TR-312

昭和61年度知識システムシェル・ワーキンググループ(KSSWG)報告書

石塚　満、溝口理一郎、溝口文雄、新田克巳
寺野隆雄、小林慎一、福永光一、石田　亨
小林康弘、木田直道、中野　剛、丸山文宏
渡辺正信　ICOT第5研究室

October, 1987

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

昭和61年度　知識システムシェル
ワーキンググループ（KSS WG）

報　　告　　書

昭和62年　3月

ICOT研究所　KSS WG

61年度知識システムシェル
KSS WG 報告書 目次

（注）別綴じにて図面集あり

以　上

1．　不完全な知識の操作による高次人工知能機能

1.1　高次人工知能の実現へ向けての視点

人工知能研究が目指す知能には、たとえば図 1-1のようなレベルがある。演算、検索は今までのコンピュータが基本的にサポートしてきた機能である。演繹的推論は知識コンピュータが基本的にサポートしようとしてている機能である。このあたりまでのメカニズムはだいぶ解明され、エキスパートシステムやその他、知識システムで広く利用されるようになってきた。今後の人工知能研究は、その上位の高次知能機能の実現が大きな課題である。

筆者（石塚）らは、不確実な知識を扱うエキスパートシステム、知識型3Dビジョン、知識型LSI パターン設計用のセルライブラリ・システムなどの、知識システムの開発を手がけてきた。現実の世界、自然界の問題に立ち向かうとき、演繹的推論機能だけでは不十分であり、Ad hocな手法を取り入れることにより、ある種の高次知能機能の実現を図ってきた。演繹的推論だけでは、いわば想定される当り前の結果しか導かれないが、応用指向の手探りのやり方で、演繹的推論の枠を超える機能を追求してきた訳である。

ボトムアップ的に進めてきた研究・開発の中に、最近、今後の高次人工知能の実現に向けての鍵となる技術を認識したと思っている。それは、不完全な知識の役割である。（この間の事情は文献［石塚86a ］に記されている。）今日の知識システムの課題となっている常識の実現法、創造的な設計のパラダイム、帰納推論、発想の実現法なども、この不完全な知識と大いに関係する。

図 1-1に挙げたような高次人工知能メカニズムに対する個々のアプローチも重要であろうが、それだけでは次世代の知識処理の全体像が見えてこない。次世代知識ベースの基本構成という観点からも、個々の高次人工知能機能を取り入れればよいというのでは不十分であり、現実の知識システムにおける必要性、実用性をふまえ、将来へ向けての展望のもとに、ある種の統一的な観点から研究開発を進めることが重要である。特に、ICOTにおける次世代Shell の設計においては、論理プログラミングの基礎の上に、将来への発展性を指向した視点からのアプローチを採ることが重要であろう。

ここでは、筆者（石塚）らが採っている視点、すなわち、完全な知識に加えて不完全な知識も含めて知識ベースを構成し、これを操作することにより高次人工知能機能を実現する視点から、関連技術について述べることにしたい。不完全な知識という場合、知識／データの拡大解釈、もう少し具体的には変数化操作は特に重要な働きをする。

1.2 あいまいな知識の表現と利用

不完全な知識をもう少し広くとらえ、知識処理における「あいまいな知識」に関する手法について概観しておくことにする。［石塚85］

人間はあいまいな知識の環境下でも日常的に、問題解決、判断を行っている。知識システムも単なる記号推論の枠を超え、人間の思考に近づくためには「あいまいな知識」の表現法、その環境下で働く推論機構を求めなければならない。

「あいまい」といっても、その意味はあいまいであり、漠然としている。知識は形式化して記述しなければコンピュータで利用できないのと同様に、あいまいさもある形式で記述するようにしなければ工学的な扱いにはならない。

現時点での知識工学において扱われているあいまいさを、次のように分類しようと思う。

(i)　制御の非決定性（non-determinism ）
(ii)　多義性（multipe meanings）
(iii) 不確実性（uncertainty ）
(iv)　不完全性（incompleteness）
(v)　ファジィ性（fuzziness or imprecision）

以下、これらについて説明を加える。

(i) 制御の非決定制

非決定的制御は人工知能システムの本質的特徴である。断片的に知識を集積し、これを利用する推論の筋道を確定できないので、ある筋道を試行し、失敗したらバックトラックを行うという制御を行う。これが多様な状況に対処できる柔軟性、知的能力発現の源となる。

単純な探索では効率が悪いので、優先して探索すべき道を決める必要が生ずる。初期の人工知能研究で開発されたA*アルゴリズムなどのヒューリスティック探索は、必ずしも正確でない。（推定値である）評価関数をガイドにした探索法である。知識工学の時代になると推論（探索）も知識のガイドによって進めるという考え方がとられるようになる。ヒューリスティックな知識（必ずしも正しいとは限らないという意味であいまいさを含んだ知識）がゴールへ到る推論を速めるために大幅に取り入れられるようになる。その最も高度な例として、数学の概念を発見するAM［Lenat84 ］［島田87］を挙げておく。作成者のLenat が協調しているように、ヒューリスティックな知識が重要な役割を果たしている。

ヒューリスティックな知識は非決定的制御をガイドするのに用いられる、対象問題に依存した知識である。後述の不確実性や不完全性を含む知識環境下では、問題に依存しない汎用の効率的推論制御法があると思うが、まだ十分な研究はない。

(ii)多義性

自然言語理解や画像・音声理解では解釈の多義性は通常である。自然言語理解で

は単語の意味の多義性、係受関係の多義性、代名詞の文脈上の多義性などが大きな問題となる。一般に、より広い文脈の情報、意味的制約を利用して多義性の解消を図る。

画像でも画像要素（線分や領域など）の解釈は多義となることが多い。これも一般に、より広い空間的位置関係などを利用して多義性の解消を図る。弛緩法（relaxation method ）は反復演算により画像解釈の多義性を解消するシステマティックな方法である。［坂上82］離散的弛緩法と確率を用いる確率的弛緩法とがある。音声理解で有名なHearsay-Ⅱ［Erman 80］では音素、音節、語、文節、文といった解釈の階層があり、ある階層での解釈の多義性は上位階層での整合性により解消が図られる。Hearsay-Ⅱは、これを黒板モデルによる知識表現、制御構造により実現している。

(iii) 不確実性［石塚83a ］

知識システムの対象とする問題では、真偽（１か０か）の２値で表せないような不確実な知識や事実も利用しなければならない場合がある。 0.7だけ正しいといった種類の知識である。このような不確実性は従来はBayes の規則に従う確率量（便宜的にBayes 確率とする）で表されてきたが、知識工学では当初から知識に付随する確実度をBayes 確率として扱うことの不都合さの認識があった。（主な不都合はBayes 確率では無知量をうまく表せないこと［石塚83b ］）それゆえに、まず開発されたのがMycin のＣＦ（certainty factor）の方法である。［Shortliffe76］この間の事情は内部での討議も含め文献［Buchanan84］に詳しい。Mycin の方法は論理的裏付けをもったものではないが、不確実な知識の扱いの一範例を示したことで大きな影響を与えた。Mycin に影響され、主観的Bayes の方法（Subjective Bayesian method ）と名付けた推論法が提示され［Duda76］、Prospectorに用いられた。確率の線に沿う方法は、数学的理論構成をもつDempster-Shafer の確率理論［Dempster67］［Shafer76］［石塚83b ］が導入されるに至り、一応の結論が出たといってよいと思う。

Bayes 確率に対比すると、Dempster-Shafer 理論の基本確率は部分集合内を自由に移動する確率量としてモデル化されることが大きな違いである。無知量は全集合内を自由に移動する基本確率量として表現できる。Dempster-Shafer 理論は集合の概念を含んでおり、汎用ツールとしてのインプリメントは、事象の知識表現に現れる部分集合が共通部分をもつことを許すと複雑になるが、互いに排他的な部分集合だけに限ると比較的容易である。

ファジィ集合論から派生したファジィ論理は、論理式を１～０の間の真理値付きで扱う連続値論理の一種である。以上の確率の線に沿う方法とは幾分異なり、より主観的な、あるいは論理的性格が強い。知識ベースの観点からは、冗長な知識も許容する点が大きな利点だと思う。

知識の不確実性の扱いを考える上で、論理を基盤としてその基礎を確立することが今後の重要な方向である。論理を基盤にすることにより、２値論理で得られている多くの成果を不確実な知識の場合にも生かすことができる。そのような方向へ向けての研究を紹介しておく。

筆者らは、ファジィ論理を扱うPrologであるProlog-ELFを作成した。［Ishizuka 85］

ファジィ論理では知識（ルール形や事実形なども含む論理式）は１～０間の真理値付きで表す。真理値の扱いは確率と比較すると制約が少なく、またファジィ論理と言われるように論理的な性格が強い。

Lee はファジィ論理における導出原理について検討している。［Lee72 ］筆者らのProlog-ELFはこの検討に基づき、実用的に有用ないくつかの述語、機能も含め、ファジィ論理を組み込んだものである。

真理値 0.5以下の定義を許すと等価的に否定の定義となり、ホーン節集合の枠を超えてしまう。そこでPrologとするために真理値の範囲を（標準モードでは）１～0.5の間に制限している。導出節の真理値はある範囲内で保証されることになるが、この下限真理値を与えるようにインプリメントしている。Prologのパターンマッチング（ユニフィケーション）、自動バックトラッキング、関係データベース機能など、知識システムの基礎言語として好ましい多くの性質を受け継いでいる。かつ知識獲得支援機能など、通常のProlog上で開発された有用な機能を組み入れることが可能である。［石塚86b ］

赤間による確信度付き論理プログラムの最良優先探索アルゴリズムの研究は、このような真理値付き論理式を扱うPrologの推論の効率化としてとらえられる。［赤間87］

知識の真理値として数値を与えると知識の全順序関係が形成されてしまうが、この関係はきつ過ぎることが多い。知識間の強さの半順序関係を扱える言語として、最近、自由ブール代数値をもうPrologが研究されている。

不確実性を最も合理的に論理と結合させているのはNilsson の確率論理である。［Nilsson84 ］［石塚86a ］ここでは論理のモデル論に基づき、複数個の可能世界（解釈が可能な世界）が確率的に同時に存在し得る状態を考える。そして、ある確率の割り当てられた論理式の組があると、対応する可能世界の確率的存在が可能であるときのみ、論理式の組は整合している（矛盾しない）と考える。

たとえば、Ａの確率がｐ（Ａ）、Ａ⊃Ｂの確率がｐ（Ａ⊃Ｂ）と与えられたとすると、Ｂの確率ｐ（Ｂ）は

$$p(A \supset B) + p(A) - 1 \leq p(B) \leq p(A \supset B)$$

の範囲でなければならなくなる。このようにして論理的帰結を（確率付きで）定めることができる。帰結の確率を一意に定める一つの方法は、可能世界はエントロピ

ーが最大になるように確率分布をとると仮定することである。論理式の数が多数になったときの計算量が非常に大きいなど、問題点が多く、システム化はされていない。しかし、論理と確率との合理的な結び付きを図る上で重要な考え方である。

Bundy は集合の要素に確率量を分配し、論理式を集合の部分集合に対応させ、集合演算により確率を伴う推論法を示している。（Incidence 計算法と称している）［Bundy84 ］しかし、利用に際してはNilsson の確率論理以上に問題点が多い。

後に知識の不完全性の扱いについて記すが、不確実性と不完全性の相互関係も、今後明らかにしていくことが必要な課題である。

Richは不確実性の扱いの視点（彼女の場合はMycin の方法をとっている）から、ディフォルト理論も考え得ることを示している。［Rich83］GinsbergはDempster-Shafer 理論による不確実性の扱いの視点から、非単調推論法を示している。［Ginsberg84］

これらの方法では、不完全な知識は 1.0未満の確実度であるとみなす。（たとえば 0.95 ：鳥→飛ぶ）不完全な知識系からは一般に矛盾する結論が導かれるが ― たとえば、Tweetyは（鳥なので）飛ぶと、Tweetyは（だちょうなので）飛ばない ― 、矛盾する結論をたとえばDempsterの結合規則により結合してしまい、矛盾の解消を図る。このとき確実度１の結論があると、他の可能性は消えうせる。非単調論理で行われる探索による矛盾の解消のやり方に比べて、制御は簡単となる。

(iv)不完全性

問題の世界を完全に記述することは、実はよく考えると非常にむずかしい。「鳥は飛ぶ」は正しい知識といえるが、よく考えると飛べない鳥もいて、完全な知識ではない。

ある部屋を考え、その中に存在するものは列挙することはできても、存在しないものを列挙することは不可能である。存在しない対象は無数であるからである。知識も（ある問題領域で）正しい知識を列挙し、定義することはできても、正しくない知識を列挙し、明示的に定義することは不可能である。そこで、知識ベースに定義されているものだけが正しく、定義されていないものは正しくないという考え方をとると、便利である。真か偽か何も言及されていないことがらは、偽としておこうという割り切った考え方である。これを閉世界仮説（closed world assumption ）という。Prologのnot 述語は、引数の節に解がないとき真となるが、この実行は閉世界仮説に立脚している。閉世界仮説は、定義されていない知識（不完全な知識の一種）に対して、しばしば適用される考え方である。

古典論理は、定義されている公理（知識）は完全であることを前提としており、正しいと導かれた結論は、後に新しい公理が追加されても変わることはない。この性質を単調（monotonic ）という。「一般に（例外を除いて）鳥は飛ぶ」というような知識を知識ベースに加えることを許すと、非単調推論の性質が出てくる。すな

わち、ある公理系（知識ベース）から正しいと導かれた結論でも、後に新しい公理が加えられると否定される可能性が出てくる。

Mという記号を導入し、Mpは"論理式pは（他の知識と）無矛盾である"（いい換えると、"Mpは他の知識から¬pが導出されないとき真である"）ことを表すとする。このとき、「一般に（例外を除いて）鳥は飛ぶ」は論理式で次のように表現できる。

（∀x）鳥（x）∧M飛ぶ（x）⊃飛ぶ（x）.

{意味：xが鳥で、飛ぶことが無矛盾であるならば、xは飛ぶ}

ここで、次の知識からなる公理系を考える。

（∀x）鳥（x）∧M飛ぶ（x）⊃飛ぶ（x）.

（∀x）ペンギン（x）⊃¬飛ぶ（x）.（¬は否定記号）

鳥（ピッコロ）.

この公理系からは、飛ぶ（ピッコロ）、すなわちピッコロ（ＮＨＫの幼児番組の鳥の名前）は飛ぶが結論される。しかし、後により詳しい情報が得られ、ピッコロはペンギンであると判明したとする。そして、上記公理系に

ペンギン（ピッコロ）.

