知識工学と人工知能 35 － 2
（１９８４．７．１２）

拡張論理型言語に基づく知的情報処理

富樫敦[†]　水野康尚[††]　野口正一[†]
（†：東北大学・通研　††：電電公社・横須賀通研）

1．はじめに

Prolog[(1)]は，柔軟かつ総合的な立場からの知的処理を行なうにはまだいくつかの問題点がある[(6,7)]．例えば，従来の枠組では述語の性質を記述し活用するのに不十分で，等式によるデータ・タイプの代数的仕様記述法[(3)]などに見られるようなデータ抽象化の概念を言語の中に自然に導入することが比較的困難である．また，本来論理型言語と関数又は等式を基盤にした等式型言語は融合の可能性があるだけでなく，融合する方向にいかなければならないが，現在のところ明らかにされていない．

本論文では，以上Prologで不備な点やまだ明らかにされていない点を踏まえ，知的な情報処理，例えば定理証明，問題解決，プログラムの正当性検証，プログラム変換などを処理対象とする新しい型の拡張論理型プログラミング言語を提案し，以上の問題をどのように扱うか並びに言語の処理系を主体としたプログラミングシステムの概要について述べる．本論文で提案する論理型言語は，従来から提唱されている論理型言語Prologを２つの観点から拡張した言語になっている．１つは，求めるべき問題を記述したゴールを負節から妥当式を求めることができるようにクラスタ論理式まで拡張したことである．もう一点は，プログラムの構成要素である確定節を，分配法則や結合法則などが表現できるようにクラスタ式まで拡張した点である[(7)]．

２章で，この拡張論理型言語の概要について述べる．本言語に基づく知的処理として，３章では，定理証明，プログラム変換や問題解決を取り挙げ言語の有用性を与える．４章では，前章までの考察を背景とした総合的な論理型プログラムシステムの概観について述べる．最後に５章では，本論文のまとめと処理効率向上の観点から種々の制御構造について触れ，今後の課題を検討する．

2．拡張論理型プログラミング言語

本言語におけるプログラムは，簡約規則の集合から成る．簡約規則は，記号“：－”を区切り記号とする次の形式を持つ式である．

$$A_1, \cdots, A_m :- B_1, \cdots, B_n$$

ここで，A_i，B_jは述語である．規則について，区切り記号の左辺は必ず１個以上の述語から成らなければならないが，右辺は０個でもよい．左辺，右辺の述語は論理積(AND)で結ばれる．規則ψに出現する変数は次の条件を満足するように“全称変数”と“存在変数”の２つに分割されている：　ULV(ψ)，ELV(ψ)（URV(ψ)，ERV(ψ)）でそれぞれψの左辺（右辺）の全称変数，存在変数の集合を表すと

$$URV(\psi) \subset ULV(\psi).$$

規則$A_1, \cdots, A_m :- B_1, \cdots, B_n$を論理式として見なした場合，次のように解釈する：　式ψの全称変数を$x_1, \cdots, x_k$，右辺及び左辺の存在変数をそれぞれ$y_1, \cdots, y_r$と$z_1, \cdots, z_l$とすると，これは“全ての$x_1, \cdots, x_k$について，B_1かつ…かつB_nを満たす$y_1, \cdots, y_r$が存在することから，A_1かつ…かつA_mを満たす$z_1, \cdots, z_l$が必ず存在する”ということを意味する．Prologにおける確定節[(4)]との関係については，左辺に存在変数が出現せず，かつ左辺がただ１つの述語から成る規則が確定節である．

プログラムに対して，求めようとする問題を記述した論理式が次のゴール（問い合せ文，目標）であり，プログラムを適用することによってその解が求められる．ゴ

ールは，述語C_i から成る系列C_1，…，C_k である．規則同様，ゴールについても出現する変数には全称記号∀と存在記号∃によって束縛された“固定変数”と“推論変数”の2つの種類があり，ゴールGに出現する変数の集合Var(G) は固定変数の集合FIX(G) と推論変数の集合INF(G) に分割されている．

$$Var(G) = FIX(G) \cup INF(G).$$

ゴール中に推論変数が出現しない場合，このゴールは出現する全ての固定変数について，各述語が真となるかどうかを尋ねる問い合せである．この場合，答として“成功”か“失敗”が返される．各述語を同時に満たす解を求めたい場合には，求めたい項を推論変数で代用させることによって求めることができる．ここで定義したゴールは，次のように解釈される： ゴールC_1，…，C_k の固定変数をx_1，…，x_j，推論変数をy_1，…，y_i とすると，これは“全てのx_1，…，x_j について，C_1 かつ…かつC_k を満たすy_1，…，y_i が存在する”ことを意味する．ゴールの解は，これにプログラムを適用して次々に簡単なサブゴールに簡約する過程を経ることによって求められる．サブゴールが空の場合，計算が成功終了する．

