

## <論 文>

# 記号論理学の教育とそれを支援する Prologインタプリタの開発について

鑰 山 徹

## 1. はじめに

記号論理学はコンピュータ科学ときわめて深い関わりがある。特に、命題論理と述語論理はコンピュータ科学を理解するためには欠かすことのできない重要な分野である。そのため、著者はゼミの中で命題論理と述語論理の具体例を取り上げ、ゼミ生に論理の基礎を学習させている。さらに、人工知能用プログラム言語Prologによるプログラミングを通じて、論理的思考の訓練を行っている。Prologは述語論理を基盤とする言語であり、それを学習することはコンピュータ動作の理解だけでなく、日常生活における「ものの考え方」を深めるのにも役立つ。

本稿では、このような記号論理学とその教育をコンピュータ科学との関連で論じる。また、著者が開発したPrologインタプリタ（CKUProlog）の機能を述べると共に、論理教育を支援するシステムのあり方を考察する。

なお、本稿では、論理を、既に定式化され利用されている命題論理と述語論理のみに限定し、時制論理や様相論理には触れない。

## 2. 記号論理学教育の必要性

およそ、論理と無縁の学問などは存在し得ないが、そこでは日本語や英語といった自然言語を用いるのが普通である。しかし、自然言語による論理では、自然言語のもつ曖昧性のため、論理自体にも曖昧性が包含されてしまう可能性がある。数学や記号論理学は、このような曖昧性を排除するために、

高度な記号化への道を選択した。記号化はややもすると無味乾燥で非人間的と捉えられがちではあるが、その有用性を捨て去ることはできない。以下では、記号論理学とその教育のあり方について、コンピュータ科学との関連で述べる。

## 2.1 コンピュータ科学と記号論理学

### 2.1.1 命題論理

コンピュータには論理回路が組み込まれており、それによって様々な論理演算・算術演算を行う。基本的には、否定回路・論理積回路・論理和回路を組み合わせることでそのような計算をすべて行っているのであり、したがって、ハードウェアの世界は命題論理（正確にはブール代数）の世界であると言える。そのため、ハードウェアを理解するためには、まず、命題論理を理解する必要がある。

命題とは、正しいか間違っているか、すなわち真か偽かが客観的に評価できる主張・事柄を言う。命題論理は、基本となる命題（基本命題）をそれ以上細分化できないアトムとして扱うものであり、論理学のうちで最も底辺に位置する基礎的な領域である。

命題論理において重要となる演算子としては、

- ・ 否定 … 「でない」を意味する。～で表す。
- ・ 論理積 … 「かつ」を意味する。∧で表す。
- ・ 論理和 … 「または」を意味する。∨で表す。
- ・ 含意 … 「ならば」を意味する。→で表す。

が挙げられる。これら演算子は、日常生活の中でもしばしば用いられるにもかかわらず、高校までの算数・数学教育では厳密な定義は与えられていない。数学Ⅰ（高校1年レベルの数学）の中に位置づけられている初等的集合論の中でも、合併集合や補集合、共通部分などを説明する際利用されるが、これ

ら演算子の意味自体は定義されずに使われているのが現状である。

そのため、コンピュータ科学を教えるためには、まず、命題論理をきちんと学習させる必要がある。

### 2.1.2 述語論理

命題論理では、現実の推論を十分に表現できない場合がある。そこで、述語論理が誕生した。述語論理では、基本命題を、述語名と項（または引数ともいう）に細分化し、述語形式

$$p(x_1, x_2, \dots, x_n)$$

で表現する。ここで、 $p$  が述語名、 $x_1, x_2, \dots, x_n$  がその項である。論理和や論理積など命題論理で用いる演算子はすべて述語論理でも用いるので、述語論理は命題論理のスーパーセットである。

プログラミングの世界では、データを保持するために変数を定義し、それを用いて計算式や条件式を記述する。まさに述語論理の世界である。プログラム理論では、プログラムの意味を述語論理で表現することもある。すなわち、ソフトウェアを理解するためには述語論理がその基礎となる。

もっとも、述語論理では変数や量化子（全称記号 $\forall$ と存在記号 $\exists$ ）が出現するため、命題論理よりはるかに複雑である。命題論理を十分理解した上で、述語論理に進まなければならない。

### 2.1.3 証明と推論

ソフトウェアを開発することは、数学において命題（定理）を証明することとよく似ている。また、プログラムが正しいことを保証することは、まさに妥当性の証明に他ならない。この証明には、演算子「含意」を中心とする推論を組み合わせて用いる。

例えば、配列に登録されている  $n$  件の数値の総和を求めるプログラムをC言語で記述すると、図1のようになる。ここでは for 文が使用されている。こ

の for 文の妥当性は、自然数の集合をドメインとする「数学的帰納法」、すなわち、

$$p(0) \wedge (\forall n)(p(n) \rightarrow p(n+1)) \rightarrow (\forall n)p(n)$$

により証明できる。

```
sum=0.0;
for (i=0;i<n;i++) sum+=x[i];
```

図1 総和を求めるCプログラム

しかし、残念ながら、証明以前に、三段論法（命題  $\alpha$  と命題  $\alpha \rightarrow \beta$  から命題  $\beta$  を導く推論）すら理解していない学生が多い。そのため、三段論法を複数用いる証明は理解不能となる。

具体的なデータを示そう。筆者は、ゼミの希望者に図2のような問題を提示して彼らの論理力を試してきた。過去6年間でちょうど100名の学生に出題したところ、何とか正解を出した学生は93名（93%）であったが、正解を出す過程を論理的に説明できた学生は19名（19%）しかいなかった。

トランプのカード52枚（ジョーカーを除く）がある。今、太郎が、そのうちの1枚をひいた。そのカードについて、以下のA～Dが成立している。

A： ハートであるか、または3の倍数である。

B： スペードではなく、6の倍数でもない。

C： 偶数ではあるが、4の倍数ではない。

D： もう1枚ひいたところ、合計が18になった。

このとき、太郎が最初にひいたカードは何か。また、その理由を論理的に説明せよ。

図2 論理の問題の例

高校で数学を勉強してきた学生の中でも、証明を苦手とする者が多い。彼らは、自分の作ったプログラムが正しいか間違っているかを判断することす

らできないのである。このような学生にプログラミングを教えるためには、推論の学習から始めなければならない。すなわち、プログラミングの教育の前に、三段論法を4～5回組み合わせる程度の簡単な証明を繰り返し学習させることによって、推論の訓練を行わせる必要がある。