が加えられたとする。すると、先に得られていた結論の飛ぶ（ピッコロ）は否定され、¬飛ぶ（ピッコロ）が新たな結論になる。これが推論の非単調性の例である。

非単調推論を必要とする不完全な知識を含む形式的扱いとして、非単調論理（non-monotonic logic ）の確立を目指す研究が進められている。代表的なアプローチに、McDermott.Doyle の非単調論理（様相論理的扱いがなされている）［McDermott80 ］［McDermott82 ］、Reiterのデフォルト論理［Reiter80］Mooreの自己認識的論理（Autoepistemic logic ）［Moore 85］、McCarthyのCircumscription ［McCarthy80］（限定するという意味）などがある。これらに関する解説には［松本81］［植田87］［中川87］がある。

McCarthyのCirumscriptionは分っていることだけが存在するという閉世界仮説に近い、コンサーバティブな解釈を採る方法だか、分らないことは存在する可能性があるという積極的な解釈を採る方法として、有馬（ICOT）はAscriptionの研究を行っている。［Arima86 ］

McCarthyは、コンピュータが現在話題になっている「知識」に加えて、「常識（common sense）をもつように飛躍させることが今日の人工知能の大きな課題であり、それには非単調論理が不可欠であると力説している。

McDermott，Doyleの非単調論理、Reiterのデフォルト論理では、先に例に出したように、論理記号Mを導入し、Mpは"論理式pは（他の知識と）無矛盾である"ことを表す。このような知識を含めることによって生じる主要な問題点は、次のよ

うに説明される。｛Ｍｐ⊃￢ｑ，Ｍｑ⊃￢ｐ｝の知識（論理式）が知識ベース（公理系）に存在したとする。このとき導かれる解（定理）には、｛￢ｐは含むが、￢ｑは含まない解｝と、｛￢ｑは含むが、￢ｐは含まない解｝の可能性が生じ、一意に定まらない。このままでは、￢ｐあるいは￢ｑを解としてよいのか否か判断できない。これを不動点（fixed point ）が定まらないといい、この大きな問題の解決に向けて研究が進められている。

知識システムのShell へのシステム化を考えると、ReiterのNormal Default表現によるデフォルト推論以外はまだ解決しなければならない問題が多いと言えよう。ReiterのNormal Defaultでは、ゴール論理式が一つのexteusion に含まれるか否かを見い出す推論手順が与えられている。

仮説推論も仮説を一種の不完全な知識（成立しているかもしれないし、成立していないかもしれない知識）とするものであり、非単調推論も行う。論理を基盤とする仮説推論は、ReiterのNormal Defaultを扱うこともできかつ具体的応用への見通しも良い。［Pool86］［国藤87］従って、不完全な知識を包含する知識ベースへの着実な第一歩であると筆者は考えている。これについては後述する。

なお、フレーム表現などには実用的な形でデフォルト値の機能が取り入れられている。この場合には、推論のたどり方に依存して値を一意に決定するという、実用的な方法がとられている。

不完全な知識とその管理に関する方法にＴＭＳ（Truth Maintenance System）がある。［Doyle79 ］これは不完全で矛盾を含む知識ベースにおいて、信じる知識と信じない知識に分けて、矛盾を排除した世界の知識で推論を行おうとするものである。ＴＭＳの推論はバックトラックを含むが、ＡＴＭＳは横型的探索によりバックトラックを行なわない並列的探索にし、推論の速度を向上させている。［DeKleer86 ］

ICOTでは、矛盾した知識の調整機能を論理プログラミング上で実現し、知識の調節・均衡化と呼んでいる。［北上85］

(v) ファジィ性

"old" といっても50歳以上なのか、60歳以上、70歳以上なのか境界が明瞭でない。集合をその境界を明瞭にしないまま扱う手法にファジィ集合論があり、1965年のZadeh の提唱以来発展を続けている。［Dubois80］

ファジィ集合で表現した論理やルールを用いて行うファジィ推論に関しても多くの研究がある。知識工学に関連する利用法はサーベイ[Whalen 83][Prade 85] に記されている。

先に、筆者らによるPrologとファジィ論理との結合例について簡単に述べた。それ以外はここでは深く立ち入らないことにする。

1.3 不完全な知識を含む次世代知識ベース

あいまいな知識に関する手法を色々とみてきたが、この中で高次人工知能機能をサポートする次世代知識ベースへ向けて、特に鍵となる考え方は不完全な知識であると思う。そのような視点から我々が行っている研究に関連して次世代知識ベースの姿について述べることにしたい。

(i) 概要

我々は、図 1-2のような完全な知識と不完全な知識を含む知識ベースの枠組みを基礎に、高次知能を実現するための知識／データ構造、高次推論機能を考えている。はっきりしない高次知能の実現へのアプローチには、人間の認知心理を見本とするやり方と、従来の論理野拡張としてとらえるやり方が考えられる。我々は論理の拡張としてのアプローチをとっている。従って、図 1-2の知識の表現は、現在は述語論理を基礎にしている。本来は2階述語論理のような高階な論理が必要になるのだが、2階述語論理には完成された体系ないのでがないので、とりあえず1階述語論理を基礎にしている。

不完全な知識には、当面、次のような種類が含まれる。

- o 仮説的な知識
- o 例外を含む知識
- o 矛盾を含む知識
- o 拡大解釈の知識
- o 今後変りうる知識

知識ベースの不完全な部分には、これらの知識をあまりこだわりなく入れることを許す。不完全な知識の表現に際し、知識の変数化は特に重要と考えている。完全な知識の一種として、制約的知識をもうける。

このような知識系を操作する高次推論により、次のような高次知能の実現をねらいとしている。

- o 不完全パターンの照合
- o 仮説推論
- o 発想的設計
- o 学習
- o 常識
- o 発想

以下に、これらのメカニズムについて簡単に述べる。

不完全パターンの照合は、パターン理解に限らず重要な機能である。変数化したパターンの要素を不完全な知識／データとして登録しておき、完全な照合が得られないとき矛盾しない範囲で動員する。

仮説とは真であるかどうか不明な知識／データであり、仮説推論ではこれを動員し

た解の導出、並びに複数の仮説が生成されたときの選択法が研究されている。仮説推論によれば、従来は不確実性の扱いによって進められた判定も、数値的な不確実性に頼ることなく可能になる。仮説は例外をもった知識の一種ともみなせるので、デフォルト推論との関連も深い。

発想的設計については、拡大解釈可能な形のデータ構造をもつ部品を集めたライブラリを構成することが重要である、と考える。拡大解釈の正当性をチェックするものとして、設計ルール等の制約的知識を働かせる。拡大解釈可能なデータ構造にするのに鍵となる技術は、パターンの変数表現である。当面はユーザがあらかじめある部分を変数化したパターン記述を与えることになる。次第に、後述のモデル推論システムやCircumscription のメカニズムのように、高次推論機構がデータ具体値を変数に換えて拡大解釈可能とし、与えられた環境に適合するような変形を行わせることに向かおう。

知識型設計では、探索空間が広くなるので拘束的知識をなるべく有効に使用する方策が重要になる。論理に基づく設計への一般的アプローチにおいて、長澤は拘束条件リダクション法の考え方を提示している。［長澤86a,86b ］これは設計の詳細化を進めるに際し、拘束条件をうまく伝播させることにより探索空間を狭めようとするものであり、他の設計問題に対するアプローチでも採られてきた考え方である。類似の考え方だが、RESIDUE では知識ベースに仮説推論と同じように不完全な知識を許容し、設計問題に当る考え方を提示している。［Finger85］変数化表現された部品知識（拡大解釈し過ぎると誤りになるという意味で不完全な知識）を含むライブラリとの結合により、次世代の創造的設計のパラダイムになると思われる。［Shapiro83 ］

学習では、Shapiro のモデル推論システム（MIS ）［Shapiro83 ］に関係が深い。MIS では正の例が証明できないとき、その例を一般化して登録し、負の例によって特殊化（精密化）していく戦略がとられている。一般化は、ホーン節の定数項を変数項に変えることに行っている。MIS を参考にした方法により、ユーザとの対話を行いながら、不完全部分の知識の修正を行う。不完全な知識が完全になったかどうかはシステムが判断する手だてはないので、人間の判断に頼るものとする。

学習モジュールとして、MitchellのVersion Space ［Mitchell77］の方法、Michalski が色々と行っている概念学習の方法［Michalski84 ］を、不完全な知識を含む知識ベースへ適用していくことも可能である。

常識に関しては種々の考え方がある。今までの知識システム（特にエキスパートシステム）は想定していた事態には答えられるが、想定していなかった事態には対処できなく幅がない、融通性がないといわれる。この壁を超えるものとして、常識の実現法が大きな課題になっている。我々は学習とも関係するMcCarthyのCircumscriptionの考え方に最も関心を寄せている。完全な知識からだけでは答が得られないとき、Circumscribeして生成した知識（一種の不完全な知識）をバックグランドで働かすよ

うな考え方で常識の実現へアプローチする。Ascriptionでは記述されていない事実をやや一般化する方向で生成するので、不完全な知識の操作と同様な高次推論が必要とされると思われる。

発想は論理ではAbduction であり、A→B，Bであるとき、Aを思い浮かべることをいう。我々は、不完全な知識を含む知識ベースの枠組で、発想のメカニズムを次のように考えている。不完全な知識／データとして変数化された拡大解釈可能な形で、発想のたねを登録する。他の推論では制約的知識のもとで　この知識／データを展開するのだが、発想においては制約的知識の一部を除いて知識／データを展開する。そこでLenat のＡＭ［Lenat82 ］のように"おもしろさ"を見い出し、そのパターンに適合するように、逆に制約的知識を変更する。発想は、制約条件を取り払ってchaosの状態を作り出し、役立つパターンを見い出して、新たな制約の枠組み（秩序）を構成することではないかと思っている。探索量は、かなり多くなるので、Lenat のＡＭのようにヒューリスティックスによるガイドを必要とすることになろう。Lenat のＡＭは論理的枠組みとは関係なく構成されているが、以上のようなメカニズムを採ることにより、整理した形で発想にアプローチできるのではないかと考えている。

拡大解釈の知識、制約的知識のもとでの推論という点で、常識と発想のメカニズムに共通点があることは興味ある事柄である。人間の世界でも、あまり常識（制約的知識）がありすぎると発想が生まれない、発想を仕事とする芸術家はよく奇妙な行動をとり、常識が欠けていると思われることなども、このメカニズムを示唆していると思われる。

以上のように我々は、図 1-2の枠組みで高次知能を展望しての諸機能の実現を目指している。

(ii)学習機構をもつ仮説推論システム

次世代知識ベースの有力な形態である不完全な知識も含む知識ベースの一具体形として、我々は仮説推論システムとその学習機構についての研究を行った。［松田87］ここでは次世代知識ベースへ向けての研究開発に資することを期待し、その内容について簡単に記すことにしたい。

不完全な知識を扱う知識ベースの実現に向けて、なぜ仮説推論システムからアプローチしたかというと、次の理由による。

- ■例外をもつ知識、他と矛盾する可能性をもつ知識も仮説として扱える。
- ■Reiterのデフォルト論理のNormal Defaultを扱える。
- ■故障原因の可能性を仮説として扱うことにより、構造に関する知識に基づいた故障診断が実現できる。
- ■使用する可能性のある部品、可能性のある組み立てを仮説と考えることにより、設計問題への応用も拓ける。
- ■ＡＴＭＳ（Assumption-based TMS）より広い範囲の仮説を扱え、ＡＴＭＳは

仮説推論の高速化手法ととらえられる。

図 1-3に基づき、仮説推論システムについて簡単に記すと、事実（完全な知識）の集合をF、仮説（不完全な知識）の集合Hとから知識ベース（ＦＵＨ）を構成する。ある事象を観測Ｏとする。仮説推論システムは知識ベースからこの観測を説明（導出）しようと働く。Ｆからだけでは説明できないとき、Ｈの部分集合ｈも含め、ＦＵｈからＯを説明しようとする。すなわちＦＵｈ⊢Ｏとなるｈを求めるように働く。（たとえば誤動作現象を説明する必要がある。複数個の仮説の集合h_1，h_2が候補となることがあるが、h_1かh_2かを選定するための質問を生成し、追加の観測により仮説を選定する機構も含んでいるのが仮説推論システムである。

論理に基づく仮説推論はワーテルロー大学のGoebelらによる先行研究があるが、我々の研究成果は次の事項である。

■原因と観測を結ぶ診断等の応用向けに強化した知識表現法。含意を含む知識の頭部の扱いを明確にした。

■仮説を含む知識ベースにおける例からの知識の学習機構（仮説を含む知識の帰納推論）

■仮説を含むフレーム型知識ベースの知識の同化・管理機構

これら考案した機構は、Prologのメタインタプリタによって具体的に実現している。

(a) 拡張知識表現と応用

我々の仮説推論システムのための拡張知識表現は次のようになっている。

事実

ｆａｃｔ（ＩＤ，ＮＩＤ，ＣＯＮＤ，知識）

仮説

ｈｙｐ（Name，ＩＤ，ＮＩＤ，ＣＯＮＤ，知識）

ここで

ＩＤ：知識の識別子

ＮＩＤ：その知識を用いると利用できなくなる知識のＩＤ

ＣＯＮＤ：数の大小比較のように、実行可能になった時点でシステムが自動的に評価すべき述語のリスト。その知識が利用可能であるための条件であるとも解釈できる。

知識：変形規則を用いてfact型、又はrule型に変形された論理式

Ｎａｍｅ：仮説の知識が固有にもつ名前。述語であり、その引数は知識の中の変数

これらの説明の前に、仮説推論では矛盾を検出する必要があるため（たとえばＦＵｈは無矛盾の確認）、Prologの基礎となっているホーン論理では不十分であり、

完全な１階述語論理の推論を扱えるようにする必要がある。Prolog上で１階述語論理を扱う（論理否定を扱う、あるいは頭部の選言を扱う）手法としてMeson proof procedureがあり、ここでもそれを用いている。これは知識の対偶（contrapositive）をすべて定義しておき、導出仮定で論理否定の対を見つけて矛盾を見い出すものである。対偶とはＢ：－Ａ（ＡならばＢだ）、￢Ａ：－￢Ｂ（Ｂでなければ Ａでない）といった真偽が一致する関係にある論理式をいう。

さて、我々の知識表現で“ＣＯＮＤ”を導入した理由は観測値間の関係など、直接の観測値と区別した方が都合がよいものを表すためである。“ＩＤ”，“ＮＩＤ”の導入の理由は、頭部に論理和をもつ知識を正しく表現するためである。

以上と複数個の仮説が候補となったとき仮説を検定するための質問生成部を含めて仮説推論システムが実現されている。応用指向の質問機能を支えるものとして次の述語が用意されている。

observable（［リスト］）

仮説検定のためにユーザに質問できる述語をリストに書く。

askable （［リスト］）

論理回路入力のように、直接制御可能な述語（すなわち観測）をリストに書く。

（ＩＣの故障診断のように観測可能な観測（ＩＣの出力）があまりなく、制御可能な観測（ＩＣの入力）を変化させたときの観測が検定に役立つ場合の表現力強化のため）

これらによる簡単な応用例を示している。

仮説推論のプロセスでは仮説の知識を用いるたびに、その仮説が矛盾を引き起こさないことを調べており、かなりの手間になる。あらかじめ矛盾を引き起こすことが判明している仮説の知識の集合をプリコンパイルして、効率化を図るのがcontradictの定義である。仮説推論プロセス中でこの知識を自動生成していく機構も実現している。

(b) 仮説を含む知識の学習機構

仮説を含む知識ベースに対して、図 I-4に示すように２種の学習機構を考案した。

第１の学習機構が与えられた例（診断例など）から仮説を含む知識を形成するものである。

鍵となっている手法は、仮説を含む場合に対する拡張最小汎化の考え方である。およその学習の手続きは次の通りである。（なお、

defgentype(p(ord)),defgentype (p(dis)),defgentype (p(any))

とは汎化する述語ｐの引数のタイプ宣言であり、それぞれ数のように順序がある引数(ord)、排他的な実体(dis)、任意(any)を表す。

学習手続

1）矛盾がない結果となっている例の最小汎化

p (a) : − cとp (b) : − cのとき

fact (ID, [ ], [a≦x≦b], (p (x) : − c))

if defgentype (p (ord))

fact (ID, [ ], [x＝a ; b], (p (x) : − c))

if defgentype (p (dis))

fact (ID, [ ], [ ], (p (x) : − c))

if defgentype (p (any))

2）矛盾の結果となっている例の拡張最小汎化

q : − cと¬q : − cのとき

hyp (name1, ID1, [ ], [ ], (q : − c))

hyp (name2, ID2, [ ], [ ], (n (q) : − c))

3）混在の結果となっている例の拡張最小汎化

q, r : − c q, s : − cのとき

fact (ID1, [ ], [ ], (q : − c))

hyp (name1, ID2, [ ], [ ], (r : − c))

hyp (name2, ID3, [ ], [ ], (s : − c))

4）過度の汎化になったとき、反例による入力例の分割

入力例を分割して上記1）～3）を適用。分割数はなるべく少ないものから試みる。帰納された知識はexclusive ORの関係で知識ベースへ。

これは具体例から反例を含まないように最小汎化を進めていくバージョン空間でのUp date-S アルゴリズムに相当している。これとはやや異なる学習手続も考案し、実現している。

学習の結果、他の概念と区別するために必要最小限の記述は事実（fact）として、その他の（常に現れる訳ではない）現象の記述は仮説（hyp）として生成される。若干の学習例により、動作を確認している。

バージョン空間での学習と比較すると、次の点が優れているといえる。

□矛盾を引き起こす可能性がある場合も学習できる。

（仮説の知識を入れられるため）

□NIDのための概念間の和集合がとれる。

知識獲得に関する第2の機構は、階層をもち知識の多重継承を行うフレーム型知識ベースで、仮説も含むケースについての知識の同化・管理機構である。これについては以下の機構を考案、実現している。

(1) 仮説を含むフレーム型知識ベースの無矛盾性を維持した知識同化機構

（入力知識の矛盾性、冗長性を判断し、事実とするか仮説とするか、あるいはそ

194

れらとは別の冗長知識ベースに入れるかを判断して、知識ベースに取り込む）関連して、仮説を含む知識ベースについての冗長性判別アルゴリズム、矛盾の原因となる知識を見い出すアルゴリズムを考案、実現している。後者は、Shapiro のデバッグアルゴリズムに基づき、仮説を含む場合への拡張となっている。

(2) 知識の削除時の管理機構

ある知識を削除すると、それまで冗長であった知識が冗長でなくなることがある。そこで冗長な知識ベースに入っている知識の冗長性を見直し、冗長でなければ知識同化アルゴリズムにかけるようにしている。

(3) 知識の継承の制御

入力からユーザがあるフレームへの継承を望まない知識があることを知ることができる。その望まない知識を発見し、その知識の本体にｎｏｔ（フレーム名）を加えることで、継承を打ち切るアルゴリズムを考案し、実現している。これはShapiro のテバッグアルゴリズムを拡張したものである。

以上、高次人工知能機能を支援すると次世代知識ベースへ向けての着実な第一歩であるとの認識のもと、我々が行った「仮説推論システムとその学習機構」の概要を記した。ICOTにおける次世代shell の研究開発の参考になれば幸いである。

〔参考文献〕

[石塚86a]　石塚，松田：不完全な知識環境下での高次推論，「知識システム方法論・夏期シンポジウム報告書，富士通国際研，1986.