簡約規則$\psi = A_1, \cdots, A_m :- B_1, \cdots, B_n \in \mathcal{L}$がゴールM＝$C_1$，…，$C_k$ （ m≦k ）に適用可能であるとは，A_1，…，A_m の全称変数に対するある照合置換θ，とMの推論変数に対する推論置換ηが存在して

$$(C_1, \cdots, C_m)\eta = (A_1, \cdots, A_m)\theta.$$

このとき規則ψがMに適用され，Mを

$$N = (B_1, \cdots, B_n)\theta, (C_{m+1}, \cdots, C_k)\eta$$

のゴールNに簡約する．Nの推論変数はψの右辺の存在変数，右辺の代入されない全称変数，及びC_{m+1}，…，C_k の代入されない推論変数であり，それ以外は全て固定変数．一般性を失うことなく，ψとMの間に共通の変数があった場合には，それらを異なる変数に置き換える．

【例1】 1つの簡約規則

A(h(x,a), f(y,*v)) :- A(g(y,*z),x)

から成るプログラム$\mathcal{L}$を考える．簡約の対象となるゴールをM＝A(h(x,*w),*u), B(*w,x)とすると，Mは規則の適用によりゴールN＝ A(g(*y,*z),x), B(a,x)に簡約される（図1参照）．ここで，存在（推論）変数と全称（固定）変数を区別するために，便宜的に記号“*”を用いて，*x でxが存在（推論）変数であることを示す．

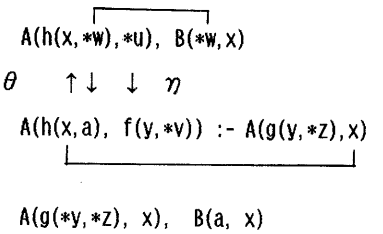

図1 簡約規則の適用 ∎

Horn節集合による入力導出も，単一化操作により，負節で表現された問題を次々に簡約するシステムと解釈できる．その違いは，本プログラムが文脈に依存して問題を簡約するのに対して，Horn節集合の方は文脈に依存しないで独立に問題を変換することである．

ゴールM，Nに関して，M$\Rightarrow_{\mathcal{L}}$N（前後から$\mathcal{L}$が明らかな場合，$\mathcal{L}$を省略することがある．以下の$\Rightarrow_{\mathcal{L}}$についても同様）でNが$\mathcal{L}$に属する規則を1回適用してMから得られることを示し，その反射推移的閉包を$\overset{*}{\Rightarrow}_{\mathcal{L}}$で表わす．M$\overset{*}{\Rightarrow}_{\mathcal{L}}$Nが成り立つとき，MはNに簡約可能であると言う．同様に，M$\overset{\zeta}{\Rightarrow}$N．但し，$\zeta$は簡約する際用いた推論置換の合成を示す．

G_0 を論理型プログラム$\mathcal{L}$に対するゴールとする．$\mathcal{L}$に関するG_0 の計算（列）は，無限列を許す簡約列

$$G_0 \Rightarrow_{\mathcal{L}}{}^{\eta 1} G_1 \Rightarrow \cdots \Rightarrow G_{i-1} \Rightarrow_{\mathcal{L}}{}^{\eta i} G_i \Rightarrow \cdots$$

である．あるG_n が停止文の場合，その計算は成功する．そのとき，推論置換の合成$\eta = \eta_1 \cdots \eta_n$ はG_0 に対する結果置換，G0 ηは計算の結果である．

3. 拡張論理型言語と知的情報処理

本章では，定理証明，プログラム変換，問題解決等に代表される知的処理を本言語の枠組の中でどのように行なうかについて述べる．これらはいずれも Prolog を拡張したことによりその扱いが簡単になったものである．

