第５５回数学教育実践研究会講演資料

人工知能研究における推論について

林　　雄一郎

1　はじめに

数学の定理を機械的に証明するということは、Leibnitz以来の数学者の夢であった。数学者のD.HilbertやB.Russell、Brouwer,K.Godelらによる数理論理学、数学基礎論など形式体系が研究され、1940年代に入ってコンピュータが登場するとその研究が加速されていった。TuringによるTuring機械を使った計算可能性の理論やKleeneらによる帰納的関数論など目を見張るものがあった。1954年、M.Davisは加算のみからなる形式体系での定理証明システムPresburger proverを作成した。1957年、A.Newell、J.C.Shaw、H.SimonらはLogical Theorem Machineという推論システムを開発して、Principia Mathematica(PMと呼ぶ、RussellとWhitehead）に掲載されている命題論理の定理38個を証明した。1958～60年にはH.WangがGentzen－Herbrandの方法を用いてPMに掲載された命題論理、等号つき一階述語論理の定理を証明した(2)。同年、DavisとPutnumはHerbrandの定理を変形したD－P手続きを開発したが難点があった。Prawitzは、代入法を提案し改良を図った。1963年、Davisは両者の利点を統合する方法を提案する。さらにこれを総合的に改良したのがJ.A.Robinsonのresolution principle(導出原理,融合原理などと訳される。以下導出原理と呼ぶ）であった。この魅力的な分野は大学教育の広範な領域のカリキュラムに取り入れられている。

論理情報科学特論・・導出原理、Herbrandの定理、定理自動証明、論理プログラミング、計算モデル、帰納法スキーマ

人工知能基礎論　・・命題論理、述語論理、完全性と健全性、導出定理、論理プログラミング

生命情報表現論　・・論理のシンタックス、セマンテックス、充足可能性、ホーン節、導出

本稿では、導出原理と推論用言語$\mu planner$、prologについて解説する。

2 導出原理の概要

導出原理は、一階述語論理のための単純で強力な演繹手法である。J.A.Robinsonの1965年の論文(4)は自動定理証明研究にbreak throughを与えるとともに1970年以降に発展を遂げた論理プログラミングパラダイムの礎となった論文である。まず対象とする論理式はすべて自由変数をもたない（自由変数が含まれると解釈が多義となるため真偽が確定しない）閉論理式とする。これを冠頭標準形に同値変形し、かつ量化子を含まない部分を連言標準形に同値変形する。

例えば、$\forall x(A(x)\rightarrow\exists y(C(x,y)\wedge D(y)))$という閉論理式は冠頭連言標準形$\forall x\exists y((\neg A(x)\vee C(x,y))\wedge(\neg A(x)\vee D(y)))$に変形できる。さらに、∃記号の除去のためskolem化（skolem関数を導入）を行って$skolem$標準形にする。例では、yに$f(x)$を代入して$\forall x((\neg A(x)\vee C(x,f(x)))\wedge(\neg A(x)\vee D(f(x))))$とする。

次に、∀全称記号を除去し論理積で結合された各部を一つの単位と考え節（clause）と呼びそれらの集合（節集合）を考える。最初の閉論理式から節集合が得られたわけである。

前例では、$\{\neg A(x)\vee C(x,f(x)),\neg A(x)\vee D(f(x))\}$となる。$S=\{C_1,C_2,\cdots,C_n\}$という仮定の節集合から、$W$という結論の節集合が導けるか否かを調べるには、結論Wの否定をＳに付け加えて$S'=S\cup\{\neg W\}$とおく。もし、このS'が充足不可能（どんなモデルでも偽）であれば結論Wの恒真がいえる。

Godelの完全性定理によれば恒真性と証明可能性は同等となるからWが証明されたことになる。

Herbrandは、Wが恒真式（トートロジー）ならば有限回の機械的ステップでS'の充足不可能の判定手続きが終わること、またモデルとしてはエルブラン領域（Herbrand Universe）に含まれる有限個の基礎例（ground instance）でのみ示せば充分なことを示した（Herbrandの定理）。しかしながら、Herbrandの手続きをそのまま実行するのは非効率であった。

1960年、DavisとPutnamは充足不能性を判定する新たな手続き（D-P手続き）を考案した。それはトートロジーを含む節の除去、一つのリテラル（単一の述語$p(t_1,t_2,\cdots,t_n)$もしくはそれに¬が付いたもの）からなる節は除去、否定と肯定の両方がない節は公理したがって真と解釈して除去、同じリテラルの肯定と否定がある節は分割する（場合分け）、ある節を包含する節があれば除去を導入するものである。しかし、これでも基礎例の生成は抑えられない。

1960年、Prawitzは従来から行われていた連言標準形への変換を選言標準形に変換することを思い立った。その中の連言項の充足不可能性を逐次調べる。充足不能な項への共通の代入があれば全体として充足不能となる。しかし、すべての連言項を調べ上げるのは結構な手間となる弱点があった。

1965年、J.A.Robinsonは単一化（Unification）に基づく導出原理を発表した(3)。導出原理は、Gentzenのcut規則に対応した推論規則を一つだけもつ演繹体系である。cut規則とは、$A \vee B, \{\neg A\} \vee C \rightarrow B \vee C$という規則で三段論法の拡張形である。Robinsonは、この体系が完全である（節集合が充足不可能であれば必ず空節empty clauseを導くことができる）ことを示した。特に、変数（束縛変数）を含む場合には二つの項を同一にする代入（substitution）があるかどうかを決定するアルゴリズムを示した。導出の戦略には線形導出法（SL導出法）など幾つか考案されている。

