人工知能と対話技法 25－4
（1982. 3. 25）

知識ベースシステムの開発環境とその応用について

東京理科大学理工学部
経営工学科
溝口 文雄

1．はじめに

知識ベースシステムの設計と開発のためのプログラム環境の全体像を考えることが本報告の目的である。ただし、知識ベースシステムというのは医療診断システムで代表されるエキスパートシステムで、いわゆる推論機能を持っている知識情報処理システムを言っている。

この報告では、知識ベースシステムの特徴を最初に述べ、次にどのように設計していけばよいかを検討し、これを支援するプログラム環境について述べていく。

2．知識ベースシステムの特徴

知識ベースシステムの基礎となる知識は専門家のものを前提としている。このことは、知識ベースで使われる知識は、常に改良していかなければならない性格を有している。つまり、知識ベースは"開放的（Open）"な形で構築されていなければならない。知識ベースシステムの開放性は、次の二つの意味を持っている。第一は、知識は各々が独立構造となっていて、単位間にはそれほど相互作用がない。例えば知識がルールの形式で記述されているときは、独立したルールとして知識ベースにストアされる。従って、知識はルールの修正・追加および削除に対して開放的である。その結果、知識はルールとして加法的に増加していく。ただし、性能は必ずしも加法的に向上していくとは限らない。第二は、知識そのものが動的な性質を持つものと考えるもので、知識の単位そのものが可変的構造としておくことである。知識がその他の知識と相互作用することにより、新しい知識を生成したり、また構造的に組み変えていくということを意味している。こうした知識の構造そのものの開放性もあるわけである。

特に、人間の知識の場合には、第二の特徴を有し、従って専門家となるためには長い経験を必要とする。

これらの第一、第二の開放性は、いずれにせよステップ的な精密化（step-wise refinement）を知識ベースに加えていくことに対応している。つまり、知識ベースシステムは短期間のうちに完成されたシステムとして構築できないという特徴を持っている。

逆にみると、次のことが言える。

1）第一次の近似システムの知識ベースシステムは、ステップ的な精密化を受けるという前提で、できるだけ短期間のうちに完成すること。

2）知識ベースシステムの基礎となる知識は変化するという開放的な性格を有しているので、できるだけ簡単な表現形式にすること。

3）知識ベースシステムに対する知識の増加が、必ずしもシステムの性能アップにつながらないこと。

これらの特徴はシステム構築のアプローチにも結びついている。すなわち、方策としては、1）に対しては、できるだけ早く知識ベースシステムを動くシステムとして

開発せよ、そして最初から高性能をねらうなと

2)に対しては、できるだけ読みやすい知識の表現形式を使うこと、そしてどんどん作り変えていくことを心がけよと

3)に対しては、専門家といっても種々のレベルがあること。もし、知識を増してもシステムの性能が変らないときは、その専門家のレベルを考えよとそれぞれ考えることができる。従って、知識ベースシステムの構築のためには、ステップ的な精密化と高度の質の良い知識の収集に時間をかける必要がある。

次に、このような特徴を持った知識ベースシステムをどのような構造にしていけばよいかを述べる。

3．知識ベースシステムの構造

我々が設計対象とする知識ベースシステムの構造を示したのが、図1である。特徴は、多層レベルよりなる複雑なシステムであり、各々のレベルは、基礎のプログラム環境から、応用システムの構成までに分割されている。

第1レベルは、基礎言語である。既存の言語でいうと、LISPのためのLAPコード、あるいは、システム記述言語と言われるものである。VAXなどを計算機として使うことを考えると、C言語などが考えられる。

第2レベルは、基礎言語から作成されたプログラム言語である。知識ベースは記号処理が多用されるので、LISPなどが具体的な言語となる。また、述語論理型の言語でみればPROLOGなども、強力なプログラム言語と考えることもできる。ただし、より応用向きの言語とするためには、LISPにおける構造エディターがPROLOGにも必要である。その他、基本関数のミススペル修正機能も必要であろう。