3．1 定理証明

定理証明とは，任意に与えた規則がプログラムの定理であるかどうかを確めることである．定理証明は，問い合せ文の計算機構と同じ枠組で行なうことができる．調べる規則の左辺をゴールと見なし簡約を試みる．新たに得られたゴールが規則の右辺より一般的なら成功．この場合，規則はプログラムの定理である．その他の場合は，得られたゴールを再び簡約し右辺より一般的であるかどうかを調べる．定理証明に関して以下が知られている．

【定理１】[(7)] $\mathcal{L}$をプログラム，M :- Nを規則とするとき，次の２つの条件は互いに同値である．

(1) M :- Nは$\mathcal{L}$の定理．

(2) あるゴールLが存在して，$M \overset{*}{\Rightarrow}_{\mathcal{L}} L < N$．

ここで，LがNより一般的（$L<N$）とは，$Dom(\eta) \subset INF(L)$ なる置換ηが存在して，任意の$A \in L$に対して，ある$B \in N$が存在して，$A\eta = B$． ■

3．2 プログラム変換

本節では，等式型プログラムから論理型プログラムへのプログラム変換について述べる．等式型プログラミング言語は，等式論理と呼ばれる論理体系の中で形式化されたシステムであり，述語論理に基づく論理型言語と非常に類似した特徴を備えていて，相互の変換の可能性を示唆している．プログラム変換に関して，特に等式型プログラムは，抽象データ・タイプの代数的仕様記述法や再帰的プログラム図式と関連が深く，論理型言語にデータ抽象化の概念を導入したり，効率良い計算戦略を取り入れる観点からも重要となる．

等式型プログラミング言語では，再帰的プログラム図式や抽象データ・タイプの代数的仕様記述のように，等式を用いて新たに定義しようとする関数が記述され，その実行は，項の書き換えによって定まる．ここで考える等式型プログラムの関数記号の集合Σは，

$$\Sigma = \Sigma^{C} \cup \Sigma^{d}$$

で示すように構成子の集合Σ^{C}と未定義関数記号の集合Σ^{d}の２つに分割されていると仮定する．

Σ上の等式型プログラムは，

$$F(\tau_1, \cdots, \tau_n) = \tau_{n+1}$$

の形をした等式の集合Rであり，その実行はRを項書き換えシステムとして見なし定義される．ここで，Fは未定義関数記号．構成子は処理されるデータの値を構成し，未定義関数記号の意味は書き換えによってのみ定まる．

実際の変換アルゴリズムや，その妥当性は文献(6) に譲り，ここでは具体例を挙げ等式型プログラムがどのように論理型プログラムに変換されるかを示す．

【例２】 リスト操作に関する等式型プログラム

```
Ap(nil,x) = x
Ap(i.x,y) = i.Ap(x,y)
Ap(Ap(x,y),z) = Ap(x,Ap(y,z))
```

は，次の論理型プログラム$\mathcal{L}$に変換される．

```
APPEND(nil,x,x) :-
APPEND(i.x,y,i.z) :- APPEND(x,y,z).
APPEND(x,y,*u), APPEND(*u,z,w)
        :- APPEND(y,z,*v), APPEND(x,*v,w).
```
■

例からも分かるように，このプログラム変換では，等式型プログラムから定まるデータ構造（構成的データ・タイプ）がそのままの形で論理型プログラムに持ち込まれ，共通のデータ構造を供給する．逆に言えば，関数記号の集合を構成子の集合と未定義関数記号の集合に分割することにより，等式型プログラムを自然に論理型プログラムに変換することができる．

3．3　問題解決

この節では，定理の証明手続きに基づく知的処理について述べる．知的処理として従来から人工知能の分野で扱われてきた問題解決を例に取り，与えられた知識のもとで，問題が前章の枠組の中でどの様に形式化され，解決されるかについて論じる．

簡単な積木の問題を考える．ここで考える積木の世界は，動かすことができる3つの積木A，B，Cと固定された3つのテーブルp，q，r及び片手ロボットから成る．積木の問題は，図2に示すように，初期状態として設定された積木の状態を目標状態として与えられた積木の状態に変換する問題である．この積木の状態を変えることができるのは片手ロボットで，ロボットの動作により初期状態から目標状態を達成する．簡単化のため，ロボットは，積木又はテーブルの上にある積木をテーブルの上に動かしたり，テーブルの上にある積木を積木の上に積む能力だけを有するものとする．