１階述語論理の定理証明の研究は1960年代に精力的に行われたが、効率についての難問があった。そこで１階述語論理のsubsetであるホーン論理（Horn logic）の研究がなされた。1951年にA.Tarskiの勧めでA.Hornが導入したホーン節とは、選言項に否定のないリテラル(正リテラル、単一の述語）が高々１つのものをいう。例えば、$A \vee (\neg B) \vee (\neg C) \vee (\neg D)$や$P$、$(\neg Q1) \vee (\neg Q2)$はホーン節である。１番目は$B,C,D \rightarrow A$でプログラム節、２番目は$\rightarrow P$で単位節、３番目は$Q1,Q2 \rightarrow ?$でゴール節と呼ばれる。その扱いやすさから盛んに研究され、近年論理プログラミング言語であるprologがホーン論理の記述言語となって脚光をあびた。

３ Ｌｉｓｐを用いた導出の例と手順の合成

人工知能研究で頻繁に例示されるMonkey-Banana問題をとりあげる。部屋の中に天井からbananaがぶら下がっている。部屋には椅子が一つあり一匹のサルが天井にぶら下がるbananaを見上げている。椅子に乗って手の届く高さである。サルはどうするか？という問題である。この状況を述語論理を用いて表現する。

まず変数を導入する。t,s:部屋の状況を表すのに状態変数,s_0は初期状態を表す

x:箱のある床の位置を表す位置変数

対象定数を導入する。　c:bananaのぶら下がる直下の床の位置

次に述語を定義する。

Onbox(ｔ)　　;状態tでサルが箱に乗っている（サルが箱の上か否か）

At(box,x,t)　　;状態tで箱はxにある（箱の位置）

Hb(t)　　;状態tでサルはバナナを得たか否か

さらに、関数を定義する。Sは状態の集合、Xは位置の集合とする。

Pushbox(x,s);Ｘ×Ｓ→Ｓ　状態sで箱を位置xに押して別の状態に変化する

Climbbox(s)；Ｓ→Ｓ　　状態sで箱に登って状態変化する

Grasp(s)　；Ｓ→Ｓ　　状態sでサルがバナナをつかむ

以上を用いて、問題状況を記述する。

① $\forall x \forall s(\neg Onbox(s) \supset At(box,x,Pushbox(x,s)))$

サルが箱に乗っていなければxに押していき箱,サルの位置はxとなる

② $\forall s(Onbox((Climbbox(s)))$

箱に登ればサルは箱の上にいる

③ $\forall s(Onbox(s) \wedge At(box,c,s) \supset Hb(Grasp(s)))$

cの位置にある箱の上に乗ったサルはバナナをつかむ

④ $\forall x \forall s(At(box,x,s) \supset At(box,x,Climbbox(s)))$

xに箱が置いてあればサルは箱に登ってみる

⑤ $\exists s(\neg Onbox(s))$　　初期にはサルは箱に乗っていない

設問は、「サルはバナナを得られるか？」であるからこれを述語で表わせば$\exists s Hb(s)$の真偽を問うものである。そこで背理法で解決するためこの否定⑥を考える。

⑥　$\forall s(\neg Hb(s))$

①～⑥を冠頭選言標準形に変形すると①’～⑥’になる。

①’ $\forall x \forall s(Onbox(s) \vee At(box,x,Pushbox(x,s)))$

②’そのまま

③’ $\forall s(\neg Onbox(s) \vee (\neg At(box,c,s)) \vee Hb(Grasp(s))$

④’ $\forall x \forall s(\neg At(box,x,s) \vee At(box,x,Climbbox(s)))$

⑤’及び⑥’はそのまま

①’～⑥’を節にすると次の６つを得る。これが節集合である。

$Onbox(s), At(box,x,Pushbox(x,s))$

$Onbox(Climbbox(s))$

$\neg Onbox(s), \neg At(box,c,s), Hb(Grasp(s))$

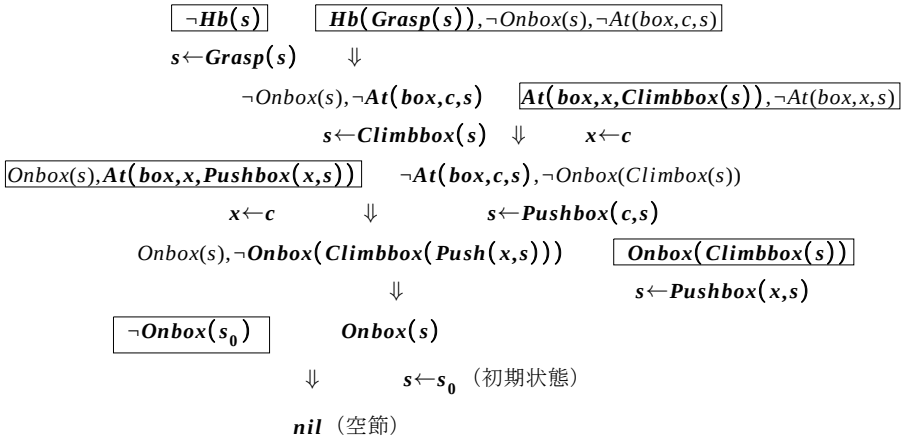
$\neg At(box,x,s), At(box,x,Climbbox(s))$