このように、コンピュータ科学は論理学とは不可分である。そのため、コンピュータを教える際は、命題論理・述語論理をふまえた後にすべきである。しかし、現実には、企業や学校でプログラム開発などを教える際に、命題論理や述語論理から入ることはほとんどない。その結果、論理をきちんとは理解していない者がソフトウェアを開発することになり、バグの多い不完全なソフトウェアができあがってしまうこととなる。

## 2.2 Prolog

Prologはその名(Program in Logic)が示すとおり、述語論理を基盤とするプログラム言語である。もともとは自然言語解析のために開発された論理型言語であるが、日本の第5世代コンピュータ開発機構(ICOT)がその核言語として採用したことから、人工知能用の言語として広まった。

Prologでは、以下のような限定された特殊な形式(ホーン節という)のみを扱う。

- a)  $\forall x_1 \forall x_2 \cdots \forall x_n \quad L$
- b)  $\forall x_1 \forall x_2 \cdots \forall x_n (\sim L_1 \vee \sim L_2 \vee \cdots \vee \sim L_m \vee L)$   
あるいは  $\forall x_1 \forall x_2 \cdots \forall x_n (L_1 \wedge L_2 \wedge \cdots \wedge L_m \rightarrow L)$
- c)  $\forall x_1 \forall x_2 \cdots \forall x_n (\sim L_1 \vee \sim L_2 \vee \cdots \vee \sim L_m)$

ここで、 $x_1, x_2, \dots, x_n$ は変数であり、 $L_1, L_2, \dots, L_m$ は述語である。これらホーン節をProlog表現すると、次のようになる。

- a')  $L.$
- b')  $L :- L_1, L_2, \dots, L_m.$

c')  $?- L_1, L_2, \dots, L_m.$

a')を事実, b')を規則, c')を質問という. また, 規則において, 演算子  $:-$  の左辺を頭部, 右辺を本体という. 事実と規則でプログラムを記述し, 質問でそれを実行する. 規則を再帰的に定義することにより繰り返しが実現できる.

なお, Prologでは, 「背理法」に基づく推論を繰り返し行うことによってプログラムを実行する. ここで, 背理法とは, 証明すべき命題  $P$  を否定して推論を進めることによって矛盾を導き出し, 命題  $P$  が正しいことを示す証明法である.

Prologは限定された形式ではあるものの述語論理に基づいているため, 他のプログラム言語 (BasicやC言語など) の場合と比較して, プログラムの意味が明確である. もっとも, Prologでプログラミングを行うためには, 述語論理や背理法などを理解しておかなければならない. 逆に言えば, Prologプログラミングによって述語論理や背理法をより深く学習することができる. すなわち, Prologは論理を理解するための教育支援ツールとして用いることができる.

### 3. CKUPrologとその機能

CKUPrologとは, 著者が開発したPrologインタプリタである. Java (正確にはMicrosoft社のVisual J++) というオブジェクト指向言語で開発されており, ウィンドウズ上で動作する. 以下にCKUPrologについて述べる.

#### 3.1 CKUPrologの概要

##### 3.1.1 CKUPrologのクラス構成

CKUPrologは, 22個のクラスから構成されている. 各クラスの概要は, 表1に示すとおりである.

表1 各クラスの概要

No	ク ラ ス 名	サイズ	概 要
1	ControlTree	2KB	ユーザプログラムの実行を制御する制御表のためのクラス。
2	DataStack	2KB	入力節の構文解析を行う際に使用するスタック。解析結果を登録したり、括弧の対応をチェックするために用いる。
3	ErrorDialog	3KB	エラーを表示するためのダイアログ。
4	ErrorMessage	3KB	文法エラー、実行エラーのメッセージを登録しているクラス。
5	Execution	79KB	ユーザプログラムの実行全体を管理する。
6	ExeWindow	11KB	実行画面のためのクラス。
7	Form1	12KB	CKUProlog全体のコントローラーで、メインウィンドウを表示する。
8	HozonKakuninDialog	4KB	ユーザプログラムを保存するかどうかを確認するためのダイアログ。
9	InitWindow	4KB	初期画面のためのクラス。
10	Kaisetsu	1KB	組込み述語の解説用クラス。
11	KaisetsuBase	13KB	組込み述語の解説内容を保持するクラス。
12	KakuninDialog	4KB	デバッグモードでユーザプログラムを実行する際、その内容を表示する画面。
13	NameList	2KB	変数名を登録するクラス。変数表の一部をなす。
14	PredListup	6KB	組込み述語一覧を表示するための画面。
15	PredStructure	1KB	述語の内部表現用のクラス
16	ProgramArea	5KB	ユーザプログラムを登録するためのクラス。
17	PrologData	4KB	Prologで用いるデータ型を管理するクラス。
18	ReadDialog	4KB	入力用組込み述語readを実行する際に表示する入力用画面。
19	SpyList	2KB	組込み述語spyのための述語リスト。
20	StrBuffer	6KB	構文解析で用いる文字列バッファ。
21	TokenData	1KB	構文解析で用いるトークンを表すクラス。
22	VariableTable	3KB	変数表。

