFrame Deletionを行ない，3番目のliteralと，Reductionを行ない，最後に，2番目のliteralとMerging Rightを行なって作成されたことを示す。

3.2 システム構造とモジュール説明

pilot 2のシステム構造を示すと右図のようになる。
最上位モジュールLPSP2は，以下の入力形式で呼ばれ，Specから反証図を作り，その反証図から情報を抽出し(OLPS)，目的プログラムに変換する。(CODER)



LPSP2 [N; NEGCONJ; AXIOM; EXECPRED; NONEXEC; OUTFUNC; PRIMFUNC; IMOVAR; CONST]

ここで，Nは，control parameterであり，反証図のレベル(負整数)または，□の個数(正整数)の制限を示す。以下順に，¬conjecture，axiom (ID axiomを含む)，executable predicates，nonexecutable predicates，出力関数名，入力・中間・出力変数名，定数を表わす。

OLPS[N; AXIOM; NEGCONJ; EXECPRED; NONEXEC; IMVAR]は，axiom (AXIOM, AX)と¬conjecture (NEGCONJ, NC)とを加えたclauseの集合から，OL-resolutionを用いて□を導出し，各□のa-expから変換(CTERM[C; IMOVAR])して得られるc-termsを値とする。

CODER[OUTFUNC; C-TERMS; IMVAR; EXECPRED]は，C-TERMSから，OUTFUNCをプログラム名とする目的プログラム(ここでは，LISP)に変換する。(リスト略)

SELECTCONJ[NC; AX]は，NCのうち出力変数を含むclauseを選ぶ。(リスト略)

PRESRULE[AX; EP]は，AXに対して，EPを用いてS-ruleの前処理を行なう。(リスト略)

AXIOMTABLE[NP; APPEND[PRESRULE[AX; EP]; NC]]は，S-ruleの前処理をされたAXとNCから，axiom table (AT)を作成する。(リスト略)

OPERATION [cc; AT]は，cc (center clause)に対して，Frame Deletion (FRAMEDELETION[cc])を行ない，つぎにReductionとMerging Rightを，REDUCT-MERGE[cc]で行なって得られたresolventsと，OL-resolutionを行なって得られたresolventsを，まとめて値とする。

NEWCLAUSE[HIS; POS; FSTLIT; L; cc]は，REDUCT-MERGEにおけるunificationを行ない，新しいresolventsを作成する。OLRESOLUTIONS [cc; A]は，ccに対し，sassocで，ATから複数のside clausesを選び，各side clauseとの間で，ordered resolution (OLRESOLV[sc; cc])を行ない，複数のresolventsを作り出す。

UNIF [E1; E2; NONVAR; INEXP; OUTEXP] は, E1とE2の間のmguを作り出す。変数の判別にNONVARを, 入出力変数の識別にINEXP, OUTEXPを利用する。(リスト略)

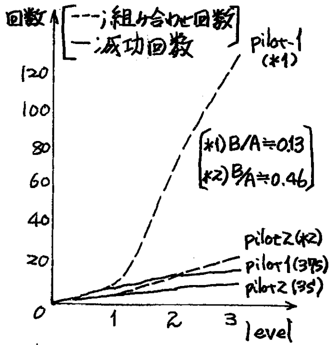
SUBSTITUTION [E; MGU] は, 式EにMGUを掛ける。(リスト略)

RENAME [SC; CC] は, SCがAxiom clauseであれば, CCとの共通変数のみを新しい変数で置き換える。

4. 評価

pilot 2は, 10段程度のレベルを持つ証明図作成を目標として開発された。これは, trivialでないプログラムの証明図には, 少なくともlinearで, 10段程度のレベルが必要であると思われるからである。

pilot 2は, 効率テストシステムとして開発されたが, 原理的には, pilot 1と同じ, calculus K_1 (+formulaの合成機能)に基づいている。また, pilot 1では, 効率化のため, S-ruleの前処理やID axiomの前処理を行なうので, Completeでないが, pilot 2は, Completeとなっている。