$\neg Onbox(s_0)$

$\neg Hb(s)$

なお、$\neg Onbox(s)$にs_0を代入した（*skolem*化）。

この節集合に導出を行う。そのrefutation（反駁）は例えば次のようになる。



こうして、refutationの結果、空節を得たので矛盾となる。したがって、$\exists Hb(s)$は真となる。サルはバナナが取れることになるのである。

このrefutationのunification(代入)の推移を順にたどって行くと次のようになる。$s_0 \rightarrow s_1 =$

$Pushbox[c,s_0] \rightarrow s_2 = Climbbox[Pushbox[c,s_0]] \rightarrow s_3 = Grasp[Climbbox[Pushbox[c,s_0]]]$

つまり、箱をバナナの真下に運んで(push box)、箱に登って(climb box)、最後にバナナに手を伸ばしてそれをつかむ(grasp banana)という手順である。

これは手順を合成関数で表したものであり導出原理を利用した手順の合成である。refutation過程そのものが構成的な証明であり手順の合成となるのである。

以上を、TLISPで書かれたresolutionプログラムを以前、T5600で動かし検証してみた(1).

まず節のデータ、そしてプログラムのＳ式表現を入力する。

$\mapsto CLAUSE((\star -(HB\ X)))$　節{$\neg Hb(x)$}の定義①

$\mapsto CLAUSE((-(ONBOX\ X)\ \ -(AT\ BOX\ C\ X)\ (HB\ (GRASP\ X))))$

節{¬*Onbox*(*x*),¬*At*(*box*,*c*,*x*),*Hb*(*Grasp*(*x*))}の定義②

↦*CLAUSE*(((*ONBOX* (*CLIMBBOX X*)))) 節{*Onbox*(*Climbbox*(*x*))}の定義③

↦*CLAUSE*((−(*AT BOX Y X*) (*AT BOX Y* (*CLINBBOX X*))))

節{¬*At*(*box*,*y*,*x*),*At*(*box*,*y*,*Climbbox*(*x*))}の定義④

↦*CLAUSE*(((*ONBOX X*) (*AT BOX Y* (*PUSHBOX Y X*)))) 節{*Onbox*(*x*),*At*(*box*,*y*,*Pushbox*(*y*,*x*))}の定義⑤

↦*CLAUSE*((−(*ONBOX S*0))) 節{¬*Onbox*(*s*0)}の定義⑥

↦*UPC*() resolutionプログラムの起動

↦*STOP* 入力終了

入力が終わるとLispインタープリタが動き始めプログラムを起動し始める。最初に*CLAUSE*プログラムが節のデータベースを作る。終わった所で導出*RESOLVE*開始となる。

ARGUMENTS OF RESOLVE

(3 (*ONBOX* (*CLINBBOX X*))) (⋆*T*⋆) *NIL* 1 *GIVEN*) ¬*Hb*(*x*)①と*Onbox*(*Climbbbox*(*x*))③の導出

(1 (*NIL* (*HB X*)) (⋆*T*⋆) ⋆*T*⋆ 1 *GIVEN*)

((*ONBOX* (*CLINBBOX X*)))

((*HB X*))

VALUE OF RESOLVE

FAIL 導出は失敗！

ARGUMENTS OF RESOLVE 次の組み合わせで導出を試みる

(1 (*NIL* (*HB X*)) (⋆*T*⋆) ⋆*T*⋆ 1 *GIVEN*)

(5 (((*AT BOX Y* (*PUSHBOX Y X*)) (*ONBOX X*))) (⋆*T*⋆) *NIL* 1 *GIVEN*)

NIL

((*AT BOX Y* (*PUSHBOX Y X*)) (*ONBOX X*))

((*HB X*))

VALUE OF RESOLVE

FAIL 導出は失敗！

同様にして、¬*Hb*(*x*)①と{*At*(*box*,*y*,*Pushbox*(*x*)),*Onbox*(*x*)}⑤の導出にも失敗。

次に、¬*Hb*(*x*)①と{*Hb*(*Grasp*(*x*)),¬*At*(*box*,*c*,*x*),¬*Onbox*(*x*)}②の導出を試み成功。

ARGUMENTS OF RESOLVE

((1 (*NIL* (*HB X*) (⋆*T*⋆ (4)(5)) ⋆*T*⋆ 1 *GIVEN*)

((2 ((((*HB* (*GRASP X*))) (*AT BOX C X*) (⋆*T*⋆) *NIL* 1 *GIVEN*)

((*HB* (*GRASP X*)))

((*HB X*))

ARGUMENTS OF UNIFY unificationプログラム始動

(*HB* (*GRASP X*)) (*HB Y*4)

VALUE OF UNIFY

((*Y*4 *GRASP X*)) 変数*y*4に*Grasp*(*x*)を代入

ARGUMENTS OF UNIFY ¬*Hb*(*x*)① *Hb*(*Gasp*(*x*)),¬*At*(*box*,*c*,*x*),¬*Onbox*(*x*)②