クラス間の依存関係は、図3のとおりである。すなわち、Form1クラスを中心として、その他のクラスのオブジェクトが必要の都度生成され利用される。

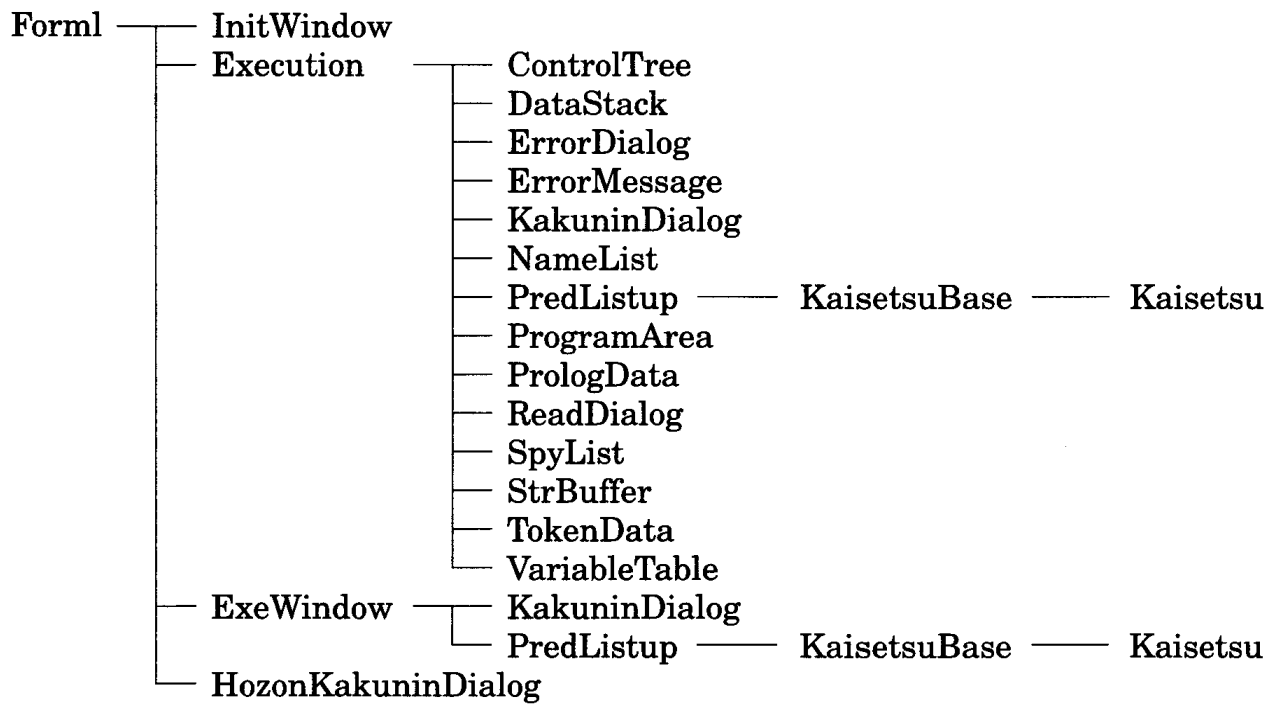


図3 クラス間の関係

### 3.1.2 CKUPrologのウィンドウと操作

#### a) 入力ウィンドウとメニュー項目

CKUPrologを起動すると、Prologのソースプログラム入力ウィンドウが現れる。このウィンドウには、図4に示すメニュー項目が付随している。

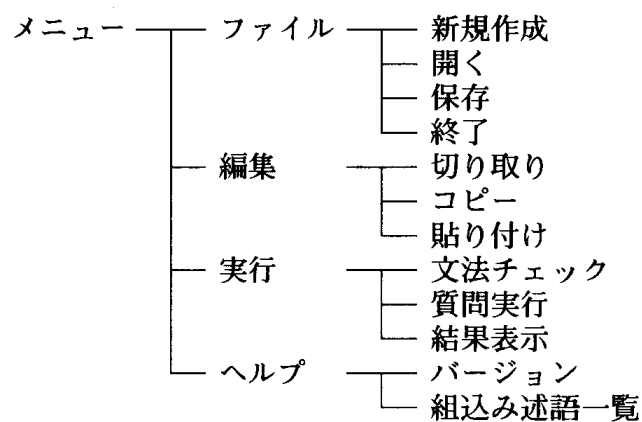


図4 メニュー項目



ユーザは、まず、このウィンドウでソースプログラムを入力する。ソースプログラムの修正に際しては、適宜、「編集」項目の「切り取り」、「コピー」、「貼り付け」を利用することができる。ソースプログラムの文法的なチェックを行うには、「実行」項目の「文法チェック」を選択すればよい。その場合、文法エラーがあれば、エラーメッセージを表示してくれる。