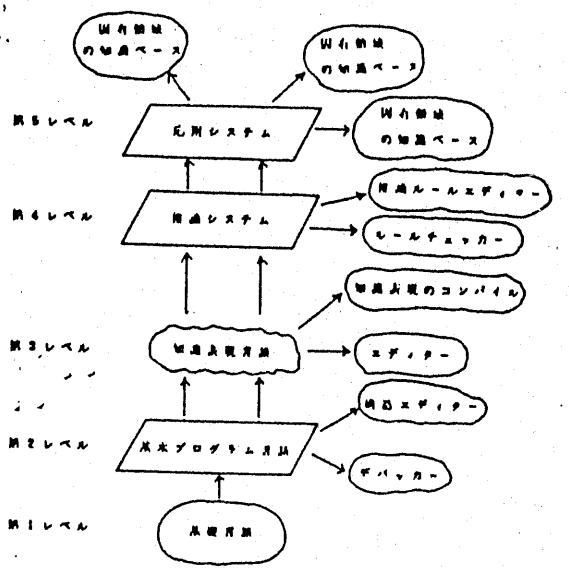

図1．知識ベースシステムの構造

第3レベルは、知識ベースのため表現言語である。フレーム構造を定義するとすれば、フレーム表現言語となる。いくつかの言語が開発されているが、我われが現在検討しているのは、階層遺伝性とクラス概念が導入できる言語である。

また、表現言語を定義できるようなエディターおよびフレーム構造のシンタックスをチェックする機能が必要である。

第4レベルは推論システムで、高次の推論を行なう部分である。具体的には、ルールベースシステムである。ここでの特徴は、知識ベースとは独立した、ルールベースという考え方である。もし、知識表現の形式に重点をおくと、ルールベースは手続き付加的な考え方で構成することになる。

また、データ駆動型の推論システムのときは、知識表現は簡単になり、ルールベース中心のシステムとなる。

第５レベルは、応用システムである。この段階では、知識ベースの対象により、種々の応用システムを構成できる。ただし、こうした応用システムを作成するためには、現在のエキスパートシステムの知識のトランスファーが必要である。そのためのインターフェイスも考慮する必要がある。以上が、知識ベースシステムの構成であるが、もちろん、各レベルを誰が担当するかで、プログラミング環境は異なってくる。

現在、我々はLispをベースとした各階層の構築を進めている。具体的には以下の通りである。（図２）

この環境をKulti-Layered Programing Environment（ＭＬＰＥ）と呼ぶ。

第１層は、核言語層（Core Language Layer）で、最も基本的なLispそのものである。核言語として他の言語を選択してもよい。第２層は、基本関数層（Basic Function Layer）で、Lispの Primitive-Functionsより成り立つ。我々の使用しているLispはInter-Lispとそのサブセットである F3-Lisp であり；この Primitive－Functions には Inter-Lisp の組み込み関数の他に、頻繁に利用するとして独自に開発した基本関数（SUBR,FSUBR,EXPR,FEXPR）が含まれている。

第３層は、Primitive-Functions内の基本関数を組み合わせて作成した、Lisp-Editorや Prity-Print で代表されるユーティリティ関数（EXPR,FEXPR）群で、Useful Function Layerと呼んでいる。

以上の第１，２層は、上位の第４；５層の作成・修正をサポートするものであるが、Inter-Lispではその大部分がInter-Lisp自身に含まれている。

第４層は、人工知能の基本的手法（e.g. Pattern-Matcher , Production System , Natural Language Processor , Knowledge Representation System）のモジュールプログラム群であり、Module System Layerと呼んでいる。

以上の第１～３層が、知識ベースシステムの開発を支援するＭＬＰＥであり、利用者はこのプログラム環境に加えて、各自の Original-Function-Setを用意し、第４層に位置する汎用の知識ベースシステムを開発することになる。第５層は、下位の支援環境を使って作成する固有の知識ベースシステムであり、Knowledge Base Layerと呼ぶ。