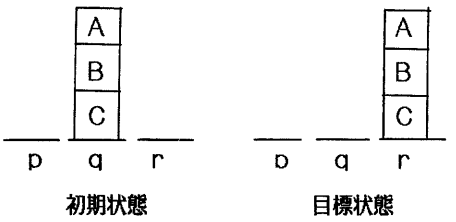

図2　積木の問題

そこで，積木の状態やロボットの動作を前章までの枠組で記述するために，以下の述語を導入する．

ONT(x,u)：　積木xはテーブルuの上に載っている．

ON(x,y)　：　積木xは積木yの上に載っている．

CLEAR(x)：　積木又はテーブルxの上には何もない．

以上のように述語を定義すると図2の問題は論理式

I = ONT(C,q), ON(B,C), ON(A,B),
CLEAR(p), CELAR(A), CLEAR(r)

で表わされた初期状態から，論理式

G= ONT(C,r), ON(B,C), ON(A,B),
CLEAR(p), CELAR(q), CLEAR(A)

で表現される目標状態を達成することである．積木の状態を変えるロボットの動作として，次の3種類があるものとする．

1)　テーブルuの上の積木xをテーブルvの上に動かす．

2)　積木yの上の積木xをテーブルuの上に置く．

3)　テーブルuの上の積木xを積木yの上に置く．

これらの動作は，作用する前に満足しなければならない前提条件と作用した後に成り立つ追加条件から成り，規則として記述すると次の様になる．

1') MOVE(x,u,v) ：
ONT(x,v), CELAR(x), CLEAR(u)
:- ONT(x,u), CELAR(x), CELAR(v)

2') TAKEOFF-PUTDOWN(x,y,u)：
ONT(x,u), CLEAR(x), CLEAR(y)
:- ON(x,y), CLEAR(x), CLEAR(u)

3') PICKUP-PUTON(x,u,y)
ON(x,y), CLEAR(x), CLEAR(u)
:- ONT(x,u), CLEAR(x), CLEAR(y)

この3つの規則の集合を$\mathcal{P}$とすると，図2で示した問題は，IとGから構成される規則G：－Iが$\mathcal{P}$の定理であるかどうかを確める問題として定式化できる．そこでは，目標状態を与えるゴールGがロボットの動作を記述した規則により次々に簡約され，初期状態を与えるIより一般的なゴールに変換された時点で終了する．この証明過程で用いた規則を逆に並べた動作系列が問題の解となる．以上述べた定理証明手続きに基づく問題解決は，与えられた問題を定理証明として形式化し解決する立場を取る．これらの問題は，PrologにおけるHorn節を用いて記述することも可能である[4]が，問題の状態を項として表現しなければならないことから記述が複雑になり，問題の記述と問題自体の関係が把握しづらい．本論文で

形式化した内容は，そのままの形でプロダクション・ルールによる問題解決[5]として捕えることができる．言い換えれば，本言語による問題の記述法は，プロダクション・ルールによる記述法と互換性があり，従来この立場で議論されてきた内容，結果を僅かの変更で本システムに取り入れることができる．

4． 論理型プログラミングシステム

本章では，2，3章で与えた考察に基づく論理型プログラミングシステム（以後ＬＰＳと記す）について述べる．今回試作したＬＰＳは，言語の処理系を主体にしたシステムであると同時に，言語の特徴を生かし，知的処理に有効な機能（定理証明，プログラム変換，基本抽象データタイプの導入）を備えた総合的なシステムとなっていて，知的処理システムを設計開発するための支援システムとして有効である．処理系のプログラムは，東北大学大型計算機センターＡＣＯＳ／６上のＥＰＩＣＳＬＩＳＰを用いて製作した．