[石塚85]　石塚：曖昧な知識の表現と利用，情報処理，Vol.26，No.12，pp.1481-1486，1985.

[Lenat84]　Lenat，D.B.: The Role of Heuristics in Learning by Discovery-Three Case Studies，Machine Learning(R.S. Michalski et al. Eds) Chap. 9.，Springer-Verlag，1984.

[嶋田87]　嶋田：ヒューリスティクスにより発見を実現するシステム，人工知能学会誌，Vol.2，No.1，pp.52-61，1987.

[坂上82]　坂上：画像処理における反復演算の応用，情報処理，Vol.23，No.7，pp.641-650，1982.

[Erman80]　Erman，L.D. et al.: The Heasay- II Speech Understanding System: Integrating Knowledge to Solve Uncertainity，Computing Surveys，Vol.12，No.2，pp.213-254，1980.

[石塚83a]　石塚：不確かな知識の取り扱い，計測と制御，Vol.22，No.9，pp.774-779，1983.

[Shortliffe76]　Shortliffe，E.H.: Computer-Based Medical Consultations: MYCIN，American Elsevier，1976.

[Buchanan 84]　Buchanan，B.G. and Shortliffe，E.H.: Rule-Based Expert Systems- The MYCIN Experiments of the Stanford Heuristic Programming Project，Addison-Wesley，1984.

[Duda76]　Duda，R.O.，Hart，P. and Nilsson，N.J.: Subjective Baysian Methods for Rule-Based Inference Systems，NCC，1976.

[Dempster76]　Dempster，A.P.: Upper and Lower Probabilities Induced by a Multivalued Mapping，Annals of Math. Statistics，Vol.38，pp.325-339，1967.

[Shafer76]　Shafer，G.: A Mathematical Theory of Evidence，Princeton Univ. Press，1976.

[石塚83b]　石塚：Dempster & Shafer の確率理論，電子通信学会誌，Vol.66，No.9，pp.900-903，1983.

[Ishizuka85]　Ishizuka，M. and Kanai，N.: Prolog-ELF incorporating fuzzy logic，New Generation Computing，Vol.3，pp.479-486，1985.

[Lee72]　Lee，R.C.T.: Fuzzy Logic and the Resolution Principle，J. ACM，Vol.19，No.1，pp.109-119，1972.

[石塚86b]　石塚，松田，金井：Prolog-ELF上での知識同化管理機構，情処全大，1P-7，1986．3．

[赤間87]　赤間：確信度付き論理プログラムの最良優先探索アルゴリズム，人工知能学会誌，Vol.2，No.1，pp.85-91，1987．

[Nilsson84]　Nilsson，N.J.: Probabilistic Logic，Technical Note 321，SRI International，1984．Also，Artificial Intelligence，Vol.28，pp.71-87，1986．

[Bundy84]　Bundy，A.: Incidence Calculus: A Mechanism for Probabilistic Reasoning，FGCS-84，Tokyo，1984．

[Rich83]　Rich，E: Default Reasoning as Likelihood Reasoning，AAAI-83，1983．

[Ginsberg84]　Ginsberg，M.L.: Non-monotonic Reasonig using Dempster's，Rule，AAAI-84，1984．

[McDermott80]　McDermott，D．and Doyle，J.: Non-monotonic logic　I，Artificial Intelligence，Vol.13，pp.41-72，1980．

[McDermott82]　McDermott，D.: Non-monotonic logicⅡ：Non-monotonic model logic，J．ACM，Vol.29，No.1，pp.33-57，1982．

[Reiter80]　Reiter，R: A logic for default reasoning，Artificial Intelligence，Vol.13，pp.81-132，1980．

[Moore85]　Moore，R.C.: Semantic consideration on non-monotonic logic，Artificial Intelligence，Vol.25，pp.75-94，1985．

[McCarthy80]　McCarthy，J.: Circumscription: A form of non-monotnonic formalizms，Artificial Intelligence，Vol.13，pp.27-89，1980．

[松本81]　松本：Default reasoning と非単調論理，電子通信学会誌，Vol.64，No.3，pp.313-316，1981．

[相田87]　相田：デフォルトを用いた推論と非単調論理，人工知能学会誌，Vol.2，No.1，pp.6-13，1987．

[中川87]　中川：論理＋サーカムスクリプション＝常識推論，人工知能学会誌，Vol.2，No.1，pp.14-21，1987．

[Arima86]　Arima，J.: A form of conjecturarl reasoning on equivalence－ascription，ソフトウェア科学会大会，1986．

[Pool86]　Pool，D.L.，Aleliunas，R．and Goebel，R.: Theorist : A logical reasoning system for defaults and diagnosis，In Knowledge Representation，(Cercone，N.J．and McCalla，G．eds) Springer-Verlag，N.Y.，1986．

[國藤87]　國藤：仮説推論，人工知能学会誌，Vol.2，No.1，pp.22-29，1987．

[Doyle79]　Doyle，J.: A truth maintenance system，Artificial Intelligence，Vol.12，pp.231-272，1979．

[DeKleer86] DeKleer, J.: An assumption-based TMS, Artificial Intelligence, Vol.28, pp.127-162, 1986.

[北上85] 北上, 國藤, 宮地, 古川：論理型プログラミング言語Prologによる知識ベース管理システム, 情報処理, Vol.26, No.11, pp.1283-1295, 1985.

[Dubois 80] Dubois, D. and Prade, H.: Fuzzy Sets and Systems: Theory and Applications, Academic Press, 1980.

[Whalen83] Whalen T. and Schott, B.: Issues in Fuzzy Production Systems, Int. J. Man-Machine Studies, Vol.19, pp.57-71, 1983.

[Prade85] Prade, H.: A Computational Approach to Approximate and Plausible Reasoning with Applications to Expert Systems, IEEE Trans. PAMI, Vol.PAMI-7, No.3, pp.260-283, 1985.

[長澤86a] 長澤, 古川：拘束条件リダクション法を用いた機械設計計算支援システム, 情報処理学会論文誌, Vol.27, No.1, pp112-120, 1986.

[長澤86b] 長澤：設計支援システムのための知識表現と推論機構に関する研究, 九州大学, 1986. 8.

[Finger85 [Finger, J.J., Genesereth, M.R.: RESIDUE--A Deductive Approach to Design Synthesis, Stanford Univ., Dep. of Comp. Sci., Report No.STAN-CS-85-1035, Jan, 1985.

[Shapiro83] Shapiro, E.Y.: Algorithmic Program Debugging, The MIT Press, 1983.

[Mitchell77] Mitchell, T.M.: Version Space: A Candidate Elimination Approach to Rule Learning, 5th IJCAI, 1977.

[Michalski84] Michalski, R.S.: A Theory and Methodology of Inductive Learning, Machine Learning (Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. eds.),Springer-Verlag, 1984.

[Lenat82] Lenat, D.B. AM: Discovery in Mathematics as Heuristic Search, in Knowledge-Based Systems in Artificial Intelligence (R. Davis, D.B.Lenat), Patr 1, McGraw-Hill, 1982.

[松田87] 松田哲史：仮説推論システムにおける拡張知識表現と学習機構, 東大電子工学専攻修士論文, 1987, 2.

2. 時間と空間の表現

2.1 時間の表現

2.1.1 はじめに

エキスパートシステムの構築にあたり、知識をいかに表現し、利用するかが大きな研究課題である。プロダクションルールやフレームなどの知識表現法などが研究され、それに基づくツールが開発されてはいるが、時間の表現を明確にできるツールはほとんどない。

エキスパートシステムにおいて、時間に関する情報を表現し、推論する必要性はいろいろなところに見られる。例えば、予測問題は、現在の事実をもとに因果関係の知識を使って将来生じる事象を推定するものであるし、故障診断は、現在の事実から、過去に生じた事象を推定するものである。また、並列プロセスが仕様通りに動作をしていることの検証に関する推論が必要になる。

時間の概念をどのようにとらえるかについては哲学的課題としても古くから研究されており、ＡＩにおいても時間の存在論（ontology）が重要であることが指摘されている。［田中86］しかしながら、ここでは問題点を簡単化し、ある時点で生じた事象（event ）とある時点で成り立つ事実（fact）をいかに表現するか、事象間・事実間・事象と事実間の因果関係をどのように表現するか、それらの知識を使ってどのように推論するか、などを主に考える。例えば、「階段から落ちて骨折した」ケースを考えると、「階段から落ちる」という事象（出来事）によって、「骨折する」という事象が引き起こされ、「骨折している」という事実（状態）が引き起こされたのである。事象とはある時点に発生するものであり、事実は一定期間持続する状態である。

時間に関する表現／推論について、従来、その明示的な取り扱いは十分ではなく、多くは一階述語論理やプロダクションルールによって暗黙のうちに取り扱われていたにすぎない。これは、時間に関する知識表現／推論方式が必ずしも十分なものではないことに基づくものであると推定される。知識エンジニアにとって、(i) 時間的要素を含む知識と(ii)時間的要素を含まない知識と(iii) 時間に関する常識的知識を同じ形式で知識ベースを構築しなければならず、その負担は非常に大きいのである。特に、論理回路のシステムでは並列プロセスの同期の問題があり、法律のシステム遡及効果（効果が過去にさかのぼって発生すること、すなわち、原因と結果の時間的前後関係が逆になること）の問題がある。これらの知識は時間に関する柔軟な表現ができ、しかも、直感に反しない推論が行われるツールがあれば、より容易に知識ベースが構築できるであろう。

ここでは、時間の知識表現／推論の必要性を述べ、関連研究を概観し、エキスパートシステム構築ツールに時間に関するサポート機能を導入するための研究課題について考察する。

2.1.2 時間の表現

時間の表現で問題となるのは、(i) 時間を点の集合と考えるのか、一定幅の期間（区間）の集合と考えるのか、(ii)点の集合と考えたときに、離散的なものか、連続的なものか、(iii) 線形なものか、分岐するものか、などのとらえかたがあることである。例えば、「その手紙を12時に見つけた」というとき、「12時」は特定の時点を表すものであるが、「その手紙を昨日見つけた」というとき、「昨日」は一定の幅を持つものである。人間は時点と時間幅（ここでは区間という）をあまり区別せずに用いているのであって、長い年月を考慮しているときは「昨日」は時点と考えても良いかもしれないが、日常の感覚では「昨日」は区間である。また、時間情報の与えられ方にもいろいろな種類があり、事象間の前後関係として与えられることもあるし、事象の絶対時間が与えられることもある。どのような与えられ方においても、人間が直感として有する時間の性質に反しないことが必要である。

このような、時間情報の態様の多様制に対応し、時間の推移性や連続性などの直感的な性質に反しないものであり、かつ、完全性や健全性などを保証された形式的な取り扱いは、以外に困難である。

ここではエキスパートシステムにおいて、従来、用いられてきた時間表現について概観し、その問題点について述べる。この中には、時間のアドホックな表現と形式的な表現、基礎理論と知識表現言語などがあるが、特に区別せずに紹介している。

(1) 手続き／プロダクションルール

時間経過に伴う状況の変化を、手続きによってシミュレートすることができる。各時点における状況は、メモリ状態の集合（データベース）として表現される。例えば、SIMULAやSmallTalk などのオブジェクト指向言語では、発生した事象を外部からのメッセージの到着と考え、その効果を手続きの起動と考えれば、オブジェクトの内部状態がその時点の状況（成立する事実の集合）を表すことになる。SIMULAでは、オブジェクトの平行動作の記述のために、

```
hold(T)    /*時間Tだけ実行を停止する*/
```

などのメカニズムが用意されており、オブジェクトの同期のシミュレートに用いられる。

プロダクションシステムでは因果関係を

```
IF  原因  THEN  効果
```

の形に記述しておいて、発生した事象をワーキングメモリに書込み、それによってルールを駆動することにより、ワーキングメモリの内容がその時点の状況を表すことになる。

このような手続きによる時間のシミュレーションは、実際の時間経過にそった推論しか行うことはできず、また、時間に関する推論と時間に無関係の推論との区別がつかないという問題がある。特に、真理維持機構（Truth Maintenance System）

の組み込まれたシステムにおいて、ワーキングメモリからの事実の消去が、仮説が成立しなくなったためなのか、時間の経過によって成立しなくなったのかが区別できないため、知識エンジニアはデバッグの際に負担が大きい。［McD82 ］

［篠原86］では、時間関係と因果関係を融合したプロダクションシステムを提案している。このシステムでは、時間の表現単位はAllen の時間論理（後述）と同じ「区間」を採用しており、知識ベースは、事実データ、時間関係データ、状態内ルール、時間依存ルールからできている。「事実データ」は、(i) 述語Pと(ii)値Vと(iii) その値が成立する期間のリストIと(iv)成立しない期間のリストnIと(v) 2つの制御情報からなる（成立期間と期間と不成立期間のいずれにも属さない期間では、値は不明である）。例えば

（色　信号　赤　（T1 T3 ）　（T2）　NIL NIL ）

は「区間T1とT3で信号は赤であり、T2で赤でない」ことを表わす。「時間関係データ」は区間と区間の関係であり、Allen の指摘した関係を用いる。例えば、

(TIME-RELATION T1 T2(BEFORE))

は「T1はT2より前である」ことを表わす。「状態内ルール」は、時間については恒等的に成立するルールであり、期間の絞り込みを行う。

(IF((P V))((Q W))

は以下のルールと等価である。

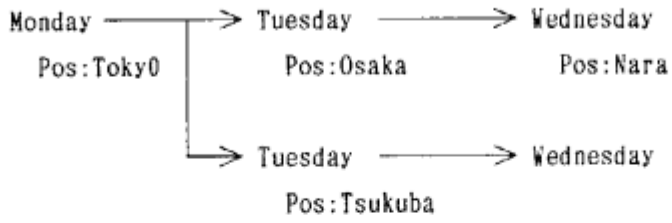
(IF((P V *I *nI ? ?))((assert(Q W *I () ? ?))))

「時間依存ルール」は以下のように、対象の時間的な状態を変更したり、区間を生成したりするものであり、事象の表現を行うものである。

```
(def-rule  rule-1  scenario-1
          (漏洩　(冷却系)　t *I *nI t 2) )
          ((make-interval *I2)
           (assert-time-relation(*I *I2)(during) )
           (assert （緩む（冷却系）t *I2 () t nil))) )
```

推論は、与えられた状態を出発点としてルールを起動することで行われ、バックトラックのメカニズムが備わっている。推論方向には、原因探求型と波及予測型の双方向があり、状態内ルールを時間依存ルールより優先的に適用する。

エキスパートシステム構築ツールＡＲＴでは、view point（ある観点から見た事実の集合）の連鎖によって時間を表現する。推論は基本的にはプロダクションルールの実行によるものであるが、動的にview pointを生成でき、適用できるルールが２以上あるときに連鎖が分岐する。view pointは上位のviewpoint からの情報の継承がある。以下に月曜日に東京を出発したトラックが所在を変更して行く様子をview pointで表わしたものを示す。



(2) 述語論理

述語論理においては、時刻（または世界、状態）を表わす引数を論理表現の中に導入し、時間に関する様々な性質（時刻の前後関係の遷移性など）をその引数の大小関係でおきかえて推論を行うのが最も基本的な方法である。例えば、時刻t1にbox がpos1の位置にあることは、

at(t1,box,pos1)

のように表わすことができる。時刻t1の状況は、t1を引数に持つ言明、または、t1より前の時点t0を引数に持ち、t1の時点で否定されていない言明の集合である。事象moveは以下のように、その効果によって定義される。

at(T,OBJ,P1)& move(T2,OBJ,P2)->at(T2,OBJ,P2)& ～at(T2,OBJ,P1)

［中川］では、多重世界記述機能をもつ論理型言語URANUS［中島85］で時間記述を行う。URANUSでは、１つの時刻を１つの世界に割り当て、世界の間に情報の継承ができることを利用して、その世界での変化分を記述する。定理証明では、どの世界から証明を試みるかを指定できるので、例えば、

(within (T2 T1 T0) (P *X))

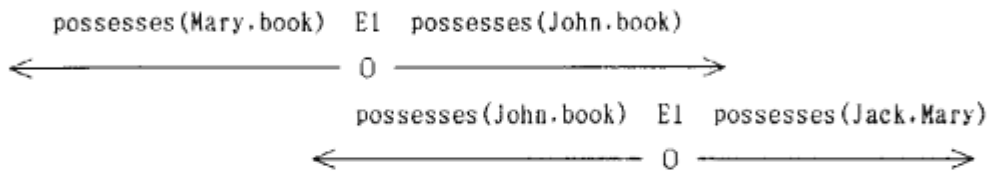
なる記述で、まず、T2の世界で(P *X)の証明を試み、失敗したときには、T1の世界で試み、さらにT0で試みる…というように実行される。時間の流れを世界の連鎖としてとらえる点ではART のビューポイントと同じである。