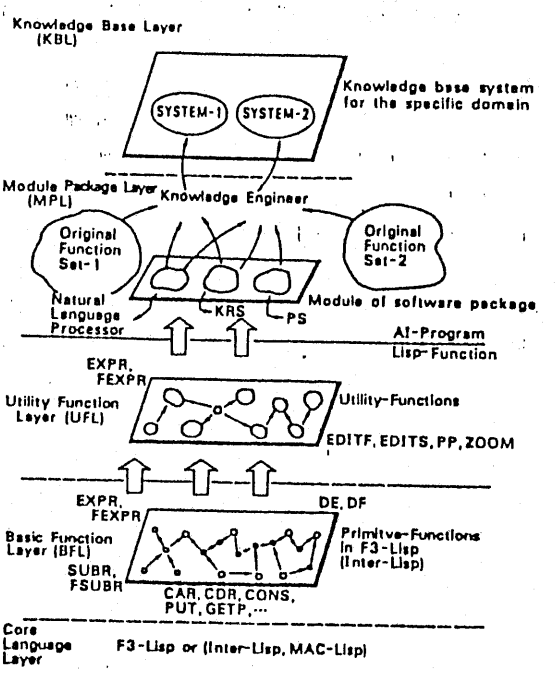

図２．知識ベース作成のための階層的プログラム環境（ＭＬＰＥ）

4．知識ベースシステムの作成

知識ベースシステムという場合には、通常、第5レベルの応用システムの設計を言う場合が多い。したがって、このレベルの環境としては、ルールの記述と対象領域の知識だけを記述しておけば、応用システムが開発できることになる。具体的には、例えばEXPERT、およびプロダクション・システムが考えられる。

このレベルの環境で考慮することは、領域の専門家との協同のルール作成だけでよい。すなわち、なんらのプログラム行為というものを意識しないで応用システムの開発ができるという環境である。

知識工学の立場からみると、知識ベースシステムの環境は、第4レベル、第3レベルとなる。すなわち、推論システムと知識表現言語の開発である。したがって、新しい推論システムと知識表現言語の作成のためには、多種多様の推論メカニズムのサブプログラムと、知識形式を開発する必要がある。このためのプログラム環境としては、パッケージプログラムの利用、およびモジュラーシステムの存在が必要となる。現在、知識ベースのためのソフトウェア支援システムに必要なのは、このレベルでのプログラム環境である。

4．1　知識表現システムの構成

MLPEで、第4層に相当する部分が知識表現システムである。このシステムは、KRS（KNOWLEDGE REPRESENTATION SYSTEM）と呼ばれ、基本的にはFRL，KRL，UNITと同様に、フレームを表現の基礎にしている。

基本的設計方針を述べると、次のようになる。

1）フレームの定義と編集が、いくつかのフレーム操作用関数群で処理できる。

2）フレーム内の遺伝性がチェックできる。

3）デフォルト値処理ができる。

という特長にある。

KRSの構造は、図3に示すように、フレームの定義と編集が操作しやすいという特徴を持っている。

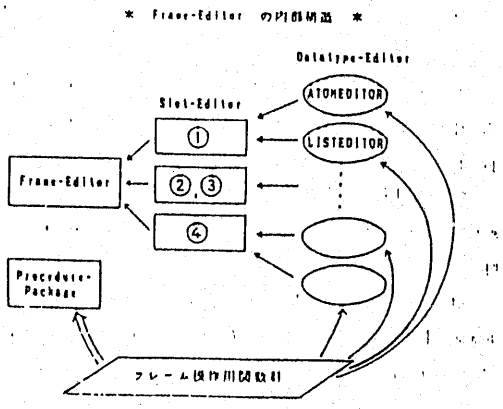

図3．KRSのエディター部の構造

以上のKRSを用いて、フレームを定義していくプロセスを示したのが、図4のプロトコールである。

```
"INPUT ED-COMMAND"
"P(RINT) A(DD) C(HANGE) D(ELETE) E(ND)"
[ED: PP
PP IS ILLIGAL COMMAND

"INPUT ED-COMMAND"
"P(RINT) A(DD) C(HANGE) D(ELETE) E(ND)"
[ED: P
     (DATE-1 (SELF (VALUE (DATE)))
             (INSTANCE (VALUE (NIL)))
             (DAY (INHERIT (R))
                  (VALUE (27)))
             (MONTH (INHERIT (R))
                    (VALUE (DECEMBER)))
             (WEEKDAY (INHERIT (R)))
             (YEAR (INHERIT (U))
                   (VALUE (1982)))
```