#### b) プログラムの実行と実行ウィンドウ

入力したPrologプログラムを実行するには、「実行」項目の「質問実行」メニューを選択する。そうすると、図5に示す実行ウィンドウが現れる。

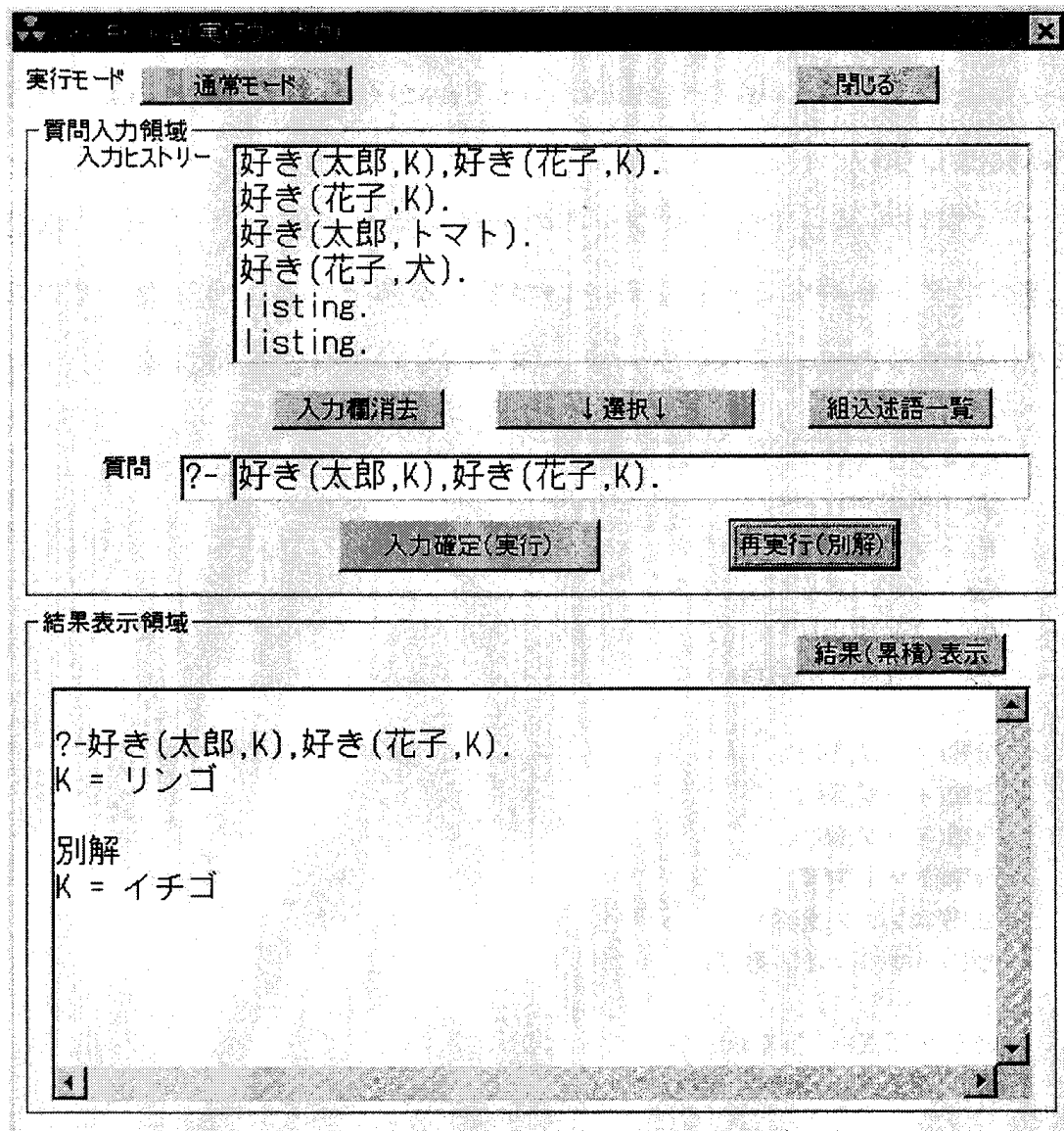


図5 実行ウィンドウ

実行ウィンドウでは、入力用エディットボックスに質問を入力すると、下部の出力領域に結果が表示される。推論結果の出力形式は、市販のPrologインタプリタと同様である。質問が成功した場合で、かつその質問に変数が含まれている場合はその変数の値を出力するが、そうでないときはyesかnoかを答える。なお、このウィンドウは入力履歴を保存しているので、入力作業を軽減できる。すなわち、過去に入力した質問を再度実行させたい場合には、その質問を入力履歴から選択するだけでよい。

Prologインタプリタは、普通、別解を求める機能を有している。別解を求めるには、論理和演算子を意味するセミコロン；を入力させるものが多いが、CKUPrologでは再実行ボタンをクリックするだけである。ただし、質問が一度も実行されていない場合や前回の質問でnoが得られた場合には、再実行ボタンは機能しない。

また、CKUPrologでは、質問とその結果をすべて保存しており、結果（累積）表示ボタンを押すことにより、いつでも参照できるようになっている。

図6に簡単なPrologプログラムを、図7にその実行例を掲げる。

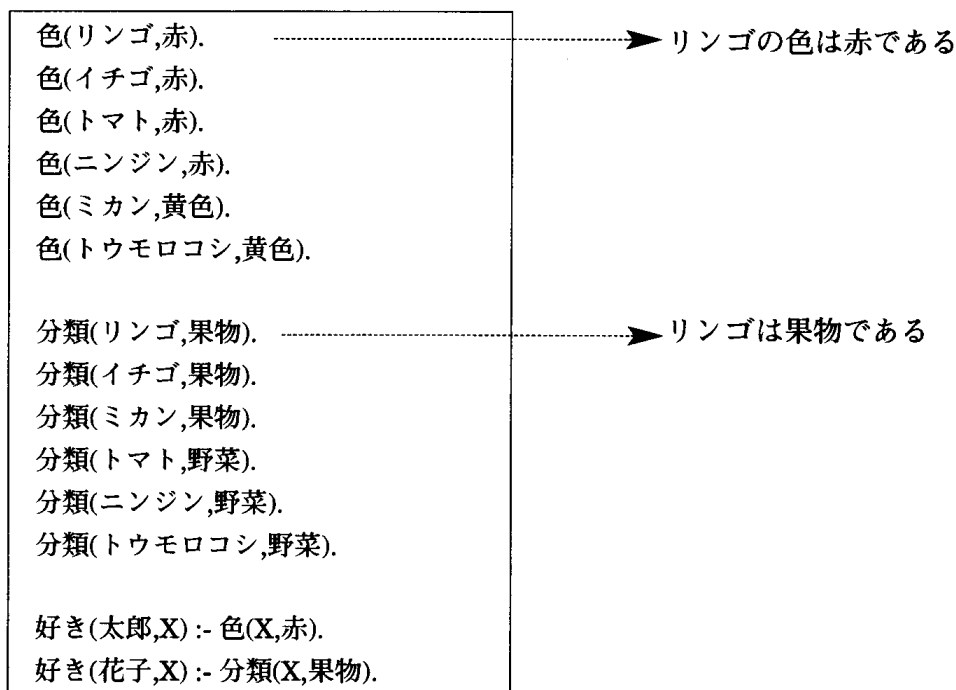


図6 簡単なPrologプログラム

<< 累積実行結果 >>	
Thu Dec 16 14:14:19 GMT+09:00 1999	
?-好き(太郎,トマト).	→ 太郎はトマトが好きか?
yes	
?-好き(花子,K).	→ 花子は何が好きか?
K = リンゴ	
別解	
K = イチゴ	
別解	
K = ミカ ン	
別解	
no	
?-好き(太郎,K),好き(花子,K).	
K = リンゴ	
別解	
K = イチゴ	
別解	
no	

図 7 実行例

## 3.2 組込み演算子と組込み述語

### 3.2.1 組込み演算子

CKUPrologは、表 2 に示す組込みの演算子を持っている。これらは、プログラムの中で自由に使用することができる。

なお、表中の記号は以下の意味を表す。

fx    …    1 項前置演算子

xfx   …    2 項中置演算子

(項内で使われる演算子の優先順位<当該演算子の優先順位)

