数理解析研究所講究録
525巻 1984年 213-224



# 知識ベースにおける
# 論理型手続き起動のメカニズム

島 健一 (Kenichi Shima)
(日本電信電話公社 武蔵野電気通信研究所)

## 1. はじめに

近年、知識工学の分野でエキスパートシステムの研究、開発が盛んに行われている。エキスパートシステムは、特定の領域の複雑な問題を専門家(エキスパート)の知識を用いて計算機上で問題解決をはかるシステムである。エキスパートシステムを構築する上では以下のことが問題となる。

(1). 柔軟でわかりやすい専門知識、推論手続きの表現。

(2). 高度な推論メカニズムの実現。

(1)に関しては、次のことが望まれる。

・汎用性があり、人間の思考過程とマッチした表現ができること。

・概念知識は一般に階層構造を持つと考えられるので、知識表現においても階層性を表現できること。

また、(2)に関しては、次のような機能を提供することが望まれる。

・利用者が推論手続きをわかりやすく、簡便に定義できること。

・細かい構造や動作を意識せずに目的に応じた柔軟な制御が行えること。ただし、利用者に制御のための負担がかかってはならない。

上記の要請を満足するため、汎用性が高く、階層的知識表現が容易に行えるフレーム型知識表現[1]と、強力なパターンマッチング機能を有するPrologを融合する。これにより柔軟な設計知識の記述を可能とする[2]。

本稿では、その基本的考え方と処理のメカニズムについて述べる。さらに、推論過程の状況を把握することにより、フレームの探索範囲を限定し、効率良い推論を可能とする適応型推論メカニズムについても報告する。

## 2. 知識ベースと論理型プログラム

知識ベースは、個別的事実の集合のみではなく、人間の概念に相当するものや、知識それ自身の利用方法なども含まれるという点で、従来のデータベースとは基本的に異なる。知識ベースの構築にあたっては、以下の点を考慮する必要がある。

(1). 知識の大容量化に伴うモジュール化が可能で、知識の追加、修正、削除などの操作が知識全体にわたり矛盾のないことを保証し、その操作は自動的に行われること。

(2). 対象の記述のみならず付随する手続きも対象と関連するものとして表現でき、その定義、更新操作が容易であること。

上記の点を満たすものとして、我々はMinskyの提案したフレーム理論[3]に基づく知識表現システムを採用した。

(1)については、階層的なデータ構造による表現、および、遺伝属性による無矛盾のチェックにより実現される。

(2)については、事実データとともに、それに関する推論手続きを、手続き付加(Attached procedure)という統合された形で表現することで実現される。

フレームシステムは人間の記憶構造と類似するといわれ、階層的な構造と遺伝属性を持つフレームを単位として構成される。フレームは、ひとつの概念を表現し、その具体的な事実、性質、関連する手続きなどはフレームの構成要素であるスロットに記述する[4]。

次に、フレーム型知識表現と論理型プログラムの関係について述べる。フレームに関する推論操作として頻繁に使われる機能は、フレームとスロットの参照、及びスロットの値の参照、格納である。また、階層関係を利用した構造の変更(フレームの生成、削除など)を行う場合もある。これらの操作を柔軟に行える推論機構を実現するものとして論理型言語の代表であるProlog[5,6]をベースに、これをフレーム処理向きに拡張したシステムを考える。ここでは、そのシステムをPRIME(PRolog wIth fraME unifier)と呼ぶ。

Prologは、一階述語論理に基礎をおき、各ステートメントはホーン節と呼ばれる形式からなる。プログラムで表現するのは、事実とか、規則のようなある関係や性質を記述することである。処理の進め方は、ステートメントの順番による命令の実行ではなく、「統一化」(Unification)と呼ばれる一種のパターンマッチングによって、実行すべき節が決まり、この関係を用いて処理を進める方式である。これらの性質は、知識ベースのアクセス、推論操作に適合する。

PRIMEの知識ベース向きの特徴として以下のような点があげられる。

(1). 階層的知識表現が容易に行えるフレーム型知識表現と、強力なパターン

マッチング機能を有するPrologを融合することにより、簡便でわかりやすい推論手続きの記述ができる。

(2). 述語型言語の特徴を生かして、知識ベース間の関係を宣言的に記述できる。

(3). パターンマッチング機能を用いて、知識ベースへの柔軟なアクセス、知識ベース内のデータに対する計算、操作(Lisp関数もしくはProlog述語による)などを、統一的に処理できる。