［Kow ］では論理型言語（Prolog）に基づいたevent calculusの手法について提案している。event calculusとは、事象（event ）を事実の変化として定義し、事象の列から生じる変化を推論するものである。従来の手法では、事実の変化をデータベースへの情報の追加や削除で行うものであったが、［Kow ］では、情報は追加されることがあっても、削除されることはない。「Maryがbookを Johnに与えた」という事象E1の結果は、possesses(Mary,book)をデータベースから削除するのではなく、「possesses(Marry,book) の期間を終了し、possesses(John,book)の期間を開始する」と考える。

initiates(e possesses (X Y))
　　if Act(e gives)& Recipient(e x) & Object(e y)
terminates(e possesses(x y))

if Act(e gives)& Donor(e x) & Object(e y)

その後、「Johnがbookを Jackに与えた」という事象E2があると、possesses (John.book) はE1とE2の間に成立していたことが推論される。



STRIPS [Fik71] は時間に関する問題解決プログラムである。問題空間は、

初期世界モデル……well formed formula(wff)の集合であって、世界の状況を表わす。

オペレータの場合…wff で表わされた前提条件と操作の効果を表わす。

ゴール条件……wff で与えられた最終目標。

の3つで定義される。例えば、3つのオブジェクトがあり、ロボットがこれを1箇所に集めることを考える。初期の世界モデルM0は、

ATR(a), AT(BOX1,b), AT(BOX2,b), AT(BOX3,c)

で表わされる（ATR(a)はロボットがaの位置にいることを表わす）。オペレータpush(k,m,n,)（オブジェクトkをmからnへ押す）とgoto(m,n) （ロボットがmからnへ移動する）は以下のように記述される。

```
push(k,m,n)
  precondition: ATR(m) & AT(k,m)
  delete list:  ATR(m),  AT(k,m)
  add list:     ATR(n),  AT(k,n)
goto(m,n)
  precondition: ATR(m)
  delete list:  ATR(m)
  add list:     ATR(n)
```

ゴール条件

(exist x)[AT(BOX1,x) & AT(BOX2,x) & AT(BOX3,x)]

が与えられると、述語論理の導出原理によって、ゴールを満たす世界モデルが生成される。

(3) 様相論理としての時相論理

時相論理とはその真偽値が時間とともに変化する論理体系である [Wol81] [Mos84] 。時相論理の論理体系では、世界が複数個存在し、時間に伴う真偽値の

変化を異なる世界での真偽値の相違としてとらえるものである。この論理においては

○p	pは次の時刻で真である。
□p	pは以後ずっと真である。
◇p	pは以後真になることがある。
p UNTIL q	pはqが成り立つまで真である。

などの様相オペレータが使われるが、○（next operator ）は時間が離散的な点の集合であることを仮定している。論理回路の検証などへの応用が見られる。

Tokio ［青柳85］はLinear Time Temporal Logic に基づく論理型言語でPrologを包含した言語である。Tokio では、時間は離散的時刻の列と考え、２つの時刻の間をインターバルと称する。

```
P:-Q.R.      /*同一インターバルでの実行*/
P:-Q && R.   /*インターバルの前半でQを後半でRを実行*/
P:-@R.       /*Pの次の時刻でRを実行*/
```

などの実行制御ができ、バックトラックは、Prologと同じものと、インターバルの決め方に対するものがある。

Temporal prolog ［桜川85］は、時相論理に基づいた並列論理型言語を提案している。Temporal Prolog の名が示すように、pure Prolog を文法的に包含していることを特徴とする。

```
temp(X)>100=>dangerous(X).
dangerous(X)=>□alarm.
```

Templog ［米崎84］は区間の記述ができる時相論理である。Prologで実現されており、データベースの更新により、推論が行われる。以下に「学生時代に夏休みには山登りをしたものだ」の記述を示す。

```
<学生->～学生><夏休み->～夏休み>◇山登り
```

(4) AIのための時相論理

様相論理では、事象と事実の区別をしていないので、現実世界の記述には、表現能力が不足している。そこで、AIのための時相論理が提案されている。これらの論理では、事象や事実の性質に関する多くの公理が導入されている。

① McDermottの時相論理

McDermott は時間を一階の多ソート論理の枠組みで表現する。彼は時間をstate （瞬間的な世界）の集合であるものとする。すべてのstate 間には時間的な前後関係に相当する順序（-<）があり、s1がs2より前であることを（s1-<s2）と表わす。任意の３つの時点s1, s2, s3において遷移関係

$$(s1 -< s2) \ \& \ (s2 -< s3) \Rightarrow (s1 -< s3)$$

が成り立つ。state は密（dense ）であり、未来に分岐する（left linear ）。

任意のstate s において時刻d(s)が存在する。state と時刻には

s1-<s2 <-> d(s1)<d(s2)

なる関係がある。

ある事実（fact）が成立するか否かはstate によって異なる。ここではfact を、それが成立するようなstate の集合ととらえる。例えば、

(ON A B)

はAがBの上にあるようなstate の集合である。また、event を、それが発生している区間（state のtotally ordered,convex set）ととらえる。

因果関係には、event が他のevent を引き起こすもの（ecause）と、event がfactを生じるもの（pcause）がある。

(ecause p e1 e2 rf i)

(pcause p e q rf i r1)

は、それぞれ、「pが区間iの間にfalse にならなければ、event e1の後にe2が時間送れiで発生する（rfは0から1までの値をとり、e1の区間のどの時点から測定するかを示すものとする）」と、「pが区間iの間にfalse にならなければ、event e の後にfact qが時間遅れiで成立し、r1の間成立し続ける（rfはecauseと同じ）」ことを表わし、その性質が公理として与えられている。状態が継続するのを表わすのに、persistence の概念が導入されている。

変化が連続的におこるもことを表わすのにfluentの概念を用いる。fluentはその値が時間とともに変化するものであり、

(vtrans v r1 r2)

はvの値がr1からr2へ変化するすべての機会からなるevent を表わす。

あるagent がある行為（action）をすることによって、event が発生する。（PUTON x y ）のような多くのactionはその効果をpersistence として公理化することができる。

② ALLENの時論理 [Allen84]

Allen の時相論理は、McDermott と同様に、一階の多ソート論理に基づいている。彼は時間の表現を区間を基に行っている。時間の表現に区間を導入したときには、区間と区間の前後関係だけでなく、区間の重なり関係も記述する必要がある。主なものに

before, equal, meets, overlaps, during, starts, finishhes

など13種類がある。これらの間には例えば以下の関係が成立する。

before(t1,t2) & before(t2,t3)->before(t1,t3)

meets(t1,t2) & during(t2,t3)->

(overlaps(t1,t3) or during(t1,t3) or meets(t1,t3)

ある事実pが区間iで成立することは

hold(p,i)

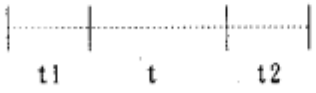
と表現される。

事象はMcDermott と異なり、primitive として導入され、その前後の事実の変化として表わされる。例えば、AをB地点からC地点へ移動する事象を

change-position(A,B,C)

とすると、これは以下のように表わされる。

```
occur(change-position(A,B,C),t)=>
        for some t1,t2
          meets(t1,t) & meets(t,t2) &
          holds(at(A,B),t1) & holds(at(A,C),t2)
```



ここで、ある事象（e）が区間（t）に発生したことを

ocuuur(e,t)

と表わし、tの任意の部分区間（s）について

～occur(e,s)

であるとしている（すなわち、事象はある区間をかけて生じるものであって、それ以上分解できない）。

occur(e,t) & in(t',t)=> ～occur(e,t')

因果関係はecauseによって表現する。t1に生じた事象e1がt2に事象e2を引き起こすことを

ecause(e1,t1,e2,t2)

と表わすとき、

occur(e,t) & ecause(e,t,E',T')=>occur(E',T,).

が成立する。原因となる事象が結果となる事象より後に生じることはありえないから、

```
ecause(e,t,e',t')=>
  in(t',t) or before(t,t') or meets(t,t')or
  overlaps(t,t') or equals(t,t')
```

である。

事象以外に「プロセス（process ）」なる概念も導入され、事象とは区別される。例えば、"I am reading a book." なる状態は事象と似ているが、その任意の部分区間についても成立する点で異なる。従って、あるプロセス（p）

が区間（ｔ）で発生したことを

occuring(p,t)

と表わすことにすると、ｔの任意の部分区間（ｓ）について

occuring(p,s)

である。すなわち、

occuring(p,t)=>in(t',t) & occuring(p,t').

あるagent によってある事象が引き起こされること(action)はacauseによって表わされる。例えば、

acause(agent, change-pos(object,source-position,goal-position))

(5) 定性推論

自然界に存在する多くの物理法則は微分方程式などで記述できる。例えば、物体が受ける力と加速度の関係の式

■*(d(dx/dt)/dt)=F

を解くことによって、空中に投げたボールがやがて落下することを予測できる。定性推論とは、このような物理的な挙動等を数式を直接使わずに推論するものである。定性モデルにおいては、パラメータの重要な値は数値としてではなく、｀－／０／＋｀などのように記号的にlandmarkとして与えられ、変化状態は｀単調増加／減少／一定｀などの定性微分の形で与えられる。

(6) その他の時間の表現と推論

並列プロセスの同期に関する記述には、csp, petri net, ccs などがある。

2.1.3 時間に関する課題

知識表現ツールに時間を導入するための課題について述べる。以下の点は［Sho 85］を参考にしている。

(1) 区間の表現

時間表現の基本は時間を点の集合と考えるものである。しかし、事象はある特定の時刻だけに発生するのではなく、一定幅の区間（interval）をかけて生じるものであるから、それ以外に区間を表現できることが必要である。［All 84］や［McD82 ］の時相論理では、区間を持った事象の表現が可能である。年月日時分秒などの時間の単位を導入したときに、点を表わすのか、区間を表すのか、また、それらの関係はどうかなどの厳密な議論が必要である。

(2) 連続的変化の表現

世界の状況は急激に変化するのではなく、徐々に変化するものが多い。数値の連続的変化を表現できることが必要である。これを行うには、ある動作が部分動作に分解できることなどについての推論ができることが必要である。［All84 ］［McD82 ］ではある程度の表現が可能である。

(3) フレーム内問題

時間の流れを状況の変化と考えるとき、前の状況から何か変化し、何が変化しないかをすべて記述するようなこと（フレーム問題）を避けることが必要である。情報の継承を利用した多重世界によるシステムや、プロダクションシステムによるワーキングメモリの書き換えではフレーム問題は問題とならない。しかし、これは「空間の問題を時間の問題に置き換えただけであって、フレーム問題の解決ではない」との指摘もある。

(4) 並列動作

同時に２以上の事象が起こる（すなわち、発生時間が重なる）ことが表現できなくてはならない。

(5) フレーム間問題

並列動作の記述を許したとすると新たな問題が発生する。異なる効果をもつ２つの事象が生じたとき、STRIPSなどのオペレータ記述では他の事象の存在を記述していないため、どちらの効果を先に計算するかによって結果が異なることがある。（actionは結合的ではない）。これを避ける手立てが必要であるが、オペレータ記述のprecondition部に他のオペレータが存在しないことなどの条件を含ませると、フレーム内問題になってしまうことに注意しなければならない。

(6) 抑圧された因果関係

①原因なく事実が成立しなくなること（自然死）、②効果が遅延して起こること、などを表現できなくてはならない。［McD82 ］では遅延についての記述が可能である。前述のように、法律では効果の遡及という問題があるが、これはTruth Maintenance のメカニズムや時間的なバックトラック［新田86］などで対処できるものである。

(7) 可能な世界

未来については可能性がいくつかある（例えばあるactionをとったときと、とらないときの仮定の世界）ことを表現できることが必要である。情報の継承によるview pointや多重世界のシステムでは、世界を分岐させることによって実現している。［McD82 ］では時間が将来にわたって分岐についての推論が可能である。

(8) 世界間の認識問題

未来の可能性が２つ以上あっても、その状況の大半が共通するとき、コピーをいくつも作るのを避けることが必要である。

(9) modularity

知識の改定に対する波及効果を最小にすることが必要である。

(10)計算の枠組み

実現のための効率的なアルゴリズムが必要である。また、推論結果についての説明機能についても考慮する必要がある。

(11)厳密な論理

検証問題などでは、時間の推論に関して正当性や完全性などの保証があることが望ましい。

2.2 空間の表現

2.2.1 はじめに

設計問題や図面認識、画像認識などにおいては、空間情報をいかに表現し、推論するかが問題となる。例えば、３次元形状の設計においては、(i) 拘束条件の表現とそれを満たす設計の生成、(11)対象物体の表示などで、空間の表現が必要となる。自由曲面の設計においては、拘束条件は点の３次元座標値や３次元曲線の式の集合として与えられ、これらを通る曲面を設計することになる。機械部品の設計においては、基本形状（円錐や角柱など）を基に、それを削ったり、結合したりして、仕様を満たす物品を設計する。また、画像や図形の認識においては、信号データから円らしい部分や長方形らしい部分などを抽出し、それらを記号に置き換えて、記号間の拘束条件から認識を行うことが行われている。

これらの問題においては、コンピュータが苦手とするパターン情報を扱うという点でまだ多くの困難な問題をかかえており、従来はエキスパートシステム構築ツールの対象外であった。すなわち、これらの問題は、従来はFortran やLispなどで書かれたソフトウェアパッケージにより解決されており、エキスパートシステムのツールはこれらの言語とのインターフェースをとるというのが唯一の解決法である。しかし、この方法では、パッケージの信号処理／数値演算アルゴリズムと、ツールの記号演算が明確に分離されており、きめの細かい推論が行えない可能性がある。この点で、ツールが空間情報を扱う機能を有する意味がある。

2.2.2 空間表現に関する課題

空間情報を扱うには数値演算が不可欠である。空間表現の研究がいまだに十分でない現在、空間表現機能をエキスパートシステムに取り込むよりも、強力な数値演算機能を備えたツールを開発する方が現実的である。ツール開発のための課題を以下にあげる。

(1) 基本的な形状等のデータ構造

画像処理においては、信号レベルでは行列演算などの数値計算が行われ、記号レベルではモデルとのマッチングが行われる。その際、数値パラメータと記号をつなぐ中間的なデータ構造が必要である。また、形状設計においては、拘束条件を入力すると、それを、中間的なパラメータに変換し、形状モデルによって目的の物を設計することが行われている。その際、ユーザによって異なる表現形式を採用されたのでは、ツールとして組み込むことはできない。基本的な形状等における標準的な

表現形式であって以下の条件を満たすものの開発が最大の課題である。

i）あいまいな情報の表現ができること

例）　「ほぼ長方形に近い」などの表現、二つの図形の近似度

ii）他の表現形式に変換できること

例）３次元形状、特定の視点から見た２次元画像

座標、曲線

iii)他の対象物体との関係が記述できること

例）ＡはＢより上にある、ＣはＤに含まれる。

(2) 豊富な基本ルーチン

行列演算、雑音処理、陰線消去など、重要な基本ルーチンを保持していることが必要である。そのためには、強力な数値演算機能を有していなければならない。

〔参 考 文 献〕

[田中86] 田中： 定性推論からOntological なAIへ
AIジャーナル No.7(1986)

[中島85] 中島： 超時空プログラミングシステムURANUS
情報処理学会プログラミングシンポジウム(1985)

[八田86] 八田、他： 多重世界による時間論理システム
情報処理学会知識工学と人工知能研究会(1986)

[篠原86] 篠原、他： 時間関係と因果関係を扱う推論方式の開発
電力中央研究所研究報告585023(1986)

[青柳85] 青柳、他： Tokio かける言語 － Prologの自然な拡張
ロジックプログラミングコンファレンス'85(1985)

[米崎84] 米崎、他： 時間論理プログラミング言語Temploq
日本ソフトウェア科学会第１回大会pp.77-80(1984)

[桜川85] 桜川： Temporal Prolog
情報処理学会ソフトウェア基礎論研究会(1985)

[原87] 原、他：エキスパートシェルSOLON における時間概念の表現と推論方式
情報処理学会全国大会(1987)

[新田86] 新田、他： 工業所有権法の知識表現システムKRIP
情報処理学会論文誌 Vol.27 No.11(1986)

[Al184] Allen,J.F.: Towards a General Theory of Action and Time
Artificial Intelligence Vol.23,No.2,pp.123-154(1984)

[Fik71] Fikes,R. et. al:STRIPS:A New Approach to Application of Theorem
Artificial Intelligence Vol.2,pp.189-208(1971)