xfy   …    2 項中置演算子で、右結合

yfx   …    2 項中置演算子で、左結合

xf    …    1 項後置演算子

表2 組込み演算子

No.	演算子	優先順位	結合性	概 要
1	<code>:-</code>	255	<code>xfx</code>	規則を表す。
2	<code>?-</code>	255	<code>fx</code>	質問を表す。
3	<code>;</code>	254	<code>xfy</code>	論理和演算子
4	<code>,</code>	253	<code>xfy</code>	論理積演算子
5	<code>.</code>	51	<code>xfy</code>	リストの先頭要素と残りのリストを連結するドット演算子
6	<code>is</code>	40	<code>xfx</code>	代入演算子 右辺の算術式を評価し、左辺の変数と単一化する。
7	<code>=..</code>	40	<code>xfx</code>	左辺は述語、右辺はそれを分解したリストを意味する。 例えば、 <code>father(hanako,taro)=..[father,hanako,taro]</code> は真である。
8	<code>=</code>	40	<code>xfx</code>	
9	<code>&lt;</code>	40	<code>xfx</code>	
10	<code>=&lt;</code>	40	<code>xfx</code>	
11	<code>&lt;=</code>	40	<code>xfx</code>	
12	<code>&gt;</code>	40	<code>xfx</code>	
13	<code>=&gt;</code>	40	<code>xfx</code>	
14	<code>&gt;=</code>	40	<code>xfx</code>	
15	<code>-</code>	31	<code>fx</code>	マイナス符号
16	<code>+</code>	31	<code>fx</code>	プラス符号
17	<code>-</code>	30	<code>yfx</code>	
18	<code>+</code>	30	<code>yfx</code>	
19	<code>*</code>	21	<code>yfx</code>	
20	<code>/</code>	21	<code>yfx</code>	

## 3.2.2 組込み述語

CKUPrologは、表3に示す組込み述語を持っている。ユーザは、組込み述語と同じ名前、項数を持つ述語を定義することはできない。

表3 組込み述語

No.	組 込 み 述 語	意 味
1	<code>arg(N,S,A)</code>	述語SのN番目の要素がAであることを意味する。 例えば、 <code>?- arg(2,father(hanako,taro),X).</code>

		を実行すると、Xはtaroとなる。
2	assert(X)	節Xをデータベースの最後尾に付加する。
3	asserta(X)	節Xをデータベースの先頭に付加する。
4	assertz(X)	節Xをデータベースの最後尾に付加する。
5	atom(A)	Aがアトムるとき成功する。
6	atomic(A)	Aがアトミック（アトムか数値）るとき成功する。
7	call(X)	節Xを実行する。
8	clause(X,Y)	頭部がX、本体がYとなる節がデータベースに存在すれば成功する。
9	dec(X,Y)	Xを1減らし、Yと単一化する。
10	display(X)	述語（または関数）Xを前置記法で表示する。 例えば、display(a+b*c)を実行すると、 +(a,*(b,c)) と表示される。
11	fail	必ず失敗する。
12	float(X)	Xが実数るとき成功する。
13	functor(S,F,N)	Sが名前Fで、項数Nである構造るとき成功する。
14	inc(X,Y)	Xを1増やし、Yと単一化する。
15	integer(X)	項Xが整数るとき成功する。
16	length(L,N)	リストLの長さがNるとき成功する。
17	listing	データベースの内容をすべて表示する。
18	listing(A)	データベースにある名前がAの節をすべて表示する。
19	name(A,L)	アトムAを構成する文字のコードリストがLるとき成功する。
20	nl	改行する。
21	nonvar(X)	Xが値を持たない変数でないとき成功する。
22	nospy(X/Y)	トレースのためのスパイポイントを解除する。
23	not(X)	否定の演算子
24	notrace	トレースを終了する。
25	number(X)	Xが数値るとき成功する。
26	read(X)	項を入力し、Xと単一化する。
27	repeat	常に成功する。
28	retract(X)	Xと単一化できる節をデータベースから削除する。
29	spy(X/Y)	トレースのためのスパイポイントを設定する。
30	tab(N)	空白をN個出力する。
31	trace	トレースを開始する。
32	true	一度だけ成功する。
33	var(X)	Xが値を持たない変数るとき成功する。
34	write(X)	項Xを出力する。
35	!	カットオペレータ

なお、CKUPrologでは、ヘルプ機能として、組込み述語や組込み演算子を参照できる機能が付加されている。これは、単に所定のボタンを押すだけで

あり、それにより図8に示す画面が表示される。

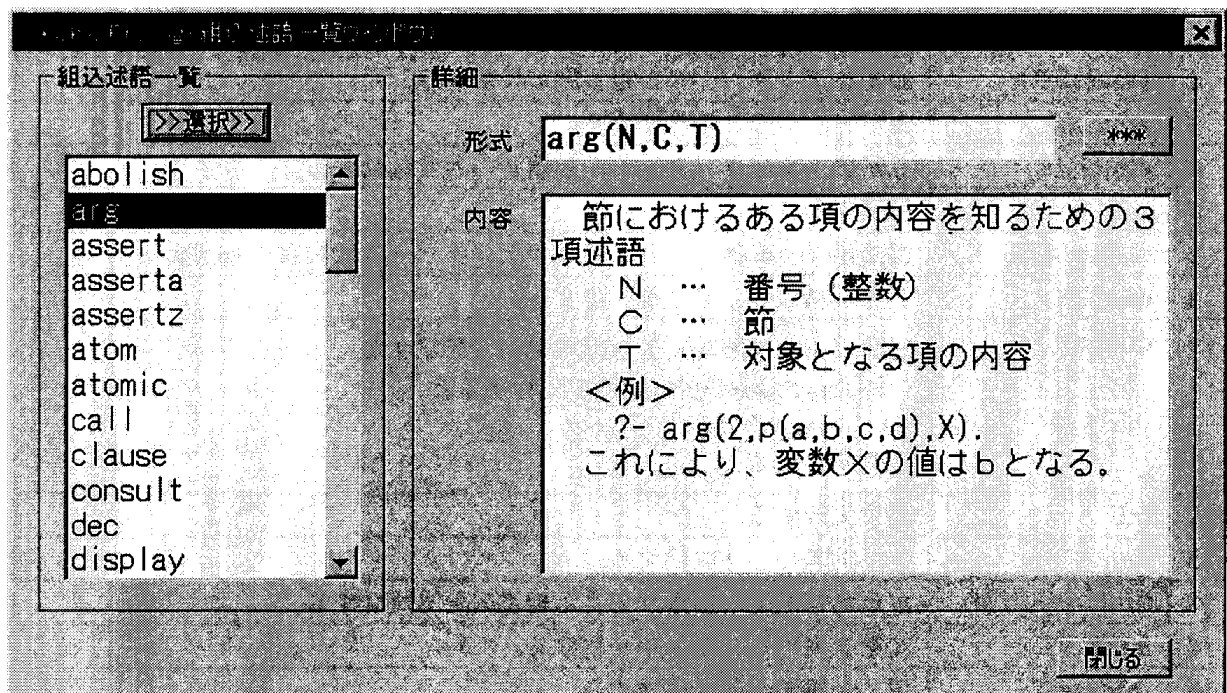


図8 組込述語一覧ウィンドウ

### 3.3 基本的なデータ構造

#### 3.3.1 データ型

CKUPrologでは、以下のような4種類のデータ型を使用できる。

- ・ 整数
- ・ 実数
- ・ 文字アトム
- ・ 関数 または 述語

これらのデータおよび変数はすべて、PrologDataクラスのオブジェクトとして実現される。関数と述語は名前（文字アトム）と複数の項からなるという同一の形式を持つが、両者は文脈により区別される。＋や－などの演算子も関数または述語として扱う。文字アトムは、項の個数が0個の述語とみなされる。