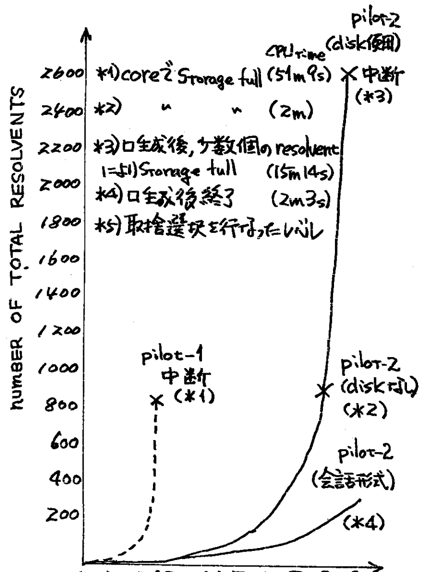


pilot1,2によるresolutionの組み合わせ回数(A)と成功回数(B)の比較(例題は付録B 例4)

[図4]

pilot 1では, 効率を低下させる原因として同一clashのresolventsを, 重複して生成していることが確認され, 簡単な手続きにより, 完全ではないが, 相当数の重複を禁止できた。しかし, breadth-first methodに基づく, 基本的な証明機構しか持っていなかったので, 最大5段程度の証明図作成が限度であった。

pilot 2は, OL-resolutionと各種の改良(3章参照)により, pilot 1では, 不可能であった, 10~15段程度の証明図が作成でき, 目標を達成した。(図4にpilot 1との効率比較を示す。また, 付録Bに7~15段の証明ができた例題を示す。)



pilot1,2の累積reslvent数による効率比較(例はMSORT)

[図5]

pilot 2を用いた実験の結果, 1) 証明図の段数が多くなるとCORE*だけでは, pilot 1に較べ, 短時間で, Storage Full (S.F.)を生じた。2) COREの補助メモリとして, disk**を利用しても, 1,2段進む程度で, 効果は少なかった。3) OL-resolutionを用いても, 無駄なresolventsが多数生成され, S.F.の原因となっている。(その多くは同一clauseが何回も適用されて, 生成されている。以下, この多重適用をiterationと呼ぶ) 4) 会話形式で, iterationが原因の無駄なresolventsを消去した結果, 効果があった。それも低レベルで行なう程, 効果が大きい。(1, 2, 4は図5参照)

これらのことから, より深い証明図の作成における

---

*) DEC system-20上の仮想記憶256KW (36 bits)から, INTERLISP自身(常駐162KW)を除いた容量以下。

**) 最大200MB (=39MW), 常時の利用可能容量は5MW前後。

効率対策上の問題としてclashの組み合わせ的な増加に較べ，iterationは，無限倍にも増える可能性があり，iterationを防ぐことの方が，より重要な問題と考えられる。pilot 2に用いたOL-resolution，およびunit-resolutionによるTheorem proverの使用経験から，従来のresolution-refutationのrefinementの方法には，効率上不充分であり，新しい方法が必要であることが認識された。

5. おわりに

pilot 2は，効率テストシステムとして，DECsystem-20上のINTERLISPを用いて作成された。pilot 2の基本的な設計と中心部のimplementは謝が行ない，学生，琴野，大村，東本，馬渡が，s-ruleの前処理，CODERの追加，そしてdiskの利用，会話等の機能を加えた。作成されたプログラムの大きさは，CODERを除く部分が，約500行で，pilot 1が，同規模のCODERを除く部分が，約2000行であったのに較べ，極めてコンパクトになった。

pilot 2は，最大15段程度のレベルを持つ証明図の作成が可能であり，目標(10段)を達成している。また，効率対策上の問題点として，iteration対策の重要さが，実験の結果，認識された。