[McD82] McDermott,D.: A Temporal Logic for Reasoning About Processes and Plans
Cognitive Science 6. pp.101-155(1982)

[Kow] Kowalski,R. et.al:A Logic-based Calculus of Events
New generation Computing

[Sho85] Shoham,Y.:Ten Requirements for a Theory of Change
New Generation Computing,3 pp.467-477(1985)

[Mos84] Moskowski,B.:Executing Temporal Logic Programs
University of Cambridge Computer Laboratory. TR-55(1984)

[Wol81] Wolper,p.:Temporal Logic Can Be More Expressive
22nd Annual Symposium on Foundation of Computer Science(1981)

3. メタな知識を用いた推論制御

3.1 はじめに

問題解決に必要な知識には二つの種類があり、対象レベルの知識に加えてシステムが備えている知識そのものについての知識（すなわちメタ知識）を明示的に扱うことが、ひとりシステムの機能拡張という観点からだけで無く、その機能を明確にし構造を透明にする上で重要であるとの指摘はエキスパート・システムの研究のごく初期のころからなされていた。しかし知識についての厳密な定義が困難であるのと同様に、メタ知識あるいはそれを用いた推論制御であるメタ推論を明確に規定することは必ずしも容易ではない。これまでにいくつかのシステムで提案されてきたアプローチも、したがって比較的アドホックなものが多く、メタなレベルの知識や推論を統一的な枠組みで扱おうとした試みや理論的な側面に重点を置いた研究はまだまだ少ない。たとえば知識表現手法をとってみても、もともとメタ知識を表現するために提案された手法というのはほとんど無く、従来のプロダクションルールやフレームあるいは述語論理などの枠組みの中で、対象レベルの知識と一体となって実現されている場合が多い。つまり、対象レベルの知識とメタレベルの知識の間には相対的な意味的相違はあるものの、形式的には明確な区別が存在しない場合がほとんどである。

しかし、主としてルールベースのエキスパート・システムを構築しようととたこれまでの多くの試みの中で、専門家が持つ経験的な知識を単純に集積しただけではシステムが意図通りに動作せず、それらの知識の適用を制御する仕組みを導入しなければならないことや、そうした仕組みに対する要求がシステムの規模の増大につれて指数的に大きくなることなどが明確になってきた。さらにフレームなどの知識表現の場合には、フレーム間の継承や付加手続きなどが一体となって動作するため、システム全体が機能的にも構造的にも見通しにくくなり、メタなレベルの知識、推論を陽に扱うことの意義がより大きくなる。

本レポートでは、メタ知識やメタ推論に対する様々な考え方をまとめると共に、代表的ないくつかのシステムでインプリメントされたメタ知識・推論の例を取り上げてそれらの機能や問題点を検討し、今後の方向を探ることとする。

3.2 メタ知識，メタ推論の機能

はじめに、メタなレベルの知識・推論が持つべき機能をいくつかの観点から捉え、整理してみる。もちろん、こうした機能の中には既にある程度実現されているものもあれば、かなり困難な研究課題のレベルにとどまっているものもある。なお、前述した通りメタ知識を用いた推論をメタ推論と呼ぶ訳であるが、以降ではこの両者を必ずしも厳密には区別しないで扱うことがある。

(1) 適用可能な知識の限定

知識ベースに格納されている知識の中から、現在の局面で用いることが可能である知識あるいは現局面で意味を持つ知識を抽出し、探索範囲を限定することによってシステムの処理効率を向上させるものである。(2) の「知識の適用順の制御」と並んで、エキスパート・システムの初期の段階から研究されてきたテーマである。たとえばEMYCINでは次のようなルールを用いることができる。

もし　：

1)培養の採取部位が無菌的でない部位であり、

2)現在の菌種と同一と思われる過去の菌種について、

その前提部で言及しているルールがある

ならば：

それらのルールは確実に役に立たない（CF＝１）

EMYCINのルールインタープリタは通常のルールの適用に先だって、メタルールを処理する。このメタルールを実際に使用するか否かは、システムのオーバヘッドの増大と結局のところ無駄に終わる膨大な探索の割愛の可能性との間のトレードオフで決めることになるが、そうした処理効率向上の観点もさることながら、このメタルールによって対象レベルのルールに対する一定の意味づけができるというメリットも見逃せない。

（ただしvan Melle によれば、EMYCINを用いた初期のシステム［HEADMED.SACON.PUFF.CLOT］ではシステムの規模が小さく、この機能は用いられていない。

(2) 知識の適用順の制御

ルールベースのシステムを対象としてインプリメントされた例では機能的に二つの種類がある。

a.探索の効率化（後ろ向き推論）

(1) と同様にENYCINで以下の様なルールを書くことができる。

もし　：

1)感染症が骨盤腔ようで

2)腸内菌科を前提部に持つルールがあり

3)グラム陽性のかん菌を前提部に持つルールがある。

ならば：

前者は後者よりも先に適用されるべきであるとする

弱い根拠がある。（CF＝ 0.4）

EMYCINは後ろ向き推論のシステムで、ある属性の値を決めたるには使用可能なすべてのルールの適用を試み、それらの結果を確信因子の結合という形で統合する。したがって、個々のルールの適用順が最終的な属性の値に影響を与えることは無いが、処理効率には大きく関与する。すなわち仮説としての属性の値が確定する（確信因子の値＝１）とその時点で残りのルールの適用を中止するからであ

る。こうして、確信因子の値が１であるルールがそうで無いルールより先に適用されることになる。

こうした理由の他にも、上のメタルールによって推論の流れを制御することが可能になり、Focus attention という意味でシステムの動作をユーザにとってより自然なものにすることができる。

b.競合の解消（前向き推論）

一般的に、前向き推論ではルールの適用順序によって最終的に得られる解が変化する。したがって適用可能な複数のルールの競合解消の戦略はシステムにとって本質的な機能である。代表的な前向き推論用のシステムであるOPS-5 では前提部の条件節の個数や属性が参照された時間的な前後関係によって競合を解消する２種類の戦略をもっている。ただし、OPS-5 ではこうした機能をメタ知識あるいはメタ推論と呼んでいる訳では無い。

(3) 知識の性質に関する知識

知識の規定や適用順序制御の方法として(1)，(2) に示したものがすべてという訳では無い。上の例ではプロダクションルールを対象とし、それらを制御するために必要な情報もルールの中に存在していた。しかし、さらに多くの情報を用いたメタ推論もあり得、たとえば各知識（プロダクションルールに限定されない）の作成者や作成日付、これまでの使用頻度、成功／失敗の比率、平均実行時間などの情報をメタ知識と称する場合もある。当然この時にはこれらの情報を用いて各知識の適用を制御するメタ推論部があり、たとえば

・成功の比率がより大である知識を先に適用せよ

・作成日付がより新しい知識を先に適用せよ

などの規則が用いられる。

(4) 知識のモデル化

プロダクションルールの前提部に現れる属性と結論部で用いられる属性の関係に注目し、たとえばそれらを統計的に処理することによって、知識ベースの中の知識をモデル化することができる。

単純な例としては、ある属性の値を決定する時に各ルールの前提部で参照される属性の種類に一定のパターンを見出すことができた場合、その関係を知識の演えき的モデルと考えることができる。この様な知識のモデルには、知識ベースの説明や不正な知識のチェックあるいは知識ベースの意味的一貫性の保持など多くの用途があり得る。

以下に示すのは、R.Davis のTEIRESIAS の例である。TEIRESIAS はMYCIN 型の知識ベースを対象として知識獲得の支援や推論仮定の説明をするシステムであるが、既存のルールセットを見てそれらが見てそれらが共通に持つ性質を抽出しルールのモデルを形成する。たとえばMYCIN のルールには感染症の原因となる菌のカテゴリ

を結論づけるルールのサブセットのモデルとして、その様なルールのほとんどは前提部で検体採取部位および感染症の形について言及しているというものがある。また、他のルールモデルは前提部で検体採取部位と感染症の型に言及するルールは原因菌の感染経路についても言及する傾向のあることを示している。こうしたルールモデルを用いることによって、次の様なやりとりが展開される。

（専門家が新たなルールを提示する）
もし　：
　　1)＊＊患者の疾患が原発性菌血症であり
　　2)＊＊採取部位が非汚染部位である。
ならば：
　　＊＊カテゴリが腸内細菌属である根拠がある（0．8）
（ここで、この新たなルールとTEIRESIAS のルールモデルが比較される）
菌の属するカテゴリを決定する規則で
　検体採取部位
　感染症
に言及する規則の多くのものでは、さらに
　［A］菌の感染経路
についても言及している。
ここで私が［A］を考慮した節を追加してみても良いか
　＊＊良い
次の条件節でよいか
　［A］感染経路が消化管である。
　＊＊良い
［＊＊以下がユーザの入力］

TEIRESIAS の場合にはメタ知識としてのルールモデルが最初から存在している訳ではないので、知識ベース構築の初期の段階から上のような機能が動作することはないが、ある程度の完成度を備えた知識ベースがあれば知識獲得のプロセスを有効に支援することができる。

(5) システムの能力のモデル化

多くのプログラムで、その処理の中心になっているアルゴリズムの部分に比較して、各種データのチェックのためのコード部分が大きな割合を占めているのは良く知られている通りである。こうしたチェックによって、ユーザが入力したデータの誤りの大部分を検出することができるが、もともとシステムが正常に動作することが保証されている範囲外で、そのプログラムが使用されることを防ぐのは容易ではない。特に大規模プラントの診断やリアルタイム制御、医療診断などの分野のエキスパート・システムの場合はこの問題は非常にクリティカルである。

システムが自分自身の能力や機能についての知識をもち、与えられた問題の意味あるいはそれを解くための知識の充足性について判断することができれば上の機能を実現することができる。

(6) 問題解決の戦略に関する推論

合成型、設計型の問題に対する標準的なアプローチ副問題への分割がある。与えられた問題を複数のより単純な副問題に分割し、その各々をさらに小さな副問題に分け、という操作を何回か繰り返すことによって問題を単純化し、最終的にそれらの副問題の解を合成することによって始めに与えられた問題の解を生成するという手法である。

最下位の副問題を解くための知識を基準とすれば、そうした副問題を生成するための知識はメタレベルの知識と考えることができ、したがってこの場合には何段階かのメタなレベルが存在することになる。一般的に問題をより小さな副問題群に分割した場合、個々の副問題は最初の問題に比べて複雑さが減少するが、一方で副問題間の相互干渉という厄介な問題を抱え込むことになる。メタレベルの推論はこの相互干渉ができるだけ発生しないような方法で問題分割を行わなければならない。

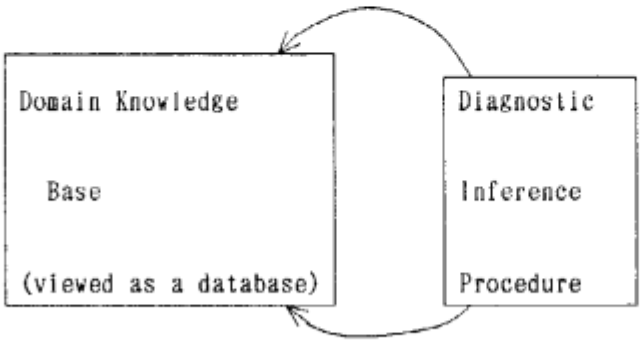
3.3 メタな知識、メタ推論の例

本章では、メタなレベルの知識、推論を実装しているいくつかのシステムを例にとりそれらの機能、問題点を検討することとする。

(1) EMYCIN, TEIRESIAS, NEOMYCIN

初期の代表的なシステムであるこれらのうち、EMYCIN, TEIRESIAS に関しては既に3.2で概略を紹介したが、EMYCINのメタルールの機能は十分にその目的を達したとは言えない。事実HEADMED, SACON, PUFF, CLOTなどのEMYCINを用いた主なシステムではメタルールは使われることはなかった。それらのシステムの規模が比較的小さかったためもあるが、たとえば知識の適用順の制御について見れば、ルールの登録順や前提部の条件節の並べ換えあるいはダミー的な条件節の追加などによって、ある程度のコントロールが可能な訳で、そうしたやり方のほうがむしろ細かい制御が実現し易い面

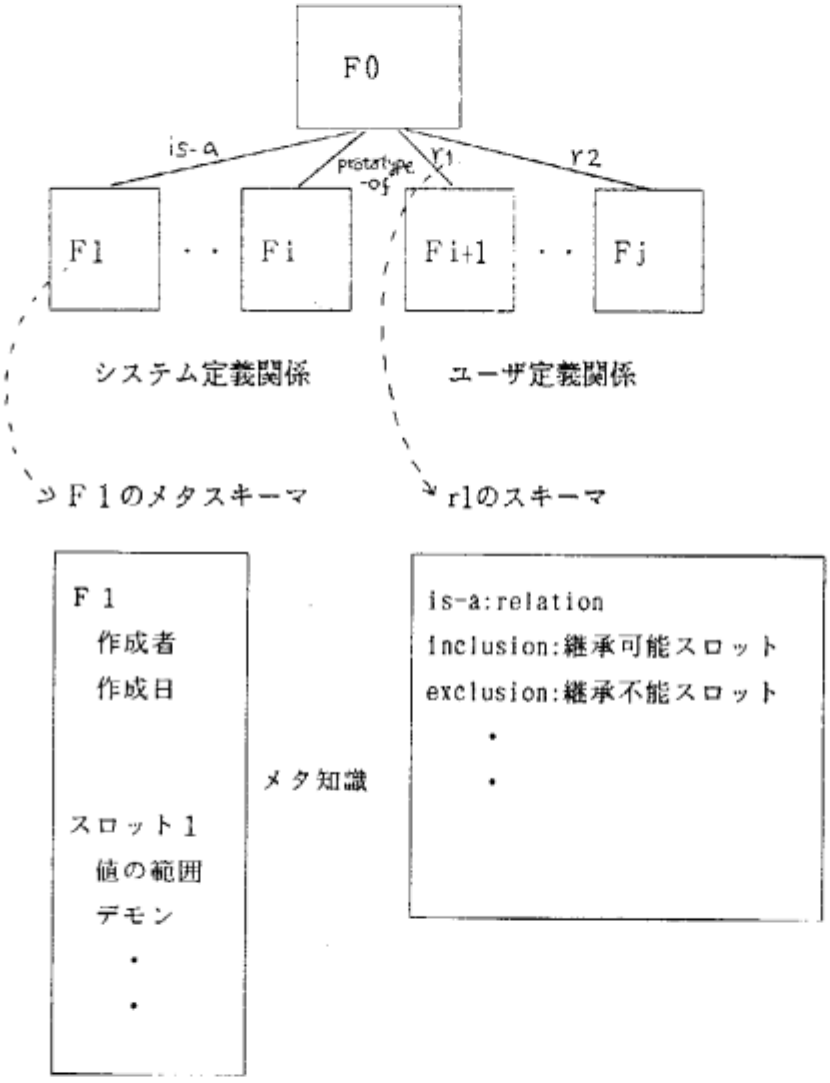
もある。しかし、この結果、本来対象分野の問題解決に必要なヒューリスティクスを表現したものであるべき知識ベースに、それらの知識の適用を操作するための知識が入り込むことになってしまい、知識ベース全体の見通しを悪くするという別の問題が生まれた。これを解決するための枠組みを提供しようとしたのがNEOMYCINである。NEOMYCINはW.J.Clancey が開発したシステムで、MYCIN の知識ベースを拡張するとともに、MYCIN の中では一体になっていた感染症診断のための知識と推論制御のための知識を分離した点に特徴がある。下図はNEOMYCINのアーキテクチャを示したものである。



Diagnostic Inference Procedureと呼ばれる部分がメタルールを集積したもので、ここで用いられているメタルールは感染症に関する知識は含まず、推論の制御のための知識のみが記述されている。

(2) KNOWLEDGE CRAFT(KC)

ＫＣはフレーム（スキーマと呼んでいる）表現を基本とし、デモンとしてプロダクションルールや述語論理型の手続きを使用することができるエキスパート・システム構築用ツールである。ＫＣで言うメタ知識は、 3.2の分類にしたがえば(3) に属するものである。下図はＫＣのスキーマの構造を示したものである。



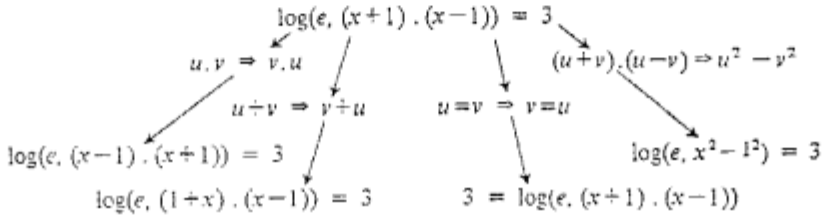
ただし、上のメタ知識を使用するための手続きであるメタ推論部についてはＫＣの中で特に標準的な枠組みが用意されている訳ではない。

(3) PRESS

PRESS(PRolog Equation Solving System) はA.Bundy らが開発した代数方程式解法のためのシステムである。ここではメタレベル推論を、ＡＩにおける組合せ的爆発という問題に対する解決の試みと考えており、求める証明の中に含まれているより可能性の高い諸規則だけが式に適用されるように、探索に指針を与えるものとしている。与えられた方程式を解くためには、書き換え規則と呼ばれる代数的恒等式をいくつか適用することによって元の方程式を単純化することが必要である。しかし、適用可能な書き換え規則は一般には複数個あり、一種の競合解消の問題が発生する。たとえば

$$\log((x+1)(x-1))=3$$

に対して、次図の書き換えのいくつかは全く無意味である。



［バンディ８５］より

上の式を含む方程式の標準的な解法はたとえば下の様である。

(1) $\log(x+1)+\log(x-1)=3$

i

(2) $\log((x+1)(X-1))=3$

ii

(3) $\log(x^{2}-1)=3$

iii

(4) $x^{2}-1=e^{3}$

iv

(5) $x^{2}=e^{3}+1$

v

(6) $x=\pm\sqrt{e^{3}+1}$

各式の右側に付した数字で代数的操作を示すとそれらの意味は次の通りである。

i ・・「ひきよせ」。未知数x をそばにひきよせて、次の「とりまとめ」の適用を可能にする。

ii・・「とりまとめ」。未知数x の出現回数を減らし、次の「きりはなし」の適用を可能にする。

iii , iv, v・・「きりはなし」。未知数以外の項を右辺に移項し、未知数を左辺に取り残して解を求める。

PRESS はこうした手続きをPROLOGで記述しており、下はその例である。

```
singleocc(X, L=R) & position(X, L, P) & isolate(P, L=R, Ans)
                    → solve(L=R, X, Ans)                    (ii)
isolax(I, FUi=V, RHS, Condition) & isolate(Rest, RHS, Ans)
                    → isolate([I|Rest], FUi=V, Ans)         (iii)
isolax(2, log(U1, U2)=V, U2=U1^V, true)                     (iv)
```

(ii), (iii), (iv) の宣言的意味は以下の通りである.