(4). 推論の方向が一定ではなく、パラメタの入出力関係を実行時に決めることができるので柔軟な制御がおこなえる。

PRIMEは、現在、DEC2060上で、MACLISPを用いてインプリメントされている。また、PRIMEは、エキスパートシステム構築用総合環境KRINE (Knowledge Representations and INference Environment) の推論機構として位置づけられる。KRINEについては別途報告する。

## 3. PRIMEの基本構造

### 3.1 知識ベースの内部表現(フレームの構造)

フレームの概念図を図1に示す。各フレームは、システム中でユニークな名前

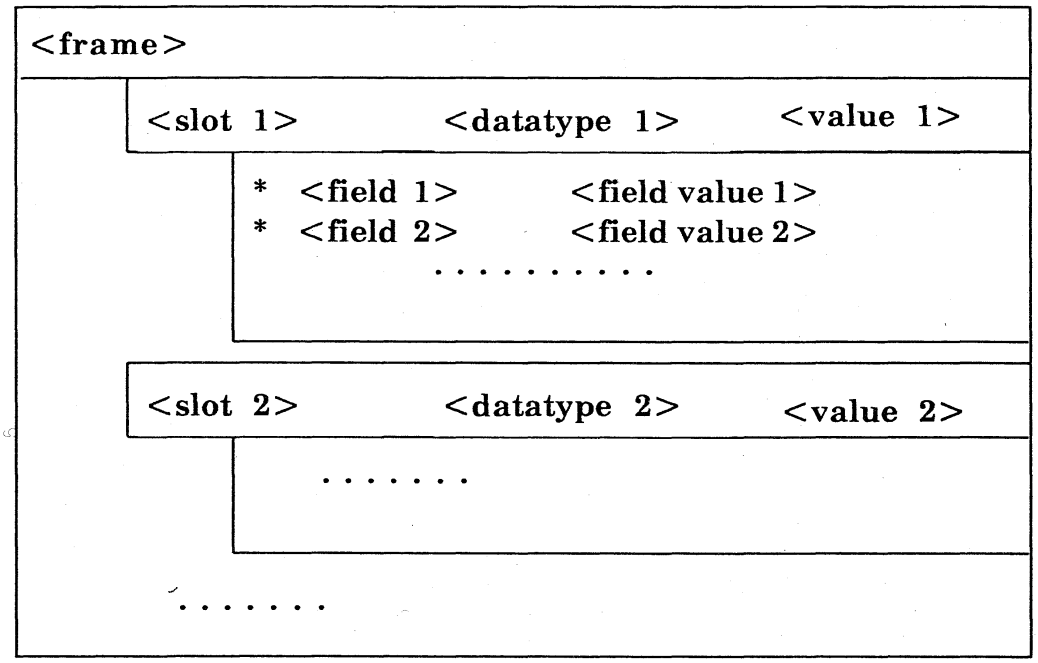


図1. フレームの概念図

を持ち、複数のスロットから構成される。各スロットは、スロットの値、このス

ロットが持つデータ値の性質を指示するデータタイプ、親フレームから子フレームへ受け継ぐべき属性を指示するインヘリタンス･ロールから成る。インヘリタンス･ロールを設定することにより、階層関係の知識定義に制約を課し、知識ベースの一貫性を保つのに役立つ。また、フィールドは、スロットの付加情報で、値として手続きを書くと、スロットに対する操作(参照、更新)が行われた場合、その手続きが起動される。この起動される手続きのことをデーモンと呼び、知識ベースの管理、無矛盾性のチェックなどに用いる。

PRIMEでは、上記構造の内、フレーム名、スロット名、スロットのデータ値を用いる。推論操作においてはこの三つで十分であり、その他は推論操作の過程でシステムが自動的に行うためである。

## 3.2 節の表現

### 3.2.1 構文規則(シンタックス)

PRIMEのシンタックスをBNF記法で示す。なお、";" 以下は注釈で、直接シンタックスとは関係ない。

＜定義＞

＜節＞::={＜正のリテラル＞} {＜負のリテラル＞} :

; ただし、各節に含まれる正のリテラルは高々1個である(ホーン節)という制限がある。

＜正のリテラル＞::= + (＜述語＞ {＜項＞})