謝辞

日頃，御指導いただく，京都産業大学計算機科学研究所の上村義明教授に感謝します。

参考文献

1. 謝(1977)：Foundation of Logical Program Synthesis，ソフトウェア工学研究会資料4-1
2. 謝(1977)：Logical Basis of Program Synthesis，京大数理研，講究録
3. 謝(1976)：プログラムの自動合成と定義可能性，京大数理研，講究録
4. 謝(1975)：プログラム・シンセシスのための情報抽出系，信学会 AL75-13
5. 謝(1975)：Mechanical Flowchart Synthesisについて，信学会 AL75-2
6. 謝(1975)：Resolution-Refutation法によるMechanical Program Synthesis 信学会 AL74-40
7. 琴野・大村・宮沢・謝：Logical Program Synthesis の Implementation (pilot 1)について (1978)情報処理・記号処理3-6
8. Chang & Lee (1973)：Symbolic Logic and Mechanical Theorem Proving，Academic Press
9. Nilson, N.J (1971)：Problem-Solving Methods in Artificial Intelligence，McGraw-Hill
10. Teitleman, W. (1974)：INTERLISP Reference manual，Xerox

付録 A　　Pilot2 プログラムリスト

```
(LPSP2
  [LAMBDA (N NEGCONJ AXIOM EXECPRED NONEXEC OUTFUNC PRIMFUNC IMOVAR CONST)
    (PROG (NONVAR C-TERMS PROGRAM)
          (RECORD CLAUSE (HIST BODY INEXP OUTEXP CONDEXP))
          (NONVAR _(APPEND EXECPRED NONEXEC OUTFUNC PRIMFUNC CONST))
          (C-TERMS _(OLPS N AXIOM NEGCONJ EXECPRED NONEXEC IMOVAR))
          (PROGRAM _(CODER OUTFUNC C-TERMS IMOVAR EXECPRED))
          (RETURN PROGRAM])

(OLPS
  [LAMBDA (N AX NC EP NP IMOVAR)
    (PROG (X2 QNC QAX AT CCS RS CTS (LVLCNT 0)
              (NILCNT 0))
          (X2_(SELECTCONJ NC AX))
          (QNC_(CAR X2))
          (QAX_(CADR X2))
          (AT_(AXIOMTABLE NP (APPEND (PRESRULE QAX EP)
                                     QNC)))
          [CCS_(for C in QNC collect (CONS (LIST (CAR C))
                                           (CDR C]
          (repeatuntil (NULL RS) OR (NCHECK N LVLCNT NILCNT)
             do (LVLCNT_LVLCNT+1)
                (RS_NIL)
                (for CC in CCS do (X2_(OPERATION CC AT))
                                  (if (CAR X2)
                                      then (RS_(APPEND RS (CAR X2)))
                                    elseif (CADR X2)
                                      then (CTS_(APPEND CTS
                                                        (CTERM (CADR X2)
                                                               IMOVAR)))
                                           (NILCNT_NILCNT+1)))
                 (CCS_RS))
           (RETURN CTS])
 (OPERATION
   [LAMBDA (CC AT)
     (PROG (QC)
           (QC_(FRAMEDELETION CC))
           (if QC:BODY=NIL
               then (RETURN (LIST NIL QC))
             else (RETURN (LIST (APPEND (DEDUCT-MERGE QC)
                                        (OLRESOLUTIONS QC AT))
                                NIL])

 (FRAMEDELETION
   [LAMBDA (CC)
     (PROG (QC)
           (QC _ CC)
           (while (FRAMELITERALP (CAR QC:BODY))
              do (QC:HIST _(APPEND QC:HIST (CONS 'F)))
                 (QC:BODY _(CDR QC:BODY)))
           (RETURN QC])

 (REDUCT-MERGE
   [LAMBDA (CC)
     (for L in (CDR CC:BODY) as POS from 2 to (LENGTH CC:BODY)
        collect (if (FRAMELITERALP L)
                    then (NEWCLAUSE 'R POS (CAR CC:BODY)
                                    (NEGLITERAL (CDR L))
                                    CC)
                  else (NEWCLAUSE 'M POS (CAR CC:BODY)
                                  L CC])

 (NEWCLAUSE
   [LAMBDA (HIS POS FSTLIT L CC)
     (PROG (MGU)
           (if (MGU_(UNIF FSTLIT L NV CC:INEXP CC:OUTEXP))='NO
               then (RETURN NIL)
             else (RETURN (CONS (APPEND CC:HIST (CONS (CONS HIS POS)))
                                (SUBSTITUTION (LIST (CDR CC:BODY)
                                                    CC:INEXP CC:OUTEXP
                                                    CC:CONDEXP)
                                              MGU])
```

```
(OLRESOLUTIONS
  [LAMBDA (CC AT)
    (for SC in (CADR (SASSOC (NEGPRED (CAR CC:BODY))
                                      AT))
       collect (OLRESOLV SC CC])

(OLRESOLV
  [LAMBDA (SC CC)
    (PROG (MGU QSC)
          (QSC_(RENAME SC CC))
          (MGU_(UNIF (CAR QSC:BODY)
                     (NEGLITERAL (CAR CC:BODY))
                     NV CC:INEXP CC:OUTEXP))
          (if MGU='NO
              then (RETURN NIL)
            else (RETURN
                   (CONS (APPEND CC:HIST QSC:HIST)
                         (SUBSTITUTION
                           (LIST (APPEND (CDR QSC:BODY)
                                         (CONS (CONS 'FRAME (CAR CC:BODY)))
                                         (CDR CC:BODY))
                                 CC:INEXP CC:OUTEXP (APPEND CC:CONDEXP
                                                            QSC:CONDEXP))
                           MGU])

(NCHECK
  [LAMBDA (N LVLCNT NILCNT)
    (if (MINUSP N)
        then LVLCNT+1 GT (MINUS N)
      elseif N=0
        then NIL
      else NILCNT+1 GT N])

(FRAMELITERALP
  [LAMBDA (L)
    (if (CAR L)='FRAME
        then T
      else NIL])

(NEGLITERAL
  [LAMBDA (L)
    (if (CAR L)='NEG
        then (CDR L)
      else (CONS NEG L])

(NEGPRED
  [LAMBDA (L)
    (if (CAR L)='NEG
        then (CADR L)
      else (CONS NEG (CAR L])
```