— (ii) もし L=R がLのなかのPにおいて正確に1度だけXの出現を含み, かつ, L=R 内のXをきりはなした結果が Ans であるならば, その Ans はXに関する L=R の解である.

— (iii) もし位置Iの引数をきりはなすために FUi=V に書換え規則を適用した結果が RHS であり, かつ位置 Rest に関して RHS をきりはなした結果が Ans であるならば, FUi=V を位置 [I|Rest] に関してきりはなした結果は Ans である. (ここで [I|Rest] はIを先頭としたときの残りのリストである.)

— (iv) log の第2引数をきりはなすために log(U1, U2)=V にきりはなし書換え規則を適用した結果は U2=U1^V である. (isolax の条件スロット内にある 'true' は何も条件がないことを指示している. 'U1^V' という式は「U1 のV乗」のことである.)

[バンディ85] より

PRESS は多項式や三角関数、指数関数、対数関数を含む諸式に対して代数的操作を行うことができ、上に示した「ひきよせ」、「とりまとめ」、「きりはなし」の他にも以下のような方法を扱うことができる。

「未知数の変換」
「均質化」
「関数の取り換え」
「多項式に関する諸方法」
「三角関数に関する諸方法」
「単純化の諸方法」
「消去」

PRESS のメタレベル公理はPROLOGの節として表現されているため、その推論制御はトップダウン、深さ優先の探索、左から右の順でなされる副目標の扱いなどPROLOG的な制御であるが、MECHO プログラムにおいてさらに拡張されている。

(4) MOLGEN

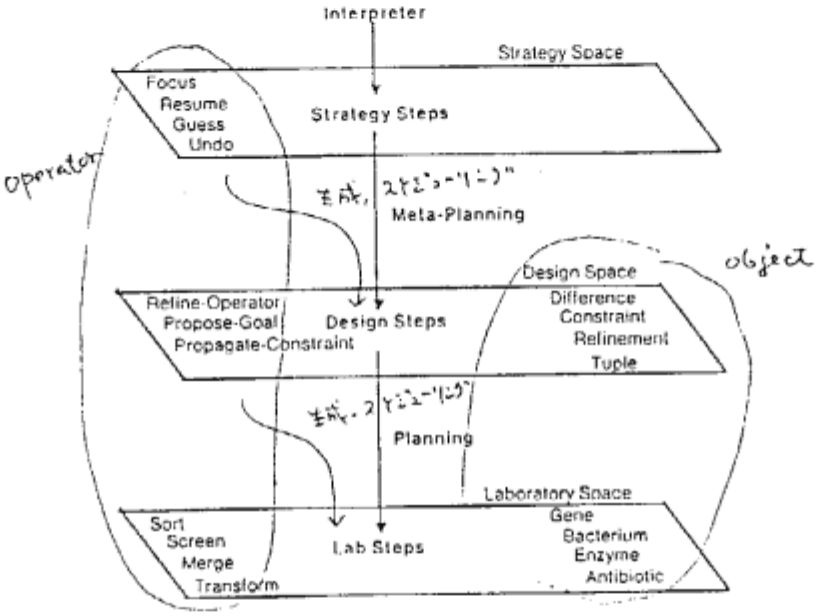
MOLGENはM.Stefikによって研究された分子遺伝学の実験計画を作成するためのシステムで 3.2の分類によれば(6) に属するメタ知識を扱っている。

問題解決システムとしてのMOLGENの特徴は大きく以下の二つである。

・制約指向のパラダイムによる副問題の解決

・メタプランニングによる副問題の生成、制御

MOLGENはネズミ遺伝子を用いてインシュリンを生成する細菌を設計する。問題を扱っているが、次の図のようにこの問題を三つの段階に分割して考える。



[Stefik 81] より

各階層にはオペレータとオブジェクトがあり、それらによって一段下位の階層のタスクの生成、スケジューリングを行う。それぞれの階層の意味は次の通りである。

a. Lab Space

実験室での操作と対象物によって実験を記述する。オブジェクトは抗生物質、DNA 構造、遺伝子などで、Merge,Amplify,React,Sortなどの遺伝子に対する抽象的操作がオペレータになる。

b. Design Space

制約伝播の手法によって実験室でのプランを生成し、詳細化する。

c. Strategy Space

least commitment (Under-constrained なとき、決定を先へ延ばす)、あるいはheuristic approach (情報が不完全の時、仮の決定をする。) によってDesign taskの生成、スケジューリングを行う。

MOLGENでは問題空間を三つのレベルに分け、それぞれの空間が一段下の空間に対するメタレベルの推論を行っている。推論制御の手法はconstraint propagation, least commitment, あるいはheuristicsなどによる。

ネズミ・インシュリンの問題に対してMOLGENは一定の成果を上げ、いくつかのシステムが同様のアプローチを用いてシステムを構築しているが、開発者によればMOLGENの問題点は次のようである。

i. Guessing and Solution Density

MOLGENは解の空間が疎であるような問題を対象としている。そうで無い場合は、こうしたアプローチが有用であるとは限らない。すなわち「考えること」と「考えることについて考えること」のあいだのcost/benefitを考慮しなければならないからである。

ii. Incorporating new information

実際の実験の過程をMOLGENによる計画段階にフィードバックさせることができない。実行前のプランニングはgoal driven であるが、実行中のプランニングはevent drivenである。

iii. Reasoning about theories

仮説テストするための実験を生成する能力がない。

iv. Reasoning about scenarios

自分自身の拘束や行動により引き起こされる未来について推論することができない。これにはhierarchical reasoningとtruth maintenance の結合が必要である。

v. Reasoning about failures

失敗の原因についての知識や知識ベースの完全性に関する知識によって失敗を減少させることのできる仕組みが必要である。

3.4 おわりに

メタ知識、メタ推論に関する研究には推論の制御、最適化から知識ベースの説明、知識獲得の支援、システムの能力のモデル化、問題解決の戦略立案など多くの側面があり、手法についても標準的なものは提案されていないと言える。今後もしばらくは個別の問題ごとに最適なアプローチ、手法が研究されていくであろうが、それらはむしろ研究の幅を拡げる方向に進むと考えられる。こうした問題向きの言わばアドホックな研究とは別に、定式化された世界を対象にした理論的なアプローチを考えると、知識表現の手法として何らかの論理に基づいた枠組みが必要になろう。PRESS の例やICOTのdemo述語の研究などはそうした流れのなかに位置づけることができる。

いずれにせよ、メタなレベルの推論制御はエキスパート・システムの実用化、大規模化が進むにつれて、ますます必須の機能となっていくことは間違いないであろう。

〔参 考 文 献〕

[vanMelle80] W.J.van Melle;System Aids in Constructing Consultation Programs, UMI Research Press,1980

[Barr82] A.Barr,E.A.Feigenbaum:The Handbook of Artificial Intelligence Ⅱ, William Kaufmann Inc.,1982

[バンディ85] A. バンディ；メタレベル推論と意識、『ＡＩと哲学』産業図書、1985

[Clancey86] W.J.Clancey;From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons:ONR Final Report 1975-1985,THE AI MAGAGINE August,1986

[Stefik81] M.Stefic;Planning with Constraints(MOLGEN:Part 1,2),ARTIFICIAL INTELLIGENCE 16 1981

[Hays-Roth86] B.Hays-Roth et.al ;PROTEAN:DERIVING PROTEIN STRUCTURE FROM CONSTRAINTS,Proc.of AAAI'86

4. 深い知識の利用

4.1 次世代シェルにおける深い知識

4.1.1 問題点

現状におけるエキスパートシステムが利用する知識の多くは、専門家が持つタスクに直結する知識（浅い知識）のみであったので、以下に示すような診断能力と説明機能及び知識獲得支援に限界があった。

(1) 浅い知識は、通常それが使われるべき状況を明確に記述した条件部とそれに対応する実行部とから構成される。そのため、条件部に一致する状況に対してはうまく動作し、専門家と同程度の振舞いをするが、条件部とすこしでも異なる状況に対してはうまく対処することができない。換言すれば、従来のエキスパートシステムはいくつかの予め想定されている問題を解く能力があるだけで、基本を理解して様々な問題が解ける能力を持つに至っていない。

(2) 従来の説明機能は、問題解決に利用されたルールを順次提示させるだけであり、そのルールがその場面で利用されることがなぜ正しいのかということ、即ちルールの正当性を提示させる能力はなく、専門家のみに理解可能で初心者には理解しにくいものである。

(3) 従来のエキスパートシステムの知識獲得支援は、メタレベルの知識により浅い知識の構造を調べ構造的に不明瞭な箇所を専門家に提示するに留まっており、専門家の誤った思い込みを正すといった局面に使えるほど高級でない。

以上の限界は、エキスパートシステムにとって本質的な問題であり、次世代のシェルの開発において考慮されなければならない課題であると考えられる。 浅い知識としての経験的知識を持ち、日常の問題解決に利用している専門家に注目すると、常に浅い知識だけを用いているのではないことに気が付く。人間の専門家は浅い知識だけではなく、関連するドメインにおける原理的な知識も持っており未経験の問題に直面して経験的知識では対処できないような場合には、原理的な知識を用いて「熟考」することによりその問題の解決にあたるのである。このような原理的な知識は、直接問題の解決に利用できるようには整理されてはいないが、様々な問題に利用できる基本的な性質を持つことから、「深い知識」と呼ばれている。本節では次世代シェルの開発に重要な位置を占めると思われる深い知識に関する検討結果について述べる。

4.1.2 深い知識

深い知識［Hart82］［Michie82］［Chanclraskaram84］［Yamada84］［上野86］はまだ統一された定義はないが、ここではタスクに直結した知識を浅い知識と捉え、浅い知識を生成することに関連した知識を深い知識と捉える。［浅い・深い」という概念は、この定義からタスクによって決定されるものであり相対的に変わるもので絶対

的に決定できるものではないものと考える。

本節ではタスクを人工の構造物（特に車）の故障診断に絞り、専門家が診断ルール（浅い知識）を生成するプロセスを考察することにより深い知識の整理を行った。その結果深い知識としては、①対象（としているメカの）モデル、②物理原理、③物理状態を徴候や故障仮説に解釈する知識が必要であると判明した。

①は対象としているメカの従来の図面情報（ＣＡＤデータ：部品の寸法、形状、配置など）に近いが、それだけでは診断ルールの生成には不十分である。診断ルールは対象モデルの様々な状態に物理式を順次適用することによって生成されるため、対象モデルには適切な物理式を選択するための情報が必要である。この情報には、設計者がどの様な目的でその部品を組み込んだか、換言すれば『設計者が意識して部品に持たせた機能』（設計者の意図:intention）、およびその部品の使用環境が含まれる。設計者の意図としては、部品の役割がある。部品の役割とは、装置を構成する各部品を独立に考えた場合の一般的な機能
(function)を指すのではなく、設計者がその対象において各部品を意図的に持たせた機能を指す。また、部品の重要性も設計者の意図に含まれる。一方、部品の使用環境とは、設計者が意識していない部品の副作用的なふるまい、およびその部品が受ける様々な作用に関する情報（動作環境）を指す。また、部品の属性、観測容易性、および耐久性も部品の使用環境に含まれる。

①の深い知識は、知識コンパイラ（深い推論）の利用法の違いから、Device World とControl World に分かれる。また②をPhysical Worldと呼び、③をInterpretation World と呼ぶ。従って、メカの故障診断における深い知識は、以下の４種類のワールドに分化される。

1. Device World（ＤＷ）
2. Control World （ＣＷ）
3. Physical World（ＰＷ）
4. Interpretation World（ＩＷ）

4.1.2.1 Device World

ＤＷは、構造などの知識を利用して深い推論を進めたり、物理式の適用基準を与えるためのワールドであり、属性(attribute) 、動作環境(environments)、部品の役割(role)および物品の構造(structure) を含んだ診断対象のモデルである。図4.1-1にＤＷの例を示す。

4.1.2.2 Control World

ＣＷは、深い推論の結果生成されたルールを診断に適用するときの順序を決めるためのワールドであり、各部品の観測容易性(observability)、耐久性(durability)、重要性(importance)の３つからなる。

観測容易性(observability)は、診断の段階で観測の容易な徴候からチェックしたいので、条件部に書かれた徴候のチェックが容易なルールの優先順序を高くし、チェックが困難なルールの優先順序を低くするために用いる。例えば、自動車のエンジンの場合、観測が容易な冷却水の量をチェックするルールは優先順序が高くなり、観測が困難なウオータポンプの動作をチェックするルールは優先順序が低くなる。

耐久性(durability)は、耐久性の低い部品の方が壊れやすいので、結論部で耐久性の低い部品をチェックするルールの優先順序を高くし、耐久性のある部品をチェックするルールの優先順序を低くするのに用いる。例えば、耐久性の低い定期交換部品であるファンベルトの状態をチェックするルールの優先順序は高くなり、耐久性の高いウォータポンプの状態をチェックするルールの優先順序は低くなる。

重要性（importance）は、装置内で重要な部品は、その故障が装置全体に対して非常に大きな損害を与える可能性があるので、重要性の高い部品をチェックするルールは適用の優先順序を高くし、重要性の低い部品をチェックするルールの優先順序を低くするのに用いる。

以上の３つの評価基準の他に、故障原因として装置のどのレベルまで調べればよいかを示す部品の階層性（hierarchy)が考えられるが、これはＤＷの構造情報から得られる。

4.1.2.3 Physical World

ＰＷは、深い推論を進めるときに重要な役割を果たすワールドである。物理式は、適用条件と共に記述され、値が元来与えられるべき変数（原因を表す変数）が右辺、その式を計算することによって値が求まるべき変数（結果を表す変数）が左辺という形式で書かれている。また、変数の値は定量値ではなく、＋（基準値以上）、０（基準値）、－（基準値以下）という３つの定性値をとる。図4.1-2 にＰＷの例を示す。

4.1.2.4 Interpretation World

ＩＷは、ルールを生成するプロセスで停止条件となるワールドであり、メカのある状態（推論ノード）を故障の徴候および故障仮説に対応させて解釈するための知識からなる。実際には、ＤＷの知識をアクセスして初めて解釈が成立する形式で書かれている知識と直接メカの状態を解釈する形式で書かれている知識が存在する。

図4.1-3 にＩＷの例を示す。

4.1.3 浅い知識の深い知識からの生成［小高　86］

知識コンパイラ（ＫＣ：knowledge Compiler）は、深い知識より診断ルールを自動生成するプロセスである深い推論を実現する機構であり、図4.1-4 にその概要を示した。すなわち所期徴候が与えられると、ＫＣはＤＷとＰＷを利用して深い前向き推論を行い最終的にＩＷを利用して故障仮説を生成する。また、ＫＣは生成された故障仮説から、やはりＤＷとＰＷを利用して深い後ろ向き推論を行いＩＷを利用して関連する徴候をすべて生成する。以上の３種類のワールドにより、初期徴候を条件部に含むすべてのルールが生成されるが、これらのルールの適用順序をＣＷにより決定する。以下、深い前向き推論と深い後ろ向き推論およびルールの適用順序決定プロセスについて詳しく述べる。