(*ONBOX S*0) ⇓

(*ONBOX J*1) ¬*At*(*box*,*c*,*x*),¬*Onbox*(*x*)⑦

VALUE OF UNIFY

((*J*1 *S*0)) 変数J1にS0を代入

VALUE OF RESOLVE

(*NIL* (*AT BOX C U*) (*ONBOX U*)) レゾルベント{¬*At*(*box*,*c*,*u*),¬*Onbox*(*u*)}⑦を得る

ARGUMENTS OF RESOLVE さらにdepth first searchで導出を行う

(7 *NIL* (*AT BOX C U*) (*ONBOX U*)) (⋆*T*⋆) ⋆*T*⋆ 2 (1.2))

(3 (*ONBOX* (*CLIMBBOX X*)) ¬*At*(*box*,*c*,*u*), ¬*Onbox*(*u*)⑦ *Onbox*(*Climbbox*(*x*))③

NIL ⇓

((*AT BOX C U*) (*ONBOX U*)) ¬*At*(*box*,*c*,*Climbbox*(*u*))⑧

ARGUMENTS OF UNIFY
(*ONBOX* (*CLIMBBBOX X*))
(*ONBOX Y*1)
VALUE OF UNIFY
((*Y*1 *CLIMBBOX X*))　　変数*Y*1に*Climbbox*(*x*)を代入
VALUE OF RESOLVE
(*NIL*(*AT BOX C* (*CLIMBBOX U*)))

こうして、レゾルベント　¬*At*(*box*,*c*,*Climbbox*(*u*))⑧を得る。
この後、⑧と③*Onbox*(*Climbbox*(*x*))、⑧と⑤*At*(*box*,*y*,*Pushbox*(*y*,*x*)),*Onbox*(*x*)と導出を試みるが失敗。次に⑧と④を試み⑨を得る。

ARGUMENTS OF RESOLVE
(8 (*NIL*(*AT BOX C* (*CLIMBBOX U*))) (⋆*T*⋆ (3)) ⋆*T*⋆ 3 (3.7))
(4 (*AT BOX Y* (*CLIMBBOX X*)) (*NIL*(*AT BOX Y X*))
ARGUMENTS OF UNIFY
(*AT BOX Y* (*CLIMBBOX X*))　　¬*At*(*box*,*c*,*Climbbox*(*u*))⑧　　*At*(*box*,*y*,*Climbbox*(*x*)),¬*At*(*box*,*y*,*x*)④
(*AT BOX C* (*CLIMBBOX Y*1))　　⇓
ARGUMENTS OF UNIFY　　¬*At*(*box*,*c*,*u*)　⑨
(*CLIMBBOX X*)
(*CLIMBBOX Y*1)
VALUE OF UNIFY
((*Y*1.*X*) (*Y*.*C*))　　*x*に*u*を代入、*y*に*c*を代入
ARGUMENTS OF UNIFY
(*AT BOX C* (*CLIMBBOX J*))
(*AT BOX C J*1)
NIL
VALUE OF UNIFY
FAIL
VALUE OF RESOLVE
(*NIL* (*AT BOX C U*)　レゾルベント⑨を得る
次に、¬*At*(*box*,*c*,*u*)⑨と¬*At*(*box*,*y*,*Pushbox*(*y*,*x*)),*Onbox*(*x*)⑤との導出を行う。
ARGUMENTS OF RESOLVE
(9 (*NIL* (*AT BOX C U*)) (⋆*T*⋆ (3)) ⋆*T*⋆ 4)8.4))
(5 (((*AT BOX Y* (*PUSHBOX Y x*)) (*ONBOX X*))) (⋆*T*⋆) *NIL* 1 *GIVEN*)
NIL
((*AT BOX Y* (*PUSHBOX Y X*)) (*ONBOX X*))　　¬*At*(*box*,*c*,*u*)⑨　　*At*(*box*,*y*,*Pushbox*(*y*,*x*)),*Onbox*(*x*)⑤
((*AT BOX C U*))　　⇓
ARGUMENTS OF UNIFY　　*Onbox*(*u*)⑩
(*AT BOX Y* (*PUSHBOX Y X*))　　*y*に*c*を代入、*y*1に*Pushbox*(*y*,*x*)を代入
(*AT BOX C Y*1))
VALUE OF RESOLVE
((*ONBOX U*))
さらに、*Onbox*(*u*)と¬*At*(*box*,*c*,*Climbbox*(*u*))⑧を試みるが失敗。
最後に、*Onbox*(*u*)と¬*Onbox*(s_0)⑥を導出して*NIL*となり証明は成功する。

ARGUMENTS OF RESOLVE
(10 ((*ONBOX U*))) (⋆*T*⋆ (8) (9))　⋆*T*⋆ 5 (9.5))

(6 (*NIL* (*ONBOX S*0)) (⋆*T*⋆) *NIL* 1 *GIVEN*)　　$\boxed{Onbox(u)}$　$\boxed{\neg Onbox(s0)}$

ARGUMENTS OF UNIFY　　⇓

(*ONBOX U*)　　*NIL*

(*ONBOX S*0)

VALUE OF UNIFY

((*U*.*S*0))　　*u*に*s*0を代入

VALUE OF RESOLVE

NIL

⋆*PROOF* ····

(3 (*ONBOX* (*CLIMBBOX X*)) *GIVEN*)