4．1 システム構成

ＬＰＳ全体のシステム構成を図３に示す．ＬＰＳは種々のモジュールから構成され，コマンドによって対応するモジュールに制御が移行する（モジュールに制御が移った状態をそのモジュールにおけるモードと呼ぶ）．システムが起動されると自動的にコマンドモードの状態に入り，ユーザのコマンド入力を待つ．ユーザがコマンドを入力するとコマンドインタプリタが作動し，入力されたコマンドを解釈して対応する処理を実行する．また，各モードからコマンドモードに戻る場合には，コマンド”Ｃ”（エディタモードは”ＤＯＮＥ”）を入力することによってコマンドモードに戻ることができる．モード間の移動はコマンドモードを通して行わなければならない．ＬＰＳを構成するモジュールの概要を以下に示す．

a) プログラム入力モジュール：INPUT MODULE

論理型プログラム（簡約規則）を端末から入力し，

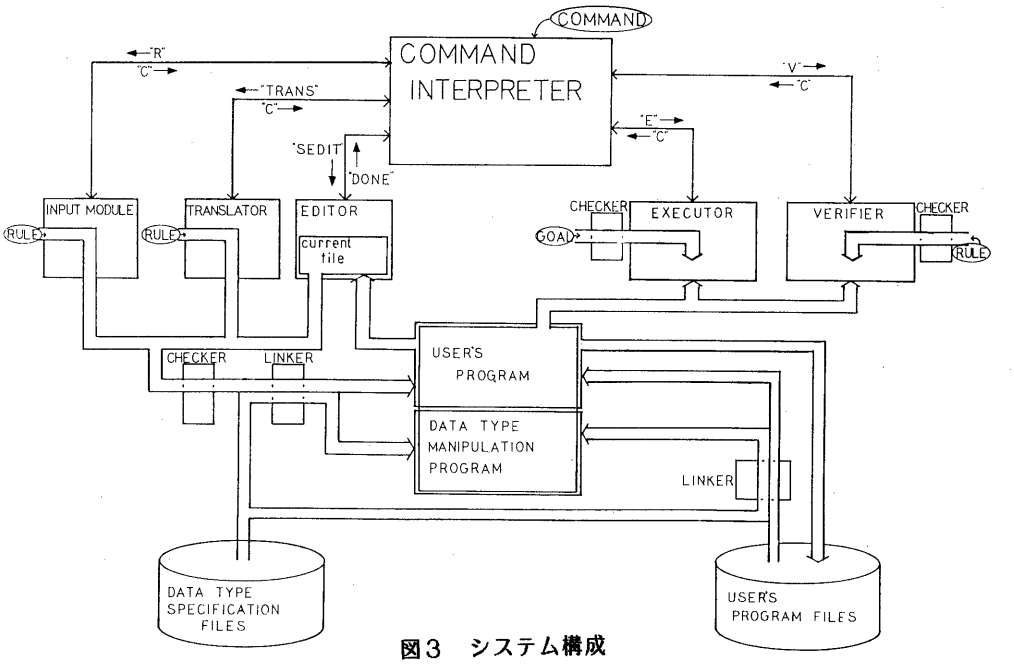

図３　システム構成

システムの内部記憶領域に格納する.

b) プログラム変換モジュール：TRANSLATOR

等式型プログラムを論理型プログラムに変換する.

c) プログラム作成，訂正モジュール：EDITOR

プログラムを作成，訂正する.

d) 計算実行モジュール：EXECUTOR

ユーザが入力した問い合せ文（GOAL）を，既にシステム内部に格納されているプログラムを用いて計算実行し，その結果を返す.

e) 定理証明モジュール：VERIFIER

定理証明を行う.

4．2　コマンド・インタプリタ

システムが起動されると”コマンドモード”の状態に入り，ユーザの入力コマンドを待つ．入力促進記号”C＝”に対して，コマンドを投入することによって対応する処理を遂行する．システムでは，現在以下のコマンドを用意している.

《モジュール起動コマンド》

モジュールを呼び出し各モードに入る.

R　：　プログラム入力モジュールを呼び出す.

E　：　計算実行モジュールを呼び出す.

V　：　定理証明モジュールを呼び出す.

TRANS　：　プログラム変換モジュールを呼び出す.

SYSE　：　ラインエディタを呼び出す.

SEDIT　：　スクリーンエディタを呼び出す

《プログラムの入出力と消去》

LOAD　：　指定したファイル中のプログラムをシステムの内部記憶領域に読み込む.

APPEND：　指定したファイル中のプログラムを内部記憶領域中のプログラムの後ろに結合する.

SAVE　：　プログラムを新しいファイルとして格納する.