＜負のリテラル＞::= − (＜述語＞ {＜項＞})

＜述語＞::=＜名前＞

＜名前＞::=＜英字＞{＜英字＞|＜数字＞}

＜項＞::=＜変数＞|
＜定数＞|
＜リスト＞|
(＜関数＞ {＜項＞})

＜変数＞::= *＜名前＞ ; 名前の前に *を付けたもの。

＜定数＞::=＜シンボル＞|＜数字＞|＜ストリング＞

＜シンボル＞::= 任意の文字列

＜ストリング＞::="＜シンボル＞" ; 任意の文字列を" "で囲んだもの。

＜リスト＞::=({＜定数＞}) ; 定数の並びを "( )" で囲んだもの。

＜関数＞::=＜名前＞

＜節の例＞

+(Append nil *x *x ) :

+(Append (*a . *x) *y (*a . *z) ) - (Append *x *y *z) :

PRIMEの節は、属性リスト(Property Lists)の形で定義することもできるが、フレーム中のスロットの値として格納することもできる。フレーム中の節の定義の例を図2に示す。フレーム中にある場合は、その述語名を持つフレームを実行時

```
cousin
slot                datatype
clause:             <prolog>
        +(cousin *x *y)
                -(parent *x *z)
                -(parent *y *w)
                -(brother *z *w):
```

```
brother
slot                datatype
clause:             <prolog>
        +(brother *x *y)
                -(*x father *z)
                -(*y father *z)
                -(not eq (*x *y)):
```

```
parent
slot                datatype
clause:             <prolog>
        +(parent *x *y)
                -(*x father *y):
        +(parent *x *y)
                -(*x mother *y):
```

図2 フレーム中の節の定義

にアクセスする。ただし同名の述語が属性リストとフレームに両方存在する場合は、属性リストの方を優先する。

### 3.2.2 リテラル、および項の記述

PRIMEは、対象知識のマッチング、推論操作の記述を可能とするため、以下のものを各節に書くことができる。

(1). フレームのアクセス

図1に示したフレームの構造を基本として、項 (term) の記述としては、フレーム名、スロット名、データの値 (各データタイプに応じた値) から成る3つ組として以下のように表現する。ただし、ある値を参照する場合は変数とする。

( <Frame name> <Slot name> <Data Value> )

<例> ( *X YUMIKO *Y ) ; スロット名に YUMIKO を持つフレーム名とその値を求める。

(2). Lisp関数

Lisp関数の呼び出しの場合は、以下のように記述する。

( <Function name> < Parameter の並び> )

これにより関数をその引数に対して適用する。但し、引数はその値が実行時に定まっている必要がある。つまり、引数は入力パラメタとして扱う。

<例> ( CONS *X ( FATHER *Y ) )

(3). Prolog述語

Prolog述語の呼び出しの場合は、以下のように記述する。

( <Predicate name> <項の並び> )

<例> ( COUSIN *X YUMIKO )

組み込み述語の一覧表を付録に示す。従来と異なるものとして、上で述べたLisp関数の呼び出し ( CALL , EVAL )、フレームのアクセス ( FCALL ) の他に、フレームのグループ化 ( &LIMITS )、結果の持ち帰り ( RETURN ) の述語がある。

## 4. PRIMEの処理概要

PRIMEの処理は大きくわけて、統一化とバックトラックに分けられる。

### 4.1 統一化機構

PRIMEは、統一化の対象として、Prologの節ばかりでなく、フレームもその対象とするように拡張した。これにより、同一の推論メカニズムのもとで、フレームのアクセスと手続きの起動が行えるという利点が得られる。その処理の概要を以下に示す。

3.2.2章の項の記述で述べたように節の各項には三種類のものが記述できるが、同じ述語名をそれぞれの種類に重ねて定義した場合、以下の優先順位に従い処理する。

(1). Prolog述語の呼び出し。

(2). Lisp関数の呼び出し。

(3). フレームのアクセス。

まず、(1)のProlog述語として扱う。述語として定義されていれば、即、この項の処理を行う。もし、なければ次に(2)のLisp関数として扱い、それを起動する。関数を適用した結果がNilであれば、バックトラックが起こり処理をやり直す。もし、(1)、(2)でなければ、(3)のフレームへのアクセスとして扱い、3つ組の節をつくりだす。これ以外の場合は、すべて失敗する。但し、(2)を優先したければCALLを、(3)を優先したければFCALLを付ける事で、優先順位の指定ができる。