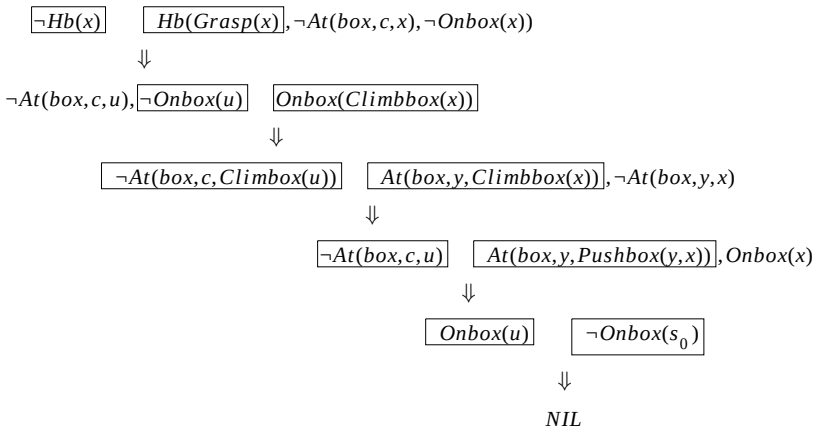
(8 −(*AT BOX C* (*CLIMBBOX U*)) *FROM* (3.7))

(9 −(*AT BOX C J*) *FROM* (8.4))

(10 (*ONBOX U*) *FROM* (9.5))

(11 *CONTRADICTION*) *FROM* (10.6)

refutation樹は次のようになる。



なお、このrefutation過程で行ったunificationのリストをストックしているのでこれから媒介変数を消去していくと先に述べたような手順の合成が可能となる。それを行うプログラム*PROGRAMGENERATOR*を作成しモンキーバナナ問題の手順を合成した。

↦*PROGRAMGENERATER*((−(*HB W*)) *W*)

↦*STOP*

FUNCTION,*ARGUMENTS*,,

PUTPROP(*MAPCAR* (*LAMBDA* (*L*) (*COND* ((*NULL L*) *NIL*)(*T* (*CONS* (*CAAR L*)

(*MAPCAR* (*CDR L*)))))) *EXR*)

PUTPROP(*PROGRAMGENERATER*(*LAMBDA* (*U OUTPUTVARIABLE*) (*PROG UNIFYLIST*1

*UNIFYLIST*2 *SUCCESFLAG GENERATEFLAG A BBB*)(*SETQ BBB OUTPUTVARIABLE*)

(*LIST OUTPUTVARIABLE*)(*SETQ UNIFYLIST*1 (*LIST UNIFYLIST*1))

(*COND* (*NULL* (*EQ* (*CLAUSE* (*CONS* (*GOUTE* ⋆) *J*)) (*QOUTE OK*))) (*GO FAIL*)))(*UPS*)

(*PRINT UNIFULIST*1)(*PRINT* (*REVERSE UNIFYLIST*1)) (*COND* ((*NULL SUCCESFLAG*)

(*GO FAIL*))) (*SETQ A* (*HHH* (*REVERSE UNIFYLIST*1)) *BBB*)) (*PRINT A*) (*RETURN A*)

FAIL (*PRINT* (*GOUTE SHIPPAI*)) (*RETURN A*))) *EXPR*)

·····

(*NIL*(*W*.*Y*1)(*Y*1.*GRASP X*1)(*X*1.*Y*1)(*Y*1.*CLIMBOX X*1)(*X*!.*Y*1)(*X*2.*C*)(*Y*1.*X*1)(*X*1.*Y*1)(*X*2.*C*)

(*Y*1.*X*1)(*X*1.*Y*1)(*X*2.*C*)(*Y*1.*PUSHBOX C X*1)(*X*1.*X*1)(*X*1.*S*0)

·····

VALUE OF HHH

(*GRASP* (*CLIMBBOX* (*PUSHBOX C S*0)))　　手順の合成がされている

```
* * * *END OF LISP JOB
JOB TIME     5MIN    547 MSEC
  1  GARBAGE COLLECTIONS
```