RESAVE：　プログラムを元のファイルに再び格納する.

LIST　：　内部記憶のプログラムを印刷する.

CPRINT：　内部記憶のプログラムをプリンターに出力.

NEW　：　プログラムを消去する.

RECOVER：　プログラムを”復活”する.

CRUN：　コマンドファイルを実行する.

《実行戦略の選択》

BREAD：　BREADTH-FIRST 戦略を指定.

DEPTH：　BREADTH-FIRST 戦略を解除する

default 値はDEPTH-FIRST である.

《実行トレース》

TRACE：　実行仮定を追跡する.

UNTRACE：　追跡を解除する.（default）

《その他》

END　：　システムを終了させる.

TIMEOFF：　CPU TIMEを出力しない.

TIMEON：　CPU TIMEを出力する.

DECLARE：　述語の引数のタイプを宣言する.

4．3　入力モジュール

このモードに於いて，入力促進記号”R＝”に対して規則（プログラム）の入力を行ない，システムの内部に格納する（図3参照）．A_1，A_2，…，A_m：－B_1，B_2，…，B_n で表わされる規則の入力形式は，

（A_1 A_2 …A_m －　B_1 B_2 …B_n ）

の様にリストで表現し，左辺（頭部）と右辺（本体）の区切りとして”－”を用いる．ここで，各A_i，B_i は述語（原子式）であり，形式

（P　T_1 T_2 …T_k ）　P：述語名；　T_i ：引数

で表される．規則の入力後，システムはCHECKER によって次のエラーチェックを行なう．(a) シンタックスチェック　(b) 引数チェック：入力した述語がシステム内部であらかじめ決まっている述語か？；DECLARE で宣言した述語か？；ユーザが過去に入力した述語の引数の数やタイプと一致しているか？システムでは，ユーザ・プログラムの作成を容易にする目的から基本データ・タイプ

に関するデータ操作プログラムを完備している．これらのプログラムは，データ・タイプの仕様記述としてユーザに解放され，ユーザはこれらを参照して容易にプログラムを作成することができる．LINKERにより，もし述語Pがシステム内部にある基本データタイプを操作する述語であるならば，それに依存するプログラムだけを自動的にシステム・プログラム領域に記憶し，使用できる．正しく規則が入力された場合，メッセージ"STORED"が返され次の入力待ちとなる．

4．4　計算実行モジュール

このモードに於いてユーザは端末からゴールを入力し，システムはそのゴールと内部に格納してあるプログラムを適用して次々に簡単な問題に簡約し，その解を求める（図4参照）．入力促進記号

E＝　：　計算実行モード（DEPTH-FIRST ）

ET＝　：　トレース指定計算モード

EB＝　：　BREADTH-FIRST 計算モード

EBT＝　：　BREADTH-FIRST トレース指定計算モード

に対して，次の形式のゴールを入力する．

$$(-\ C_1\ C_2\ \cdots\ C_n)$$

C_i は述語で規則の左辺がない形に等しい．実行戦略として縦型に実行するDEPTH-FIRST 戦略と，横型に実行するBREADTH-FIRST 戦略を用意している．

《DEPTH-FIRST 戦略》

ゴールに対して，格納順に規則の適用を試み，成功したら適用によって新たに得られるゴールについて再び格納順に規則の適用を試みる．全ての規則の適用に失敗したら1つ前の状態に戻り（バックトラック），次の規則について確める．ゴールと規則の照合操作は，規則の左辺の最左端にある原子式を一旦固定し，照合可能な原子式をゴール中から（複数あれば左から）捜し出す．これが成功すれば，照合した原子式を取り除いたゴールと左辺に対して同様の手続きを行なう．この操作が左辺の全ての原子式について成功したら，右辺をその時点のゴールに（前から）結合した文を新たなゴールとする．

《BREADTH-FIRST 戦略》

ゴールに対して，格納順に適用可能な全ての規則を用いて新しいゴールを生成する．この場合，新たに得られるゴールは，適用可能な規則の数だけである．以上の操作を得られたゴール全てについて得られた順に再び行なう．ゴールと規則の照合はDEPTH-FIRST 戦略と同じ．