付録 B　Specの例

(ℓ : Pilot2によって構成された最大反証図のレベル数)

(1)　ℓ=13

```
~Conjecture : {-FIBP(x,y)}
Axioms : {FIBP(0,1),FIBP(1,1),
         -FIBP(sub1(s),t)v-FIBP(s,u)v
          FIBP(add1(s),plus(t,u))}
Input variables ........... x
Output variables .......... y
```

(フィボナッチ関数)

(2)　ℓ=7

```
~Conjecture : {-P(x,y,z)}
Axioms : {P(0,i1,add1(i1)),
       P(i2,0,j2)v-P(sub1(i2),1,j2),
       i3=0vj3=0v-P(i3,sub1(j3),l3)v
     -P(sub1(i3),l3,k3)vP(i3,j3,k3)}
Executable predicates ..... =
Input variables ........... x,y
Output variables .......... z
```

(アッカーマン関数)

(3)　ℓ=7

```
~Conjecture : {PvQ}
Axioms :{-PvQ,Pv-Q,-Pv-Q}
```

([8] p73 EX.5.4 より)

(4)　ℓ=3

```
~Conjecture : {P(x,z),-P(y,w)}
Axioms : {-P(quote(a),i)vP(quote(b),f(i)),
    -P(quote(b),j)v-q(j)vP(quote(c),g(j)),
    -P(quote(b),k)vq(k)vP(quote(c),h(k))}
Executable predicates ..... q
Input variables ........... x,y,z
Output variables .......... w
```

([8] p248 EX.11.10 より)

(5)　ℓ=15

```
~Conjecture : {-B(x)v-C(x),-B(y)v-D(y)}
Axioms : {-A(x)vF(x)vG(f(x)),
         -F(x)vB(x),-F(x)vC(x)
         -G(x)vB(x),-G(x)vD(x)
          A(j(x))vF(k(x))}
Executable predicates ..... a
Output variables .......... x,y
```

([9] p219 より)