*μplanner*はインタープリタなので時間がかかる。これがネックとなり普及しなかった。

4　*μ*Plannerによる推論表現

*μplanner*というプログラム言語はMITのAI Lab.で人工知能研究の道具として1971年に開発された。基本構想はC.Hewittによるものでそのサブセットを G.J.Sussman, T.Winograd, E.charniakがPDP-10のＴＳＳに、またStanford大でもB.G.Baumgartがインプリメントした。さらに、MITでは改良型のCONNIVERを開発した。人工知能に関する国際会議でこの種の言語研究が発表されている。*μplanner*の基本的特徴は、問題解決に当たり解を目標として設定し(*goal oriented*)、推論を自動的に行いうることでそのために探索にバックトラッキング(backtracking)の機能やデータベースでのパターンマッチングがある。この言語を用いて2で述べたMonkey-banana問題の記述を掲載する。*μplanner*は詳細な記述が可能である事が特徴である。以下は、Stanford大[(5)]によるプログラムの実際のプロトコルである。

```
   ***MICRO-PLANNER***
>>>TOP LEVEL
LISTENING TAVAL
?(THASSERT(CLIMBABLE BOX))
 ((CLIMABLE BOX))
?(THASSERT (BOX AT A))
 ((BOX AT A))
?(THASSERT (MONKY AT B))
 ((MONKEY AT B))
?(THASSERT (BANANAS AT C))
 ((BANANAS AT C))
?(THASSERT (MONKEY OFF BOX))
 ((MONKEY OFF BOX))
?(PUTPROP  'REACH'(THCONS (XYZ)(MONKEY GETS BANANAS)
              (THASSERT(MONKEY WANTS BANANAS))
              (PRINT(QOUTE "THE MONKEY THINKS HE WANTS SOME BANANAS"))
              (THOR (THGOAL (BANANAS AT (THV XYZ)))
                    (THFAIL THEOREM (QOUTE "YES,WE HAVE NO BANANAS"))    )
              (THASSERT (MONKEY AT (THV XYZ))(THPSEUDO)(THTBF THTRUE))
              (THOR (THAND (THGOAL (MONKEY AT (THV XYZ)))
                           (THGOAL (MONKEY ON BOX))      )
                    (THFAIL THOREM (QOUTE "MONKEY DID NOT MOVE MONKEY.NOT WELL"))
              (THERASE (MONKEY WANTS BANANAS))
              (PRINT (QOUTE "MONKEY GETS BANANAS"))
              (THSUCCEED THEOREM(QOUTE SUCCESS))    )'THOREM)
 (THASSERT REACH)      プログラムＲＥＡＣＨを登録
  REACH
?(PUTPROP 'MOVEBOX'(THANTE (X Y Z)(BOX AT (THV X))
          (THGOAL (BOX AT (THV Z)))
          (THOR (THAND (EQUAL (THV X)(THV Z))  (THSUCCEED THOREM)) T)
          (THGOAL (MONKEY AT (THV Q)))
          (THOR (THOR (THGOAL (MONKEY OFF BOX))
                      (THAND (THNOT (THGOAL (MONKEY ON BOX)))
                             (THASSERT (MONKEY OFF BOX)))  )
```

```
                    (PRINT (QOUTE "MONKEY NOTICES HE IS ON THE BOX"))
                    (PRINT (QOUTE "MONKEY GETS OFF THE BOX"))))
          (THOR (EQUAL (THV Q)(THV Z))
                (THASSERT (MONKEY AT (THV Z) (THPSEUDO)(THTBF THTRUE)))
          (THERASE (BOX AT (THV Z)))
          (THASSERT (BOX AT (THV X)))
          (THERASE (MONKEY AT (THV Z)))
          (THASSERT (MONKEY AT (THV X)))
          (PRINT (LIST (QOUTE "MONKEY MOVE BOX FROM")(THV Z)(QOUTE TO)(THV X)))
          (THSUCCEED THEOREM)) 'THEOREM)
(THASSERT MOVEBOX)     プログラムｍｏｖｅｂｏｘを登録
 MOVEBOX
?(PUTPROP 'CLINB'(THANTE (X Y Z  W S Q)(MONKEY AT (THV X))
          (THGOAL (MONKEY AT (THV Q)))
          (PRINT (LIST (QOUTE "MONKEY IS AT")(THV Q)) )
          (THCOND (THOR  (THGOAL (MONKEY OFF BOX))
                    (THAND (THERASE (MONKEY ON BOX))
                           (THASSERT (MONKEY OFF BOX))
                           (PRINT (QOUTE "MONKEY NOTICES HE IS ON THE BOX"))
                           (PRINT (QOUTE "MONKEY CLIMBS OFF THE bOX")) ))
                  (THOR (THGOAL (MONKEY AT (THV X)))
                    (THAND(THERASE (MONKEY ON BOX))
                          (THASSERT (MONKEY OFF BOX))
                          (PRINT (QOUTE "MONKEY NOTICES HE IS ON THE BOX"))
                          (PRINT (QPOUT "MONKEY CLIMBS OFF THE BOX")) )  )
                  (THOR (THGOAL (MONKEY AT (THV X)))
                    (THAND (THERASE (MONKEY AT (THV Q)))
                           (THASSERT (MONKEY AT (THV X)))
                           (PRINT (LIST (QOUTE "MONKEY GOES FROM ")(THV Q)
                                        (QOUTE "TO")(THV X)))  ))
                  (THSUCCEED THEOREM (QOUTE SUCCESS))  )
          (THFAIL THEOREM (PRINT(QOUTE "WHAT MONKEY WANTS ---")))
          (PRINT (LIST (QOUTE "THE MONKEY WANTS SOME ")(THV Y)))
          (THGOAL ((THV Y) AT (YHV S)))
          (PRINT (LIST (QOUTE "MONKEY NOTICES THAT")(THV Y)(QOUTE "ARE AT")(THV S)))
          (THOR (EQUAL(THV X)(THV S))(THGO A))
          (THOR (THAND(THGOAL((THV W) AT (THV Z)))
                      (THGOAL(CLIMBABLE (THV W)))
                      (PRINT (LIST(QOUTE "MONKEY NOTICES A")(THV W)(QOUTE "AT")
                                  (THV Z))) )
                      (THFAIL THEOREM
                      (PRINT (LIST (QOUTE "ALONE IN THE WORLD WITHOUT A FIREND"))))
          (THOR (EQUAL(THV Z)(THV S))
                (THASSERT ((THV W) AT (THV S))(THPSEUDO)(THTBF THTRUE)))
          (THOR (THGOAL(MONKEY AT (THV S)))
          (THAND(THERASE(MONKEY AT (THV Q)))
                (THASSERT(MNKEY AT (THV S)))
                (PRINT(LIST(QOUTE "MONKEY QOES FROM")(THV Q)(QOUTE "TO")(THV S))) )
          (THAND(THOR(THERASE(MONKEY OFF (THV W))) T)
                (THOR(THAND(THASSERT(MONKEY ON (THV W)))
```

```
                    (PRINT(LIST(QOUTE "MONKEY CLIMBS ON")(THV W)))  )
               (PRINT(QOUTE "MONKEY ALREADY ON BOX BUT YOU KNEW THAT"))))
          (THSUCCEED THEOREM)) 'THOREM)
(THASSERT CLIMB)          プログラムＣＬＩＭＢを登録
 CLIMB
?(THGOAL (MONKEY GETS BANANAS)(THTBF THTRUE))   設問を入力
THE MONKEY THINKS HE WANTS SOME BANANAS          探索開始、以下途中経過が印字される
("MONKEY IS AT "B)
("THE MONKEY WANTS SOME "BANANAS" ARE AT "C)
("MONKEY NOTICES THAT "BANANAS" ARE AT "C)
("MONKEY NOTICES A "BOX"  AT  "A)
("MONKEY IS AT "B)
("THE MONKEY WANTS SOME "BANANAS)
("MONKEY NOTICES THAT "BANANAS" ARE AT "C)
("MONKEY GOES FROM "B TO "C)
("MONKEY MOVES BOX FROM "A TO "C)
(MONKEY CLIMBS ON BOX)
MONKEY GETS BANANAS
SUCCESS                                          結果は成功となる
```