また，どちらの戦略ともユーザに対する情報として成功した規則の番号が与えられる．DEPTH-FIRST の場合バックトラックが起った時に"B" が，BREADTH-FIRST の場合最終的に成功したパスの適用規則の番号が出力される．

4．5　定理証明モジュール

入力した規則が，システム内部に格納されているプログラムの定理であるかどうかを確める．このモジュールに於いても計算実行モジュール同様，ユーザはオプションとして追跡機構や実行戦略を指定できる．それぞれに於いて次のように入力促進記号に分けられる．

V＝　：　定理証明モード

VT＝　：　トレース指定証明モード

VB＝　：　BREADTH-FIRST 定理証明モード

VBT＝　：　BREADTH-FIRST トレース指定証明モード

入力促進記号に対して入力する規則の形式は入力モジュールの規則と同じである．規則が定理であるといえればシステムは"SUCCESS" を，定理でなければ"NIL" を返す．

4．6　プログラム変換モジュール

このモードに於いては，ユーザの入力した等式型項書き換え型プログラムを論理型プログラムに変換して出力する．このモードに入いると，システムから"ADD TO PROGRAM (Y/N)?"と聞いてくるので，Yを入力すると変換されたプログラムはそのまま内部に記憶される．Nと応えると変換するのみで記憶はされない．入力は入力促進記号"LEFT=" に対して項書き換え系の左辺をS式の形で

入力し，"RIGHT="に対して右辺を入力する．終了の場合は"LEFT=" に対してCを入力する．本システムで扱う等式型プログラムに於いては，未定義関数記号（将来述語となるもの． 3.2プログラム変換参照）は”Ｐ”で始まらなければならない．

４．７　組み込み述語

本システムでは，種々の組み込み述語を用意している．組み込み述語の適用順序は，ゴールの最左端にある時のみ適用され，失敗すれば全体を失敗とする．これは左端の組み込み述語が失敗した後で将来（右側が成功した後に）その述語が成功する可能性がないとみているためである．以下代表的な組み込み述語を示す．Ｔは真，Ｆは偽を表わす．

（EVAL T V）：　LISPの関数を適用する．

(DISPLAY T)：　T の束縛されている値を印刷する．

(PATH T)　：　成功したパスの情報をT に置き換える．

(FAIL)　　：必ずＦ．

その他算述演算に関する述語等がある．

４．８　基本データ・タイプ操作プログラム

本システムでは前にも述べたように，あらかじめ基本データタイプを操作するプログラムを持っていてユーザは自由に参照できる．必要なプログラムはシステムが自動的に"LOAD"する．次にプログラムを示すが，構成子に使われている文字列は混乱を避けるため定数として使わないことが望ましい．

５．　むすび

本論文では，知的な情報処理システムを設計するという観点から拡張論理型プログラミング言語を提案し，言語の処理系と主体としたプログラミングシステム及び知的処理の具体例について述べた．既に述べたように，本言語はPrologにない有用の特徴を備えているが，その半面言語の構文がPrologと異なるためどうしても処理効率が悪くなってしまう．効率を高めるためには，何らかの制御構造を導入したり，プログラムの制御に関する知識を規則同様にプログラムの中に明確に記述してやる必要がある．プログラムの実行制御法や制御に関する知識表現として次が考えられる．

(1)　処理系に依存した制御

Prologでは，規則については上位優先，ゴールについては，左側のアトム優先

(2)　制御primitivesを用いる方法

Prologにおける“cut ”，read only annotationや入出力の表記

(3)　制御に関する知識表現

メタ述語や制御用の文法を用いる方法

詳細については，講演時に触れるつもりである．

参　考　文　献

(1) Clocksin, and Mellish :"Programmin in Prolog", Springer (1981).

(2) Gallair, and Lasserre :"Metalevel control for logic programs", in Logic Programmin, Academic Press (1982).

(3) Guttag, et. al.:" The design of data type specification", in Current Trends in Programming Methodology, Vol. 4 (1978).

(4) Kowalski :" Logic for problem solving", North-Holland (1979).

(5) Vere :"Relational production systems", Artif. Intell.,8 (1977).

(6) 富樫，野口:"等式型プログラムから論理型プログラムへの変換アルゴリズム．信学論(D),昭和59-6

(7) 富樫，水野，野口:"新しい型の論理型言語と知的情報処理", Logic Programmin Conf.'84 (1984).