図4．出力例
（KRSのエディター部）

現在、ＭＬＰＥの構成の下にＫＲＳをインプリメントしているが、実際の応用についてはスケジュールおよびソフトウェアのドキュメントを考えている。　図５は、スケジュールの応用例を示したものである。この例では、自体のフレームが作成されていく様子を示している。ただし、ＫＲＳでは、理論部がないので、勤務には、推論手続をＫＲＳに付加している。

```
"INPUT ED-COMMAND"
"P(RINT) A(DD) C(HANGE) D(ELETE) E(ND)"
FED> C
"WHAT CHANGE ? ---> S(LOT) F(ACET) V(ALUE)"
FED> V
"WHICH SLOT ?"
"INPUT SLOT-NAME AND FACET-NAME"
FED> DAY VALUE
"INPUT NEW VALUE"
FED> 26

"INPUT ED-COMMAND"
"P(RINT) A(DD) C(HANGE) D(ELETE) E(ND)"
FED> A
***SLOT-NAME-?***
FED> WEEKDAY
***FACET-NAME-?***
FED> VALUE
***VALUE-?
FED> SUNDAY
     (DATE-1 (SELF (VALUE (DATE)))
             (INSTANCE (VALUE (NIL)))
             (DAY (INHERIT (R))
                  (VALUE (26)))
             (MONTH (INHERIT (R))
                    (VALUE (DECEMBER)))
             (WEEKDAY (INHERIT (R))
                      (VALUE (SUNDAY)))
             (YEAR (INHERIT (V))
                   (VALUE (1982)

***MAKE-OTHER-VALUE-?***
FED> N
***MAKE-OTHER-FACET-?***
FED> N
***MAKE-OTHER-SLOT-?***
FED> N

*** DATE-1 IS SORTED ***

*** DATE-1 - DAY --- DATATYPE - INTEGER -- OK ***
*** DATE-1 - DAY --- BOUNDED-INTEGER CHECK -- OK *
*** DATE-1 - MONTH --- DATATYPE - NAME -- OK ***
*** DATE-1 - MONTH --- ONE-OF CHECK -- OK ***
*** DATE-1 - WEEKDAY --- DATATYPE - NAME -- OK ***
*** DATE-1 - WEEKDAY --- ONE-OF CHECK -- OK ***
*** DATE-1 -INSTANCE-CHECK- END ***

"INPUT ED-COMMAND"
"P(RINT) A(DD) C(HANGE) D(ELETE) E(ND)"
FED> E

"INPUT EDIT-COMMAND"
(NF FL P* ED FIN ?)
FED> FIN
"EDITED"
```

図５．フレーム作成プロセス

４．２　推論システムの構築

知識ベースを実際に構築する上では、'Production System（以下P-Sと略称）が、知識の記述に対して、柔軟で利用しやすいシステムとなっており、種々の分野に応用されている。ところが、実用に耐え、膨大でかつ内容の変更ができるような知識ベースの作成と処理速度の向上を統合して知識ベース作成の直見だてとすることが実際には必要である。ここでは、Productionルールから'Productionルールが生成可能で、かつ'Productionルールの　tree構造化を行なったＡＤＩＰＳ（Acquisition-DIrected Production System）を述べる。

なお、Productionをtree構造化する利点は、

（Ａ）同一条件を一つのリンクに集約できる。

（Ｂ）Production間に優先順位を作らない。

であり、検索効率のよい、Adaptive PSを構成可能にすることである。すなわち、新しいProductionルールをTreeに統合していく形で、Adaptationが行なわれる。