５　論理プログラミング言語prolog

２でも言及した１階述語論理における定理証明の研究がPrologの背景にある。その手続き的意味はJ.A.Robinsonの導出原理に基づいている。1974年、エジンバラ大のR.A.Kowalski[(6)]が述語論理をプログラミング言語とみなす論理プログラミングの考えを提案した。また、この処理系をマルセイユ大のColmerauerが開発したが、さらにエジンバラ大のD.Warrenが効率の良い処理系を作った。

Prologで扱える述語の対象はHorn節（たかだか１個の正リテラルしか含まない節）の集合である。これは次式のように書ける。 ( )内はprolog文, A,A_i,Bはアトム

① $B \leftarrow A_1 \wedge A_2 \wedge \cdots \wedge A_n$　　$(B{:-}A_1,A_2,\cdots A_n)$

② $B \leftarrow .$　　$(B.)$

③ $\leftarrow A_1 \wedge A_2 \wedge \cdots \wedge A_n$　　$(:{-}A_1,A_2,\cdots,A_n)$

④ $\leftarrow .$　　$(:-)$

①はμ*planner*の*THCONS*に対応しgoal-orientedな定理文。

②は*HASSERT*に当たる表明文である。事実または公理に相当。

③は*THGOAL*の目標文に当たる。質問文。

④は矛盾を表しSTOP文に対応する。

PrologはHorn節を記述言語とし、SLD導出法（Horn節に限ったSL導出法）を推論規則としたプログラミング言語であるといえる。

簡単な以下の例題を実際にPrologに解かせてみる。

１．*human*(*turing*).　　Turingはhumanである

２．*human*(*soctates*).　　Socratesはhumanである

３．*greek*(*socrates*).　　SocratesはGreekである

４．*human*(*newton*).　　Newtonはhumanである

５．*fallible*(*X*):−*human*(*X*)　　*X*がhumanならば、*X*はfallibleである

初めにこれらをPrologデータベースに入力する。

次に、”ギリシャ人で間違えを起こしやすい人は誰か？”という問いを③型で表すと　:−*greek*(*Y*),*Fallible*(*Y*).となり、これを入力する。以下そのプロトコル(SWI-Prolog)である。

1?−[*user*].

|: *human*(*turing*).

|: *human*(*socrates*).

|: *greek*(*socrates*).

|: *human*(*newton*).

|: *fallible*(*Y*):−*human*(*Y*).

2?−*fallible*(*Y*),*greek*(*Y*).

Y=*socrates*

問いの結果がSocratesであることを演繹して返している。

prologではHorn節しか扱えないので表現できる述語に制約がある。例えば、2 で扱ったMonkey−banana問題で扱った述語$\forall x \forall s(\neg Onbox(s) \supset At(box,x,Pushbox(x,s)))$は節に変形すると{ $Onbox(s),At(box,x,Pushbox(x.s))$}と正リテラルが２つとなってHorn節でなくなる。したがって、このままの定式化ではprologを使って解けない。しかし、これは問題の定式化に原因がある。I. Bratkoは次のようなprologによる解法をあげている(7)。

今、部屋の中央にバナナが吊り下げてある。窓の傍に箱が置いてあり、サルはドアの所に立っている。この問題の定式化（表現）のため、状態写像を*state*(*X*,*Y*,*Z*,*W*)で表す。*X*はサルの水平位置、*Y*はサルの垂直位置、*Z*は箱の位置、*W*はサルがバナナを持っているか否かを表す。持っていれば*has*、いなければ*hasnot*である。

状態を変化させる関数は、①バナナを取る、②箱に乗る、③箱を押す、④は歩き回る、の４つに対応したものを考える。prolog表現は次の通りである。

①*move*(*state*(*middle*,*onbox*,*middle*,*hasnot*),*grasp*,*state*(*middle*,*onbox*,*middle*,*has*))

②*move*(*state*(*P*,*onfloor*,*P*,*H*),*climb*,*state*(*P*,*onbox*,*P*,*H*))