例えば、

`append([X | L],M,[X | N]) :- append(L,M,N).`

という規則を図示すると，図9の木構造となるが，その各節点がPrologDataクラスのオブジェクトである．

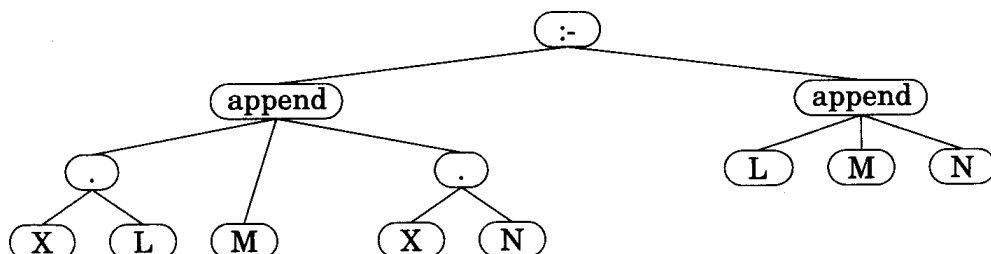


図9 規則を表す木構造

### 3.3.2 データベース

Prologにおけるデータベースとはソースプログラム登録領域であり，CKUPrologでは，ProgramAreaクラスで実現している．ユーザが定義した事実および規則はすべて，このクラスのオブジェクトに登録される．その際，述語名と項数がキーとなる．キーが同一の事実や規則はリスト構造で連結される．図10にデータベースの状態を例示する．

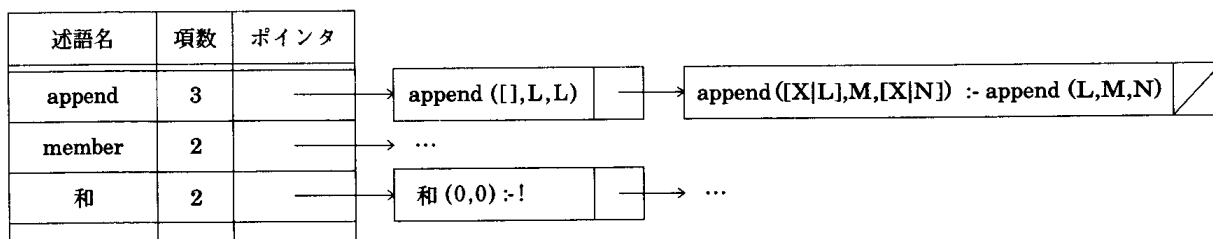


図10 データベースの構造

### 3.3.3 変数表

同一の名前を持つ変数は，同一節内では同じオブジェクトを意味するが，節が異なる場合は異なるオブジェクトを意味する．すなわち，Prologにおける変数は，節毎の局所変数である．しかも，Prologでは述語の再帰的定義に

より繰り返しを表現するため、同一の規則を何度も実行することがあり、変数名だけで変数値の登録や取り出しを行うことはできない。Prologでは、変数は後述の単一化という手続きを経て初めてその値を持つことになるが、それがプログラム実行上どの時点であるかが重要となる。

そのため、図11に示すデータ構造で変数と変数値を制御する必要がある。これは、CKUPrologでは、NameListクラスとVariableTableクラスで実現している。

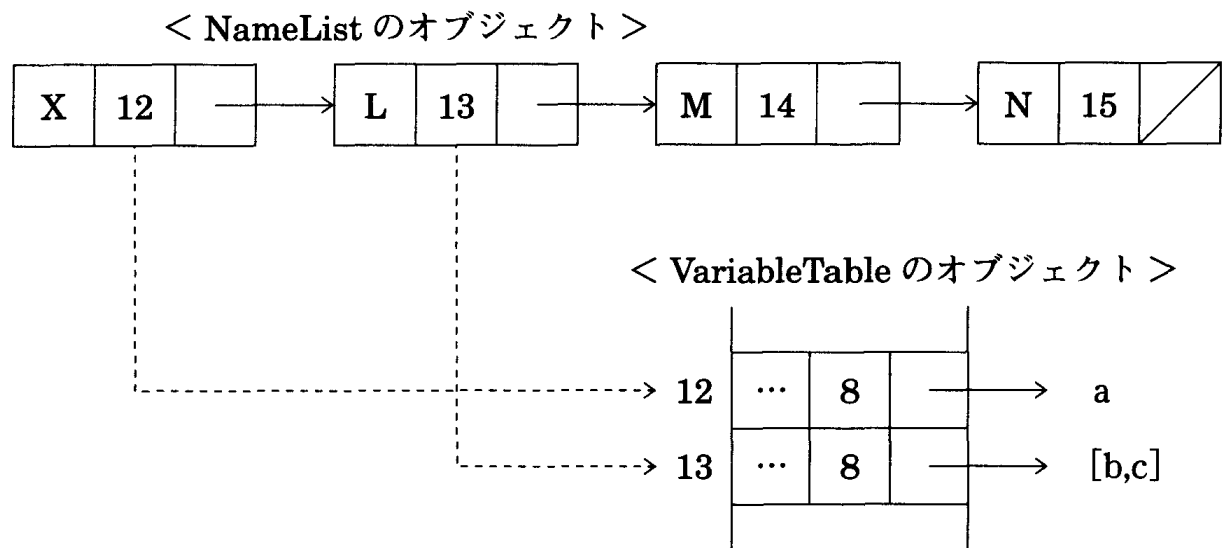


図11 変数表の例

### 3.3.4 実行制御表

Prologプログラムの実行は、入力された質問を「成功」させるような試行錯誤の繰り返しである。この実行状況は木構造で表現できる。実行制御表とはこの木構造全体を表すためのデータ構造であり、ControlTreeクラスで実現している。詳細については、3.4.2で述べる。