一般的なSingle level PSでは、

<Condition>　->　<Action>

のような形式をしているが、ADIPSでは、

<CV>　->　<Condition>

=>　<Action>

の形式をしている。ここで<CV>は、Productionのコントロール変数（CV）とコントロールタグ（CT）のペアを、<Condition>は、Conditionの集合を、<Action>はActionの集合を表わしている。例えば、図６のようなProductionは；図７のようなTree構造になる。

このようにProductionルールは、Tree構造化されているため、Produc-

tionの検索方法、Tree構造の検索に帰する。Adipsでは、DFS（Depth-First-Search）を用いている。

PRODUCTION NAME	CV & CT	CONDITION	ACTION
P1	(ABS VCON) ->	C1 & C2	=> A1
P2	(ABS VCON) ->	C1 & C3	=> A2
P3	(VCON = A) ->	C2 & C3	=> A3
P4	(VCON = A) ->	C2 & C4	=> A4
P5	(VCON = B) ->	C3 & C4	=> A5
P6	(VCON = B) ->	C3 & C5	=> A6

図6　ADIPSのProductionルールの形式

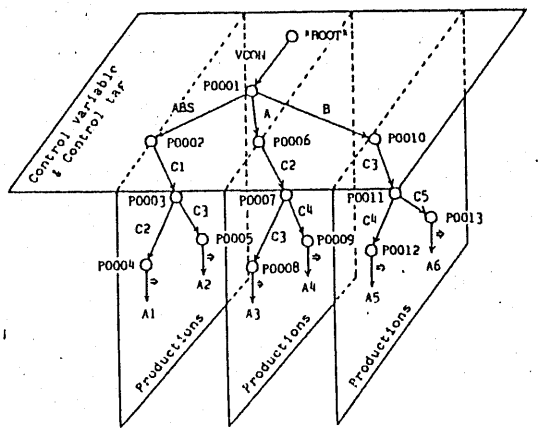

図7　ADIPSの制御構造

ADIPSはInter-Lispでインプリメントされ、DEC-20上で実動しているが、現在までに、人工知能の代表的例題、に適用し、ADIPS3（第3バージョン）になっている。このADIPS3特徴は、性能的には既存のproduction systemのruleを吸収することができる。

実際の診断システムを作成するためには、手続的なrule形式の他に、分類データ、故障データなどの宣言的表現が必要である。こうした目的を実現するために、ADIPS3ではframeを導入している。実際の応用例は、原子炉の故障問題に適用し、その有効性を確認している。

グローバルな制御構造は、production間のsemanticsを用いて図8のように、状態遷移ネットワークとして表現することができる。

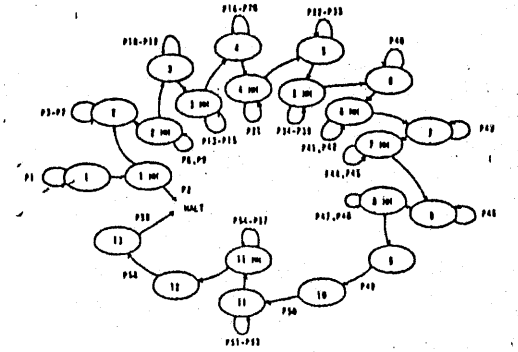

図8．　Prpductionルールの状態遷移ネット

診断は、1番目のfindingのframeを呼出して開始される。例えば、2番目のfindingのframeが呼出され、解答が与えられて、productionがfireした状態の変化とこの図は示している。

5．　おわりに．

本報告では、知識ベースシステムを作成するために必要な機能を、多層の階層システムとして、提案した。具体的なシステムは、LISPをベースにしたプログラム環境として開発を進め、知識表現システムKRS、推論システムADIPSを作成している。また、その応用領域としても、スケジュール計画、および原子炉の診断の適用している。

本研究は、日米科学協力研究（代表　東大　開原助教授）；総合研究A（代表　東大　東教授）、総合研究B（代表　阪大　田中教授）の補助によることを付記する。

文献）

1）、溝口、斉藤，
"知的情報処理の設計"
コロナ社（印刷中）