## 4.2バックトラック機構

節の実行に失敗したものは、その過程で生じた変数への代入を解除し、バックトラックを起こし、次の候補をさがす。従来は副作用の生じた節はバックトラックの対象にならなかったが、これでは推論操作に不向きであるので以下の点を改良した。

(1). Lisp関数を起動した結果"NIL" が返ると、自動的にバックトラックが生じる。これは試行錯誤的に推論操作を行う場合などに便利である。

(2). 推論の実行過程において、フレームの変更操作が行われた後バックトラックが起きると、自動的に以前のフレームの状態に戻す機能を備えている。例えば、ある評価関数にもとづいて設計を行う場合、ある状況で良いと思って設計したものでも、別の 場面ではまずい場合がある。この時、当然、設計対象は変更されており、これをもとの状態に復元しなければならない。この操作を実行時に行う。その方法は、推論の各過程で生じたフレームの変更情報を、変数に対する代入(substitution)と同様な機構で保存し、バックトラックが生じた時、この情報をもとに復元することで実現している。ただし、フレーム全体を保存するのではなく、復元に必要な最小限の情報のみを蓄える。

## 5. PRIMEの拡張機能

## 5.1. 適応型推論メカニズムの基本的考え方

推論時に導出が失敗した場合、自動的にバックトラックが起こり、他の多くの候補を探索するため実行時間がかかりすぎるという問題点がある。この解決策として、探索範囲を限定し、バックトラックの回数を極力少なくし、推論のパスをなるべく短くすることが必要となる。バックトラックを引き起こす失敗の原因を捜し、それを取り除くようにシステムを構造化することが適応型推論である。そのためのメカニズムとして、以下のレベルが考えられる。

(レベル 1). 推論結果を蓄積し、それを再利用することによる適応化。

(レベル 2). 節に対する制約条件の付加など、探索範囲限定による適応化。

(レベル 3). 節の構造や節の関係などの抽出による新たな推論手続きの発見による適応化。

以下では、レベル1、レベル2について報告する。レベル3については今後の検討課題である。

## 5.2 推論結果の蓄積、再利用による適応化

従来のPrologは、失敗した節も、成功した節も、再度、統一化の対象になり、何度も同じことを繰り返し、多くのバックトラックを引き起こした。幾つかのProlog処理系[5]では、節をダイナミックに作成し、蓄積する"Assert"という組み込み述語があるが、それは成功した節しか登録できない。そこで、推論の実行過程における各節の結果を予め用意したフレーム中に蓄積し、その蓄積結果を利用して、同じ節は繰り返しの対象とせず、早めに枝刈りを行い、統一化を効率良くする。以下にその手順を示す。

### (ステップ1). 推論結果の蓄積

実行された節を、導出の成功、失敗に応じ、それぞれ、:Proved-Clause (P-C), : Not -Proved-Clause (N-P-C) というフレーム ( 対応する節のヘッダ名と同名のスロット ) に追加格納する。これらの格納された節は、ステップ2の統一化の時に用いられる。

### (ステップ2). 蓄積結果の再利用による統一化

フレーム中に蓄えられた節を用いた場合の統一化の動作を述べる。まず、入ってきた節がフレームN-P-Cの節の中に含まれていれば、即、統一化は失敗する。その結果、次の候補を捜す。もし、P-Cの節の中にあれば、それを定義節として利用して統一化の処理を行う。

## 5.3. 探索範囲限定による適応化

推論実行のログをとり、これを解析することにより、以下の戦略に従って探索範囲を限定する。

(1). 失敗の多いフレームのアクセスを禁止する節を付加することで探索範囲の限定をする。

(2). 使用された実績に応じてフレームをグループ化し、探索範囲の分割を行う。

(3). 各節の変数がどのようなフレームと結合するか調べ、なるべく単一のフレームとマッチするような関係を導き出し、それに応じて節も変更する。例えば、親一子の関係があれば、子一親の関係を作成することで探索範囲を絞る。

## 5.4 適応化の実行例と効果

例として、図2のCOUSINの推論を考える。推論結果の蓄積、再利用による適応化(レベル1)では、試行回数10回程度で、実行時間が半分に短縮された。どちらかというと、失敗した場合の情報が重要なことが多く、バックトラックの回数を軽減できる。