## 3.4 Executionクラスの概要

Prologプログラム実行のための中心的なクラスが、Executionクラスである。このクラスには、構文解析、実行制御、組込み述語処理などの重要な処理を行うメソッドが含まれており、そのサイズもCKUPrologの中では最大の2972



ステップとなっている。

以下に、このクラスの主要な機能について述べる。

### 3.4.1 ホーン節の構文解析

ユーザが作成したプログラム（事実と規則）、および入力された質問は、演算子順位文法に基づいて構文解析し、内部表現に変換する。

#### 1) トークンの表現

括弧などの各種記号、文字アトム、数値など意味のある文字の列をトークンという。CKUPrologにおけるトークンには以下のものがある：

整数、実数、文字アトム、演算子、(, ), [, ], |

各種トークンデータは、TokenDataクラスのオブジェクトとして表現される。それを構成するデータは、種類名とトークン内容だけである。

#### 2) スタック

構文解析に際しては、以下のスタックを用いる。

- ・ テンポラリスタック
- ・ 出力スタック
- ・ 括弧スタック

テンポラリスタックは取り出されたばかりのトークンを一時的に登録しておくスタックである。一方、出力スタックは解析済みのトークンを登録する。括弧スタックは、括弧の対応を調べるためのスタックである。いずれも、DataStackクラスのオブジェクトとして実現しており、TokenDataクラスのオブジェクトのみが登録対象である。

#### 3) トークンの接続関係

2つの連続するトークンの間の接続関係は表4に示すとおりである。表中、○はその接続が妥当であることを意味し、×はエラーを意味している。

表4 トークンの接続関係

直 前 \ 現 在	(   )   [   ]				演 算 子					アトム   その他		
					xf	xfx	xfy	xfx	fx			
初期状態	○	×	○	×	×	×	×	×	○	×	○	○
(	○	×	○	×	×	×	×	×	○	×	○	○
)	×	○	×	○	○	○	○	○	×	○	×	×
[	○	×	○	×	×	×	×	×	○	×	○	○
]	×	○	×	○	○	○	○	○	×	○	×	×
演算子 xf	×	○	×	○	○	○	○	○	×	○	×	×
xfx	○	×	○	×	×	×	×	×	○	×	○	○
xfy	○	×	○	×	×	×	×	×	○	×	○	○
yfx	○	×	○	×	×	×	×	×	○	×	○	○
fx	○	×	○	×	×	×	×	×	○	×	○	○
	○	×	○	×	×	×	×	×	○	×	○	○
アトム	○	○	×	○	○	○	○	○	×	○	×	×
その他	×	○	×	○	○	○	○	○	×	○	×	×

## 4) 解析アルゴリズム

CKUPrologでは、既に述べたように演算子順位文法による構文解析を行う。すなわち、表4に示すトークンの接続関係を調べながら、ホーン節を内部表現に変換する。その際、前述の3つのスタックを用いる。

Java風に記述した解析アルゴリズムの概要を図12に示す。図の中で用いられている記号の意味は以下の通りである。

- Token           : 処理すべきトークン
- PrvToken       : 直前のトークン
- Ttop           : テンポラリスタックのトップにあるトークン
- Otop           : 出力スタックのトップにあるトークン
- P1             : TTopの演算順位
- P2             : Tokenの演算順位

```

if (Tokenがアトムか数値か変数)  {Tokenを出力スタックにプッシュする}
else if (Tokenが'(') {
    if (PrvTokenがアトム)
        {出力スタックからOtopをポップし、述語名としてテンポラリスタックにプッシュする}
    else if (PrvTokenが'|' か '[' か '"')
        {Tokenをテンポラリスタックにプッシュする}
}

```

```

else if (Tokenが'|')
    do {
        テンポラリスタックからTtopをポップし、出力スタックにプッシュする。
    } until (Ttopが'|' か述語名)
else if (Tokenが'|') {Tokenをテンポラリスタックにプッシュする}
else if (Tokenが'|')
    if (PrvTokenが'|') {アトムnilを出力スタックにプッシュする}
    else
        if (Ttopが'|') {ドット演算子を出力スタックにプッシュする}
        else {アトムnilを出力スタックにプッシュする}
        while (Ttopが'|' でない)
            テンポラリスタックからTtopをポップし、出力スタックにプッシュする。

else if (Tokenが'|') {Tokenをテンポラリスタックにプッシュする}
else if (Tokenが演算子)
    if (テンポラリスタックが空) {Tokenをテンポラリスタックにプッシュする}
    else if (Ttopが演算子でない) {Tokenをテンポラリスタックにプッシュする}
    else if (P1<P2) {Tokenをテンポラリスタックにプッシュする}
    else if (P1>P2)
        do {
            テンポラリスタックからTtopをポップし、出力スタックにプッシュする。
        } until (テンポラリスタックが空 or Ttopが'|' or Ttopが'|' or Ttopの順位<P2)
    else if (TtopはTokenと等しくない)
        {テンポラリスタックからTtopをポップし、出力スタックにポップする}
    else if (その演算子の種類がxfy) {Tokenをテンポラリスタックにプッシュする}
    else if (その演算子の種類がyfx) {Tokenを出力スタックにプッシュする}

```

図12 解析アルゴリズムの概要

## 規則

`member(X,[A|Y]) :- member(X,Y).`

の場合を例に取り、その構文解析状況を図13に示す。(次頁)

### 3.4.2 実行の基本制御

Prologインタプリタは、入力された質問を「成功」へと導くよう試行錯誤を繰り返す。

質問が成功するのは、以下の3通りである。

- a) 組込み述語で、その内部処理が成功した場合
- b) ユーザ定義述語で、データベース内の事実との単一化に成功した場合
- c) ユーザ定義述語で、データベース内の規則の頭部との単一化に成功した場合

いずれでもない場合、その質問は「失敗」となる。また、c)の場合、その規則

トークン	テンポラリスタック	出カスタック
member		member
(	member ( ,	
X	member ( ,	X
,	member ( ,	X
[	member ( , [	X
A	member ( , [	X A
	member ( , [	X A
Y	member ( , [	X A Y
	member ( ,	X A Y
		X A Y , member
:		X A Y , member
:		X A Y , member member
:	member	X A Y , member
:	member	X A Y , member X
:	member	X A Y , member X
:	member	X A Y , member X Y
:		X A Y , member X Y , member
:		X A Y , member X Y , member