③*move*(*state*(*P*1,*onfloor*,*P*1,*H*),*push*(*P*1,*P*2),*state*(*P*2,*onfloor*,*P*2,*H*))

④*move*(*state*(*P*1,*onfloor*,*B*,*H*),*walk*(*P*1,*P*2),*sate*(*P*2,*onfloor*,*B*,*H*))

述語としては、サルがバナナを取れるか否かを表す*canget*(*S*)を導入する。状態*S*でサルはバナナをもっていれば真、さもなくば偽となる。サルがすでにバナナをもっている状態では

④*canget*(*state*(__,__,__,*has*))が真となる。__は*don't care*である。また、サルが状態*S*1でバナナを取れる（可能性も含む）のは、状態*S*1から状態*S*2への*M*が存在し、*S*2でバナナを取れる場合である。prolog表現は次の通りとなる。これは再帰的手続きとなる。

⑤*canget*(*S*1):−*move*(*S*1,*S*2),*canget*(*S*2).

以上、①〜⑤をprologデータベースに入力しておく。そこで、サルはバナナを取れるかという次の質問をprologの?−モードで入力する。初期状態は*state*(*atdoor*,*onfloor*,*atwindow*,*hasnot*))で表せる。

⑥*canget*(*state*(*atdoor*,*onfloor*,*atwindow*,*hasnot*)).

prologの答えは*yes*である。prologの探索木は以下のようになる。×は失敗であり、途中の試行錯誤には自動backtrackの機能がある。これは*μplanner*にもあったものである。

canget(*state*(*atdoor*,*onfloor*,*atwindow*,*hannot*))

↓→→*grasp*→→×

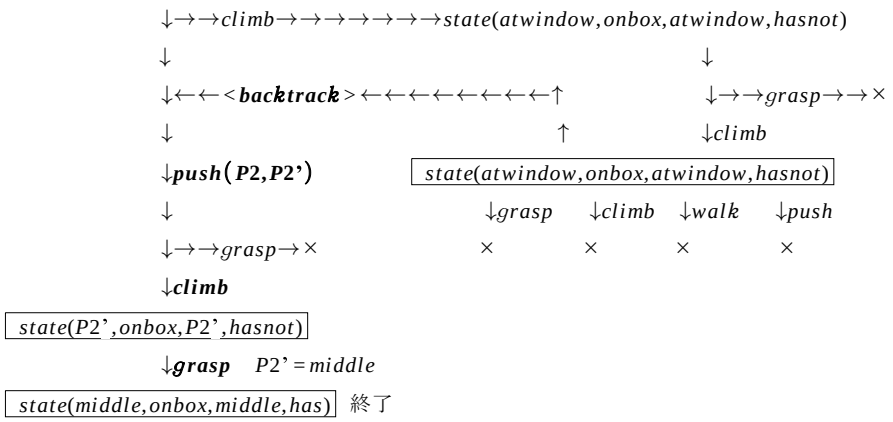
↓→→*climb*→→×

↓→→*push*→→×

↓***walk*(*atdoor*,*P*2)**

state(*P*2,*onfloor*,*atwindow*,*hasnot*)

↓→→*grasp*→→×



6　おわりに

筆者は30歳前にLispを用いた*μplanner*の日本初の開発[(4)]に携わり、導出プログラムを用いた手順合成を試みた経験がある。最近は手持ちのパソコンで動くprologでいろいろ楽しんでいる。人工知能研究の推論用言語が数多く提案されその処理系が稼動しているが、そのルーツはこれら３つの研究であるといってよい。筆者が情報処理の仕事から遠ざかって30年が経過したが、その間に何が変わったかといえば、ハードウエア技術の進歩である。ＶＬＳＩ技術の発達に伴うダウンサイジングとスピードアップ、ネットワーク技術の質的、量的な進歩は目を見張るものがある。以前無理とされた計算（数値、非数値）が現在は可能となっているものは多い。コンピュータを用いて証明された数学の問題は例えば、1976年、Appel,W.Hakenが解いたといわれる四色問題などがある。1982年から92年までの10年間官民一体となったプロジェクト第５世代コンピュータの研究が行われ、prolog拡張型言語ＧＨＣとその並列推論マシン（非ノイマン型）が開発された。実用化には今一歩であったが人工知能、意思決定支援、ＯＲ，シミュレーション、技術計算、データ処理など他領域の応用分野が期待されている。高校生たちが少しでも情報工学特に人工知能の面白さ、楽しさに気づくためにはどうすべきか。或るSSH推進校では高大連携により記号論理やprologなどの出前授業や演習を取り入れている高校がある。数学の近隣分野に対する生徒の興味関心の啓発は数学の教師の係わりが是非必要であると考えている今日この頃である。

（参考文献）

1 林雄一郎、倉持矩忠、resolutionに基づく問題解決システムとその一応用としての手順の合成について、東芝総合研究所技術報告、1973

2 H.WANG,Toward mechanical mathematics, IBM Journal,vol4,1960

3 J.A.robinson, A Machine Oriented Logic based on the Resolution Principle,JACM, vol.12,1965

4 電子技術総合研究所、Micro-Planner User’s Manual,1974

5 Stanford版,Micro-Planner Program List,1972

6 R.Kowalski, A Predicate logic as Programming Language,Proc.IFIPC,1974

7 I.Bratko,Prolog Programming for Artificial Intelligence,1986