探索範囲の限定による適応化(レベル2)を行うため、トレース情報などをもとに解析を行った。この結果、図3のような親のいない人は除くという制約条件を付加した節を考えると効率の良い推論を実行できる。この場合は4—5倍高速化される。

```
+(cousin *x *y)
        -(parent *x *z)
        -(not (memq *z
              (shintarou kazuko toshio junko)))
        -(parent *y *w)
        -(not (memq *w
              (shintarou kazuko toshio junko)))
        -(brother *z *w):
```

図3. 制約条件を付加した節

## 6. おわりに

フレーム型知識表現とPrologを融合することにより推論手続きの記述、実行において以下の点を可能にした。

(1). フレームの形で表現された知識ベース内の構造的な情報も、必要に応じてダ

イナミックにアクセスし、それを利用できる形とした。
(2). 知識ベース内のデータに対する操作、計算などを行うLisp関数、Prolog述語の両者を柔軟に扱える様にした。

以上のことは、知識を扱う場合、ある場合には、フレームの値をアクセスし、また、ある場合には、手続きを起動した結果を必要とすることがあり、これらを適宜柔軟に取り扱うために有効である。

今後は、5.3章の適応化の知識をメタ・ルールとして表現し、より良い推論手続きを作成し、効率の良いシステムの構築を進めて行く。さらに、高度な推論メカニズムの実現を検討する。

最後に、日頃御指導頂く、山下情報通信基礎研究部第一研究室長はじめ、知識工学グループの方々に感謝します。

<参考文献>

[1]. 小川 裕、島 健一、菅原 俊治："知識エディタKE-0について",情報処理学会第26回全国大会3C-4

[2]. 島 健一、高木 茂："知識エディタKE-0における手続き付加について",情報処理学会第26回全国大会3C-5

[3]. Minsky, M : " A Framework for Representing Knowledge", The Psychology of Computer Vision , Mcgraw-Hill, New York, 1975

[4]. Stefic, M : "an Examination of a Frame-structured representation system", Proceeding of IJCAI, 1979

[5]. W.F.Clocksin, C.S.Mellish : "Programming in Prolog", Springer-Verlag, 1981

[6]. 中島 秀之："Prolog 入門", bit vol. 11, no. 5, 6, 7, 1982

付録　　システム述語一覧

・(& LIMITS ＜リスト＞)
　フレームのグループ化
・(ADD ＜数字＞ ＜数字＞ ＜項＞)
　LISPのADD
・(APPEND ＜項＞ ＜項＞ ＜項＞)
　リストのAPPEND
・(ASSERT ＜項＞)
　節の定義
・(CALL ＜関数＞ ＜パラメタ＞)
　LISP関数の呼び出し
・(CCALL ＜項＞)
　＜項＞の呼び出し
・(CDUMP ＜名前＞ |ALL|INIT|FCLAUSE)
　節の表示
・(CUT)
　CUT OPERATOR
・(DELC ＜名前＞ |ALL|INIT|{＜数学＞})
　節の削除
・(DELETE ＜名前＞)
　節の削除
・(END ＜項＞)
　PRIMEの終了
・(EQ ＜項＞ ＜項＞)
　等価性のチェック
・(EVAL ＜項＞ ＜項＞)
　LISP関数の結果の統一化
・(EVAL-CALL ＜関数＞ ＜パラメタ＞)
　LISP関数の呼び出し
・(FAIL), (FALSE)
　失敗の節
・(FCALL ＜Frame＞ ＜Slot＞ ＜Value＞)
　フレームのアクセス
・(IF ＜項＞ ＜項＞ ＜項＞)

IF-THEN-ELSE タイプの条件選択

・(NONVAR ＜変数＞)
　代入変数のチェック

・(NOT ＜項＞)
　＜項＞の否定

・(OR {＜項＞})
　＜項＞の並びの論理和

・(REPEAT)
　くり返し節

・(RETURN ＜変数＞)
　変数の値のもち返り

・(RUNTIME-END ＜変数＞ ＜変数＞)
　時間計測の終了

・(RUNTIME-START ＜変数＞)
　時間計測の開始

・(TRUE)
　真の節

・(VAR ＜変数＞)
　未代入変数のチェック