4.1.3.1 深い前向き推論

深い前向き推論では、まず絶えられた初期徴候から計器のroleにより初期ノードを生成し、制約伝播を行う。例えば徴候が“ＴＥＭＰメータが８０℃以上”であるとき、ＤＷの“ＴＥＭＰメータ”のroleより“冷却水温度が８０℃以上”という初期ノードを生成する。ただし、メカが異常な状態の時に、計器が測定するレンジを初期徴候とする。

次に制約伝播プロセスについて述べる。制約伝播プロセスは、ＤＷとＰＷの知識の適用（マッチングと物理式の定性的な解釈）の繰返しである。ＤＷとＰＷの適用優先順序は基本的には差はない（本ＫＣではＤＷが先に適用されている）が、ＤＷは構造等の情報を利用して制約伝播が可能になる場合に適用され、ＰＷは物理原理を考えて制約伝播が可能になる場合に適用される。

以下、ＰＷの適用法について述べる。マッチングは、深い前向き推論が結果から原因を探る過程であるので、推論ノードが言及している部品の物理パラメータと物理式の左辺の結果を表す物理パラメータ（変数）とがマッチする物理式が選択されることによって為される。ただしコンフリクトが生じた場合は、その状態に関与する部品の持つ物理パラメータと適用条件の一致度が最も高い物理式が選択される。一方、物理式の定性的な解釈は、左辺の変数から右辺の変数に値が伝播されることによって為される。値は、右辺の変数がｎ個あるとすれば、（ｎ－１）個の変数は基準値（即ち０）と仮定することにより残りの１つの変数を求め、次々と値を求める変数を与えることによって右辺の全ての変数に伝播する。この結果新しい推論ノードが生成される。例えば、“冷却温度が８０℃以上”という推論ノードに大しては温度（Ｔ）を左辺に持つ物理式Ｔ２＝Ｑ／ｃｆ＋Ｔ１が選択され、値を伝播するとＱ（＋）、ｃ（－）、ｆ（－）、Ｔ１（＋）となり、“冷却水の吸熱量が基準値

以上”、”冷却水の比熱が基準値以下”、”冷却水の流量が基準値以下”、”冷却水のＹの温度が基準値以上”という新しい推論ノードが生成される。

このようにして得られた推論ノードは、故障仮説生成用のＩＷを適用することにより、適用可能ならばその推論ノードに対して故障仮説を立て推論を停止し、適用不可能ならば上記の制約伝播プロセスによりさらに新しい推論ノードを生成する。

以上のプロセスを繰り返すことにより、すべての故障仮説が立てられ、与えられた徴候に関連するすべての不完全なルール群が生成される。

図4.1-5 に”ＴＥＭＰメータが８０℃以上”という徴候が与えられたときの深い前向き推論のプロセスを示し、図4.1-6 にその結果生成された一つの不完全なルールを示す。

4.1.3.2　深い後ろ向き推論

深い後ろ向き推論では、まず与えられた故障仮説が言及している部品のroleおよびenvironmentsの否定をとり初期ノードとする。例えば、与えられた故障仮説が”オイル不足”である場合、オイルのroleおよびenvironmentsから、”シリンダ、ピストン間の摩擦係数が増加”および”オイルが圧送されない”という初期ノードを生成する。

深い後ろ向き推論の制約伝播プロセスは、深い前向き推論の場合と逆の過程となる。以下ＰＷの適用法の差異について述べる。マッチングは、深い後ろ向き推論が時間的には原因から結果を探る過程であるので、推論ノードが言及している部品の物理パラメータと物理式の右辺の原因を表す変数がマッチする物理式が選択されることによって為される。コンフリクトの解消法は同様である。また物理式の定性的な解釈は、右辺の変数から左辺の変数に値が伝播されることによって為され、他の右辺の変数には値が伝播されない。例えば推論ノード”オイルが圧送されない”にマッチする物理式は、Ｆ＝ρＱＶであるが、Ｑ（－）から他の原因を表すρとＶへ値は伝播されず、結果を表す変数Ｆのみに伝播しＦの値が（－）に決まる。

このようにして得られた推論ノードは、徴候生成用のＩＷ適用することにより適用可能ならばその推論ノードを徴候とし、不可能ならば上記のプロセスによりさらに新しい推論ノードを生成する。

以上のプロセスを繰り返すことにより、故障仮説に関連するすべての徴候を生成し、完全なルールとなる。図4.1-7 に”オイル不足”という故障仮説から関連するすべての徴候を生成するプロセスを示し、図4.1-8 に完成されたルールを示す。

4.1.3.3　適用順序

深い（前向き・後ろ向き）推論によって生成された診断ルール群は、ＣＷの観測容易性によりルールの条件部を評価し、ＣＷの耐久性、重要性および階層性により

ルールの結論部を評価することによりルールの適用順序が決定される。例えば、以下のＡ－Ｃのルールが生成され、その適用順序について考える。

Ａ：　ＩＦ　TEMPメータが８０℃以上
　　ファンがくるくる回る
　　ＴＨＥＮ　ファンカップリングの故障

Ｂ：　ＩＦ　TEMPメータが８０℃以上
　　異常音がする
　　ＴＨＥＮ　ウォータポンプの故障

Ｃ：　ＩＦ　TEMPメータが８０℃以上
　　オイルプレッシャーランプの点灯
　　エンジンの異常振動
　　ＴＨＥＮ　オイル不足

観測容易性の観点からは、Ｂ→Ｃ→Ａの順であり、耐久性の観点からはＣ→Ｂ→Ａであり、重要性の観点からはＢ→Ｃ→Ａであり、階層性の観点からはＣ→Ａ→Ｂであるので最終的な適用順序はＣ→Ｂ→Ａとなる。

4.1.4　深い知識に基づく説明機能

ルールの正当性を中心に説明機能を拡張する試みは、文献[Swartout 83][Neches 85]を通してすでに見られるが、ここではＫＣと深い説明機能との関連について述べる。診断ルールを用いた説明に関しては従来のWHY とHOW の他に、ルールの適用順序を示すWHENを考える。これは、診断ルールが適用される適用順序をそのまま提示するものである。

次に深い知識を用いた説明を考える。ルールの正当性に関する説明は、仮説から徴候を導くプロセスを示すことにより可能であり、WHY(X)（X ：ルール番号）で処理される。WHENの正当性、つまりルールの適用順序の正当性に関しては、ＫＣがＣＷを用いてルールに順序付を行うプロセスを示すことにより可能であり、WHEN(X)で処理される。さらに、メカに関する構造は機能に関する説明WHATを考える。これは、深い知識のうち、ＤＷをそのまま提示することによって処理される。最後に、ある仮定において発生するイベントに関する説明WHAT_IFを考える。これは、ある仮定において起こりうることをシステムにシミュレータさせ、そのときに発生するイベントを説明するものとなる。

以上のように、深い知識を充実させることによって、説明機能は従来より大きく

向上することが期待される。

4.1.5 深い知識に基づく知識獲得

現在の知識獲得支援機構[Davis 82][Politekis 84][Kahn 85] は、メタレベルの知識を利用し、入力知識をチェックしたり、知識ベースの不明瞭な構造を見つけることにより、知識ベースを洗練化することに重点が置かれている。しかしながら、このアプローチでは、診断ルールの正当性等、意味的な考慮を全く行わずに知識獲得支援を実現しているため、ある程度洗練化された知識ベースの性能向上、あるいは専門家の誤った思い込みという局面の適用には限界があり、受身的な知識獲得であると言える。そこでＫＣの利用が考えられる。完全なＫＣが構築できれば、知識獲得は必要ないはずであるが、現時点ではその様なＫＣの構築はなかなか困難であり、ＫＣを専門家の知的パートナと見なす方が現実的であろう。すなわち、専門家の与える診断ルールとＫＣの生成した診断ルールが異なり、専門家の方に誤りがあればその生成プロセスを提示することによりその誤りを改めさすことが可能である。すなわち、従来の知識獲得支援機能の限界を打破でき、知識獲得支援をインタビュー［川口 86 ］に関連づければ、深い知識がインタビューのドライバーとして利用できることが分かる。また、専門家の方が正しければ、深い知識が不足していることが考えられ、原理の再発見につながる可能性がある。

4.1.6 検討課題

これまで、深い知識の利用に関して、浅い知識の生成、説明機能、知識獲得の３つの課題について述べた。ここでは、今後の課題について述べる。

4.1.6.1 予期していなかった故障の診断

深い知識を用いることの効用の中に、予期していなかった故障の診断が可能になる可能性があることが挙げられる。３節において述べた浅い知識の生成法は、事前に用意されている深い知識の範囲で、全ての症状と故障原因とを関連づけるルールを生成することができる。一般に、あらゆる場合を想定して完全な診断ルールを予め用意しておく事は困難であり、この方式によって、予め想定し難い故障の診断が可能となるものと思われる。この考えを更に進めてImplicit structureやMal-structure が原因する故障の診断ルールの生成への拡張を検討することも重要である。［田岡 87 ］

4.1.6.2 深い知識の獲得

既に述べた４種類の深い知識の獲得に関する問題を検討しなければならない。ＰＷの知識は対象とするドメインに関する教科書から抽出することができる。Ｃ

WとＩWの知識は単純であり、個々の部品に関する独立した知識であるためその抽出は問題ない。ＤWが量の点でも最も多く、かつその獲得は重要である。我々が対象としているものは、人工物であることからその設計情報にＤWの知識は全て集約されているはずである。このことに注目して、ＤWの知識は対象物の設計情報から得る事を考える。設計情報は一般に膨大なものであり、その記述は困難であるが、近年進展が著しいＣＡＤシステムが生成するドキュメントからの獲得法を検討することにより、深い知識の獲得は現実のものとなると考えられる。

4.1.6.3 Generic Tasks [Chandrasekaran 86]

エキスパートシステムの構築を支援するツールは大きく進歩し、プロダクションシステムに基づく単一のアーキテクチャを持つ第一世代のツールから、マルチパラダイムといわれる、プロダクションシステムを初め、フレームやオブジェクト指向の概念などを統合した第二世代のツールが開発されている。しかし、中心はあくまでもプロダクションルールであり、前向き推論か後向き推論、或はその両方が使えるにすぎない。確かにプロダクションルールは様々なタスクに共通する基本的な知識表現手段として有用ではあるが、共通であるが故に各タスクに本質的な性質やサブタスク等を表現する能力に欠け、エキスパートシステムを構築するにはあまりにも記述のレベルが低過ぎるという問題がある。プロダクションルールより抽象度が高く、タスクが持つ固有の性質を反映したビルディングブロック（generic tasks ）を整備する必要がある。

Generic Tasks の考えは、タスクに潜在する本質的なサブタスクの発見、すなわちより高い抽象レベルにおけるタスクの表現を可能にする。診断型のタスクでは、症状と原因との因果関係を表すルール以外にも、あるデータの信頼性を評価するためのものや、原因を階層的に分類するためのものや、複数の仮説を統合するためのもの等のルール（知識）があることが既に明らかにされている。これらの概念を用いることによって、従来、ルールだけで表現されていた基本的なタスクを容易に表現する道具を提供することができる。これらの道具はエキスパートシステムのビルディングブロックとして機能し、大規模システムの構築にとって極めて強力な手段を提供するものと考えられ、今後更に検討しなければならない。

4.1.6.4 統合化された枠組み

従来、診断型のエキスパートシステムと設計型のエキスパートシステムは互いに独立に研究開発がなされてきたが、両者には取扱の対象となるモデルを持つという点で共通点を持つ。診断においては、対象の構造と部品の機能に関する知識が利用できれば、上述のように経験則だけでは対処できないような問題も解決す

ることができるようになる。一方、対象モデルは設計型のエキスパートシステムによって構築される。このように、対象モデルは両者に共通する知識となっている。

又、モデル（診断の対象物の構造と機能に関する知識）に基づく深い推論は診断においては、表層レベルの知識では処理できない問題に遭遇したときに、原理に戻って考える際に用いられ，設計においては、設計の中間段階における概略的な情報に基づく評価のための定性的な推論を行う際に用いられる推論技術である。いずれの場合も、対象とする物のモデルに関する情報に基づく推論であり、共通する高次推論技術として位置付けられる。

このように、診断と設計の二つの型に共通する知識と推論技術を検討して、共通点を核とする統合化された新しいフレームワークを開発することは意義深いものと思われる。

〔参考文献〕

[Hart 82] P.E. Hart: "Direction for AI in the eighties" SIGART, 79, p.79 (1982)

[Michie 82] D. Michie: "High-road and low-road programs" AI Magazine, 3(1), pp.21-22 (1982)

[Chandrasekaran 84] B. Chandrasekaran and S. Mittal: "Deep versus compiled knowledge approaches to diagnostic problem-solving" Developments in expert systems edited by M.J. COOMBS, pp.23-34, Academic Press (1984)

[Yamada 84] N. Yamasda and H. Motoda: "A Plant Diagnosis Method Based on the Knowledge of System Description" Journal of Information Processing 7, 3, pp.143-148 (1984)

[上野 86] 上野："対象モデルの概念に基づく知識表現について－深窓知識システムへのアプローチ－" 電子通信学会、人工知能と知識処理研究科意思料、186-4, (1986-4)

[小高 86] 小高、野村、田岡、山口、溝口、角所："知識コンパイラの構成とその応用"、情報処理学会、知識工学と人工知能研究会、48-2 (1986)

[Swartout 83] W.R. Swartout: "XPLAIN: A System for Creating and Explaining Expert Consulting Programs" Artificial Intelligence, 21, 3, pp.285-325 (1983)

[Neches 85] R. Neches, W.R. Swartout, and J. Moore: "Explainable (and Maintenable) EXPERT System" Proc. of IJCAI'85, 1, pp.382-389 (1985-8)

[Davis 82] R. Davis and D.B. Lenat: "Knowledge-Based Systems in Artificial Intelligence", MacGrow-Hill (1982)

[Politekis 84] P. Politekis and S. Weiss: "Using Empirical Analysis to Refine Expert System Knowledge Bases" Artificial Intelligence, 22, pp.23-48 (1984)

[Kahn 85] G. Kahn, S. Nowlan and J. McDermott: "MORE: An Intelligent Knowledge Acquisition Tool" Proc. of IJCAI'85, 1, pp.581-584 (1985-8)

[川口 86] 川口、溝口、山口、角所："データベースの論理設計を支援する知的インタビューシステム"、情報処理学会、知識工学と人工知能研究会、48-1 (1986-9)

[田岡 87] 田岡、山口、溝口、角所："深い知識に基づくドメイン特化型シェルの構築"、情報処理学会、知識工学と人工知能研究会、51-9 (1987)

[Chandrasekaran 86] B. Chandrasekaran: "Generic tasks in knowledge-based reasoning: Highlevel building blocks for expert system design", IEEE Expert, pp.23-30 (1986)1986.