図13 構文解析状況



PrologData cls;      … 節内容

NameList vlist;      … 局所変数リスト

図15に実行制御表の例を掲げる。これは、図14の実行過程を表したものである。

	oya	elder	younger	child	flag	kekka	…	cls	
0	—	—	—	1	'Q'			—	→ <code>?- member(c,[a,b,c,d]).</code>
1	0	—	2	—	'F'	失敗		—	→ <code>member(X,[X _]).</code>
2	0	1	—	3	'R'			—	→ <code>member(X,[_   Y]) :- member(X,Y).</code>

図15 実行制御表

CKUPrologでは、exeQuestメソッドが中心となって実行の基本制御を行う。また、場合により、rechallengeメソッドにより失敗ノードの処理を行う。

### 3.4.3 単一化

単一化 (unification) とは、2つの構造を同一にするというPrologにおける基本的なプロセスである。例えば、2つの述語

$p(X,b)$  と  $p(a,Y)$

は、変数Xの値をaに、変数Yの値をbにすることによって、同一の述語 $p(a,b)$ を表すようにすることができる。これが単一化である。また、このように、値を持たない変数は単一化によって値を持つことができる。

CKUPrologでは、単一化の処理はunifyというメソッドで実現している。そのアルゴリズムの概要を図16に示す。図中、t1とt2が単一化の対象を表す。図16からも明らかなように、unifyメソッドは再帰的な定義となっている。これは対象となるデータが再帰的な構造であるからである。

```

if (t1とt2のアドレスが等しい) {trueを返す}
if (t1かt2のいずれかが無名変数) {trueを返す}
if (t1が値を持たない変数) {
    if (t2も値を持たない変数) {両者を単一化し、trueを返す}
    else {
        t2をコピーし、t1の値とする。
        trueを返す。
    }
}

```

```

    }
    else {
        if (t2が値を持たない変数) {
            t1をコピーし、t2の値とする。
            trueを返す。
        }
        else {
            if (t1のルートとt2のルートが等しくない) {falseを返す}
            以下を項数だけ繰り返す {
                t1のk番目の項arg1を取り出す。
                t2のk番目の項arg2を取り出す。
                unify(arg1,arg2)を再帰的に実行する。
                if (結果がfalse) {falseを返す}
            }
            trueを返す。
        }
    }
}

```

図16 単一化メソッドunifyのアルゴリズム

#### 3.4.4 バックトラック制御

バックトラックとは、木構造上既に通ったノードに逆戻りすることをいう。Prologでは、質問がどうしても成功しない場合や別解を求める場合に、最新の成功ノードにバックトラックし、そのノードを強制的に「失敗」させ、処理を続行する。

最新の成功ノードは、存在する場合必ず木構造の右下にある。そこで、実行制御表の先頭から、末っ子ノードを順次調べていけばよい。この処理は、CKUPrologではbacktrackメソッドで実現している。このメソッドも再帰的に定義されている。

なお、ノードを強制的に失敗させる際は、そのノードで付値された変数があればその変数値を解放する。

#### 3.4.5 カットオペレータ制御

カットオペレータ（!で表す）とは、組込み述語の一つで、バックトラックによる別解探索を阻止する機能を持つ。カットオペレータは、通常、組込み述語trueと同様単に成功するだけである。しかし、バックトラックで戻ってくると、そのカットオペレータを含む規則の頭部と単一化した質問（親ノード）を強制的に失敗させる。

カットオペレータの処理を行うのは、前述のrechallengeメソッドである。

#### 4. 実験結果と今後の課題

鑰山ゼミの3年生・4年生に、実際にCKUPrologを使用してもらい、アンケートに答えてもらった。以下にその結果を示す。

まず、システム全体の操作性に関しては、回答者17名中16名（94.1%）が使いやすいと答えている。これは、ゼミ生達がウィンドウズの操作になれており、ウィンドウズ対応のソフトに違和感を持たないためと考えられる。ただ、全角入力と半角入力の切り替えを面倒だと答えている学生が2名（11.2%）いた。どちらでも処理できるように改良する必要があるだろう。

次に、CKUPrologを使用することによりPrologや記号論理学について理解を深めることができたかどうかという質問に対しては、14名（82.4%）がyesと答えている。Prologも記号論理学も文系の学生には理解しにくいものではあるが、システムを使いながらの学習は効果があると評価できる。

CKUPrologはまだ試作品の段階を出ていないが、とりあえずの実験では教育支援システムとなりうる良い方向性が得られた。今後は、ヘルプ機能をもっと充実させることにより、さらに使いやすいシステムへと発展させる必要があるだろう。

また、今回試作したCKUPrologはPrologのインタプリタであり、記号論理学の入門的学習を行うためのシステムではない。Prologを理解できない学生達の多くは、論理演算子や三段論法といった基礎レベルの理解ができていない可能性が高い。そのため、CKUPrologを、それらの学習を支援するシステムへと拡張する必要がある。そこでは、日本語による質疑応答や、日本語文による推論といった機能が要求されるであろう。これは、新しいシステムが、知的CAI (ITS) として実現されなければならないことを意味している。



## 参考文献

- 1) Aho,Sethi,Ullman : Compiler , Addison Wesley(1985)
- 2) Chan (舟木将彦 訳)「Javaプログラミング1001チップス」 オーム社(1998)
- 3) Chang : Symbolic Logic and Mechanical Theorem Proving , Academic Press(1973)
- 4) Clocksin : Programming in Prolog , Springer Verlag(1987)
- 5) Dowsing : A First Course in Formal Logic and its Application in Computer Science, Blackwell(1986)
- 6) Kagiya,T. 「On implementing a Prolog Interpreter System」 産業能率大学紀要 (1986) 第6巻, 第2号
- 7) 鑰山徹「RUN/Prologを用いたPrologプログラミング入門」 工学図書(1987)
- 8) 河西朝雄「Visual J++入門」 技術評論社(1998)
- 9) 志村正道「人工知能」 森北出版(1994)
- 10) 岡部恒治ほか「分数ができない大学生」 東洋経済新聞社(1999)

(かぎやま とおる 本学教授)