4.2 深い推論としてのシミュレーション

4.2.1 深い知識とシミュレーション

「深い知識」とは、既に記したように、経験的なレベルの知識に対比して、基本的な事実や一般的な原則を指すものとされる。［科学技術庁85］対象とするシステムの機能や構造に関する知識、システムを支配する原理・原則関する知識は、深い知識であることになる。たとえば、エンジニアリング分野におけるシステムでは、深い知識が利用可能な場合が多い。この分野では、知識処理で深い知識の必要性が注目される以前より、数値シミュレーションの中で深い知識が用いられている。したがって、シミュレーションは、深い知識を用いた深い推論の一例である。ここでは、シミュレーションという言葉を広い意味に用いており、具体的に自然現象を模擬した数値実験だけでなく、何らかのモデルに基づいた手続きによる数値計算と言い換えてもよい。

ここでは、深い推論としてのシミュレーションに注目して、数値処理との結合による知識処理の複合化と、それを支援するためのシェルについて検討する。

エンジニアリング分野では、システムの設計にしても診断にしても、シミュレーションを行ないながら問題解決を進めることが少なくない。エンジニアは、通例、過去におけるシミュレーションの経験から、システムに関する知識を有しており、少ない回数のシミュレーションの結果から、要領よく問題を解決している。問題によっては、この種の知識で解の見当がついてしまい、シミュレーションは、エンジニアが頭の中で出した答を確認するために使うツールとなっている。

このとき、エンジニアが使っている知識は、シミュレーション・モデルの知識よりも「浅い知識」ということになる。しかし、このような推論プロセスで使っている知識でも、何らかのモデルに基づいたものである場合には、やはり深い知識ということになる。エンジニアは、このとき、従来の数値シミュレーションとは異なるレベルで、頭の中でモデルを用いたシミュレーションを行なっていることになる。定性的シミュレーションやメンタル・モデルによる推論は、このような場合にエンジニアが行なっている推論を知識処理により実現するための技術であると考えられる。

エンジニアがシミュレーションを用いて問題解決をするプロセスで、従来の計算機利用技術が有効でない部分として次のようなものを挙げることができる。

(1) 対象とする問題に類似した問題の解決に用いた、過去のシミュレーションの経験から得られた知識を問題解決に役立てる。

(2) 問題解決のプロセスに対応して必要なシミュレーションを実行し、その結果を解釈して解決策を検討することにより、効率的に問題解決のステップを進める。

エキスパートシステムのためのシェルで深い知識を利用していく場合、上記二点

に関連して、次のような基本的課題がある。

(a) 知識の獲得

深い知識がシミュレーション・プログラムという形で利用できない場合、どのように深い知識からエキスパートシステムで利用し易い形の浅い知識を作り出すかが課題となる。

(b) 知識処理の統合

シミュレーション・プログラムが利用できる場合、どのようにシミュレーションを知識処理の中に組み込んでいくか、逆に、シミュレーションの中で知識処理を使っていくかが課題となる。

前者に対しては、前節で検討している。ここでは、後者に関して検討を試みる。

4.2.2 知識処理を用いたシミュレーション

シミュレーションは、現在、計算機を用いた問題解決方法の中心であり、数値シミュレーションだけを取り上げても、エンジニアリングの分野で広く用いられている。例えば、エンジニアは、システムの設計、計画、検証、運用、診断、保守と種々のフェーズでシミュレーションに基づいて問題解決を行なっている。問題解決の上で知識処理とシミュレーションを統合する目的は、基本的には、図4.2-1 に示すように数値演算と記号演算を組み合わせることにより、高い処理能力と広い適用範囲を備えた計算機利用システムを実現することにある。[Kitzmiller 86] 現状においては、種々の観点から知識処理とシミュレーションの境界領域において研究開発が進められている。ここでは、特に、診断とか設計とかいうように目的を限定せずに、知識処理がシミュレーションの分野でどのような形で役立ち得るのかを検討する。

既往の研究事例により、どのようなニーズから知識処理をシミュレーションに応用しているのかを整理すると次の五項目を挙げることができる。[Kerchoff 86]

(a) シミュレーションを含めた複雑な問題解決プロセスの制御

(b) シミュレーションのための助言システムの提供

(c) シミュレーション言語に替わる新しいシミュレーション方法の提供

(d) 複雑なプロセスのシミュレーションの実現

(e) シミュレーションによる知識ベースシステムの評価

これらの項目ごとに、知識処理のシミュレーションへの応用を概観する。

(1) 複雑な問題解決プロセスの制御

知識処理を用いることにより、シミュレータを含み、いくつものステップにわたる処理の流れを適切に制御して効率的に問題解決が可能となるようにできる。処理のステップの単位は、エンジニアの指示によるパラメータの設定であったり、シミ

ュレーションによるパラメータの決定であったりする。
システムのモデリングのプロセスでは、パラメータ決定のためのネットワークに従って処理ステップを制御し、効率的に処理を進める方法を採ることがある。このようなプロセスで、利用可能な知識は、次の二種類に大別できる。

・モデルの数学的特性に関する知識
・モデリングのプロセスで、どのパラメータを決めるのかという選択に関する知識

前者に関しては、原理・原則が有り、一般的な知識として利用可能である。後者に関しては、モデル決定のプロセスが、対象としている分野や個々の問題に大きく依存しているため、経験に基づく知識が必要となる。例えば、パラメータを決めるためのネットワークを展開していく場合については、次のようなヒューリスティックな知識を用いてガイドすることになる。

・複数のルールが適用可能となり、ネットワークの中で進むべき方向について判断が必要となる場合の対策
・何の結論も出ない手詰まり状態となった場合の対策

パラメータ決定のためのネットワークに従って処理ステップを制御していくというプロセスは、システムの診断を扱う場合に異常原因を同定するためのプロセスや、システムの設計を扱う場合に設計パラメータを選定するためのプロセスと共通する面が多い。

知識処理を計画問題に応用した事例を紹介する。原子力発電所の燃料取替計画作成エキスパートシステムPONDは、シミュレーションを行ないながら、新たに装荷すべき燃料の量や、装荷パタンと呼ばれる燃料の配置を決定する計画業務を支援する。このシステムの例では、処理の流れは図4.2-2 のように整理できる。［福崎87］この手順の中で，破線の枠で示したブロックは、燃料移動順序解析、炉心特性解析等のシミュレーション・プログラムによる処理ステップを表わしている。この手順の処理の流れは、シミュレーション効果を反映して制御され、計画の改善が行なわれる。ここで、計画の改善が必要となる場合に、燃料取替計画の上で基本的な取替方針の修正をするのか、装荷パタンの修正をするのか修正のレベルについて判定する。この判定に基づいて、処理ステップを切り替えて適切なガイド情報を提供できるようにしている。燃料取替計画を記述するパラメータには、相互の依存関係を定義しておき、これらのパラメータの修正に際して、関係依存型のバックトラッキングにより、データの間の整合性を維持している。PONDシステムでは、計画案の改善のプロセスで必要となる処理の流れの制御を高度化するために、知識処理を応用している。

(2) シミュレーションのための助言システム

エキスパートシステムの基本的な応用形態に、質問回答や助言を行なうシステムがある。シミュレーションに関しても、種々のフェーズにおいて、専門のエンジニアの知識に依らざるを得ない問題があり、このようなエキスパートシステムの開発が重要な課題となっている。例えば、文献[Bennet 78] のSACON システムや文献[Holt 86] のADEPT システムは、シミュレーションに関連したエキスパートシステムの典型的な例である。SACON システムについてはよく知られているので、以下にADEPT システムを簡単に紹介する。

構造解析の問題では、解析対象の幾何形状を有限要素で近似することがしばしばあらう。このように近似した結果は、構造解析プログラムの入力データとして用いられることになる。このとき、考えている対象について適切にメッシュを切るためには、有限要素法一般に関する知識とともに、今使おうとしているプログラムの詳細仕様に関する知識が必要となる。ADEPT システムは、このような知識を使ったエキスパートシステムと従来の幾何モデリングシステムを結びつけたものである。

解析対象が与えられると、ADEPT は、対象の幾何形状から導かれる特徴と、システム使用者から入力された情報を組み合わせて、自動的にモデリングの手順を決めるとともに、構造解析プログラムに合わせて入力ファイルを作成する。プロトタイプを開発した段階であるが、次のような機能を実現することを目標としている。

(a) 最も適した構造解析プログラムの選択に関して助言
(b) 選んだ構造解析プログラムの制約の範囲で、最も適切な要素のタイプに関して助言
(c) 選んだ構造解析プログラムで利用可能なタイプの要素を用いて、解析対象メッシュを生成
(d) 入力データファイルを作成し、選んだプログラムをラン

この他にも、シミュレーション分野で助言システムに求められる機能として、次のようなものを挙げることができる。

(e) 解析プログラムの入力データをチェック
(f) シミュレーション結果の評価を支援
(g) シミュレーションのランでエラーを生じたときの対策法をガイド

プロトタイプの知識ベースに格納している知識は、有限要素法のテキストにある知識がほとんどであるが、次のようなモデリングについてのヒューリスティックな知識を加えていく必要がある。

（イ）ある計算精度を得るために必要な要素の数を推定するための知識
（ロ）不要な詳細幾何形状を無視して解析を簡素化するための知識

エキスパートシステムのほとんどは、知識のタイプとして、問題領域で既に得られている知識だけを扱っている。しかし、助言システムをモデリングやシミュレーションに広く応用して効果的に利用していくためには、モデルベースや実験結果の

データベースを加えることも検討する必要がある。

(3) 新しいシミュレーション方法

知識ベース型シミュレーションでは、シミュレーションのモデルの中に知識処理を取り入れる。シミュレーションのモデリングの上で知識処理を加えて自由度を大きくすることにより、次の二つの利点が期待できる。［Kerchoff 86 ］

(a) 従来手法の弱点をカバーできること

従来用いられているモデリング手法は、次の二種類に分けられる。

・拡張したプログラミング言語

この中には、通常のプログラミングで用いる構成要素の他に、シミュレーション向きに待ち行列、事象等の構成要素を追加したタイプのものと、前処理プログラムやサブルーチンを追加したタイプのものがある。

・専用のプログラム・パッケージ

プログラミングのためにあまり多くの知識を必要とせずに、モデリングできるようにしたソフトである。このようなパッケージは、ペトリネットとか待ちのネットワークとかのような特定のアプローチを基礎としているために、手法上の限界がある。

これらの従来手法には、共通して次のような弱点がある。

①モデル構造を表現する上でフレキシビリティが小さい。

②プログラミング作業量が大きい。

③バッチ処理での応用に偏り過ぎている。

これらの弱点をカバーできるモデリング手法として、シミュレーション・モデルに含まれる知識を知識ベースの形で表現し利用することが考えられる。

(b) 種々の知識表現を利用できること

モデリングに知識処理を応用することにより、対象とするシミュレーションに応じて、種々の知識表現が利用可能となる。例えば、論理表現、手続き表現、ネットワーク表現のためのスキーマが使える。問題によっては、LispやPrologで直接シミュレーション・プログラムを書くことも有効であるかもしれない。特に、オブジェクト指向の知識表現は、シミュレーションのモデルを記述するための最も人気のある方法とも言える。最近の文献においても、オブジェクト指向に基づく知識ベース型シミュレーションの試みは散見される。これは、オブジェクト指向言語の源流にシミュレーション言語があることからも、自然な帰結であると考えられる。

(4) 複雑なプロセスのシミュレーション

複雑なプロセスやシステムのシミュレーションには、知識処理を応用する余地が大きい。問題によっては、それぞれが何らかの知識を用いたタスクを実行する複数

の独立した意思決定のためのメカニズムが関与するようなシミュレーションを考える場合がある。一つ一つの意思決定のメカニズムは、時には、不確実であったり、不完全であったり、歪曲した情報のもとで、計画を立案したり、スケジューリングを行なったり、仮説を生成したりする。また、大規模なプロセスやシステムのシミュレーションを扱う場合がある。そのような場合、システム分析や数値計算の手法だけでは充分とは言えない。

このようなシミュレーションを行なうためのシステムでは、意思決定というようなヒューリスティックでかつ記号表現に適するプロセスは、知識ベースでモデル化し、物理的プロセスは、従来からのシミュレーション手法でモデル化することになる。

(5) 知識ベースシステムの評価

シミュレーションが知識ベースシステムを評価するためのツールとして用いられる場合もある。例えば、知識ベース型の制御システムを考える。このシステムを実際のプラントに接続して直接試験できることもある。しかし、制御対象とするプロセスによっては、このような直接的な試験が不可能であったり、不適当であったりすることも多い。この場合、制御システムをプロセスのシミュレーションモデルに接続して試験することができる。

4.2.3 シミュレーションを結合したシェル

(1) 設計支援における知識処理の応用

知識処理を用いることにより、シミュレーションという形で使われていた問題解決の処理能力を向上させることが可能となる。シミュレーションと関連の深い設計自動化や設計支援の分野においても、知識処理の応用研究が進められている。[大須賀85] ここでは、設計への応用を目的としたシェルの研究事例を概観し、シミュレーションを統合したシェルについて検討する。エンジニアリング分野でのシミュレーションを用いた問題解決は、設計とは限らない。しかし、設計の問題では、設計の改善の上で試行錯誤的なプロセスが果たす役割が大きく、シミュレーションは繰り返し行なわれる性格をもつので、知識処理とシミュレーションの連携を密とせざるを得ない事情がある。

設計対象とするシステムの設計案の評価にシミュレーションが不可欠であり、その結果に基づいて、設計案を繰り返し改良するような設計プロセスを考える。これは、次の三つのステップから成ると見なせる。

①仕様の記述：設計案を表わす入力パラメータの値を選定する。

②シミュレーション：シミュレーション・プログラミングをランして設計案の性能を評価する。

③結果の解釈：性能評価の結果を解釈して必要であれば設計案の改良の方針を決定する。

知識処理とシミュレーションを統合したシェルでは、この①－②－③の繰り返しを効率的に行なうことにより、処理のスピードアップをはかったり、最適化という面から設計案の品質の向上をはかる機能が求められることになる。この①－②－③のループを行なうシステムの基本構成を図4.2-3 に示す。

(2) ENGINEOUS システム

このようなアプローチをするシェルの研究事例として、ENGINEOUS システム [Nicklans 87] を紹介する。ENGINOUSは、コンプレッサの最適設計を行うためにGE社が開発した設計支援シェルである。コンプレッサの設計には、図4.2-4 に示すようなシミュレーション・プログラミングを用いる。この図で、矢印はこれらのプログラムを用いた場合の処理の流れを表わしている。処理ステップに再び戻っている矢印は、設計案の評価を反映して適切な所まで戻りシミュレーションを繰り返すことを意味している。ENGINEOUS システムのない状態では、この処理の流れを制御し、効率的に設計案の改良をおこうためには、エンジニアが介入する必要があった。このシェルでは、そのような場合にエンジニアが取る手順や、使う知識を用いて、設計の支援を行なう。これらの知識は、ルールとオブジェクトを用いて表わしている。

このシステムを用いると、処理の流れは次のようになる。

(i) 初期仕様でプログラミングをラン

(ii)ルールを使って、設計案を改良できるパラメータ修正の候補をリストアップ

(iii) プログラムをランしパラメータ修正をテスト

(iv)設計目標に合うパラメータ修正が見つかったら、そのパラメータに関して制約条件のもとで最適化

(v) ルールが使えれば、ルールを使って探索を実施

(vi)ルールを使ってもパラメータ修正の候補が出ないときには、ルールが与えられていないパラメータを用いて探索を実施

(vii) それでもうまく行かないときは、探索続行のためのヒューリスティックスを使用

このシステムで用いている知識の表現スキーマは、基本的にはオブジェクト指向である。オブジェクトの例を以下に記す。

(a) Parameters

サブクラスとして、input, variable input, output, limited, sysmbolic, array parameters がある。

(b) Goals

設計目標はパラメータの値を増加させる、あるいは、減少させることに設定する。すなわち、下記のいずれかに帰着させる。

(INCREASE parameter) or

(DECREASE parameter)

(c) Program

シミュレーション・プログラムの入力データに関する情報

(d) Program-Sequence

プログラムの実行順序、前のプログラムへのループ等に関する情報

(e) Rules

ルールの例としては、次のようなものがある。

To accomplish (INCREASE EFF).

if T then ((DECREASE α)(INCREASE β))

ここで、EFF : adiabatic efficiency

α : impeller abs. exit air angle

β : backsweep angle

である。このルールの意味は、「EFF を増加させるという目標達成のためには、どんな設計案に対しても（if Tで指示）、αを増加させるという目標、βを減少させるという目標を設けなさい。」ということになる。

このシステムでは、主として、設計の最適化のための知識処理を用いている。すなわち、エンジニアが行なうように知的に設計案を探索させるために、種々の工夫を重ねている。以下に、探索手法における工夫点を整理する。

（イ）Suspension

・目標と反対の方向の結果が出たら、パラメータ修正は一時中止。

・続いてSusupension となったら、長いステップの間、中止。

（ロ）Dynamic Delta Adjustment

・パラメータ修正のための増減幅の初期値を自動決定。

（ハ）Numerical Technique

・パラメータの微係数を局所的に線型近似。

・近似の更新、適用範囲を管理。

・微係数計算を目的関数、制約違反変数に限定。

（ニ）Rule Look-Ahead

・1パラメータの修正では制約違反なしに改善できない状況で適用。

・違反量との関連で有利なパラメータ修正を選択。

・トレードオフには、現データのみによる方法と計算追加による方法を併用。

（ホ）Monte Carlo Methods

・局所最適解から逃げるためのランダム探索。

・固定のステップ数だけ最適解から移動。

（ヘ）No Backtracking

・Rule Look-Ahead とMonte Carlo で局所解の問題をクリア。

（ト）Relations Between Parameters

・別のプログラムの変数の間の関係をlisp風の形式で定義可能。

（チ）Avoiding CAE Analysis Halts

・プログラム実行時エラーを検出、パラメータ修正幅を変更。なお、CAE とは、Computer-Aided Engineeringの略であり、ここでは数値シミュレーションと同義である。

（リ）Meeting Constraints

・制約条件を満足させることを目標に設定し、満足させたときに目標から削除。

(3) シミュレーションを統合したシェルの実現に対する課題

以上、紹介したENGINEOUS システムは、コンプレッサの機械設計という適用例からもわかるように、ルーチン設計に対する設計支援を対象としている。知識処理をルーチン設計に応用するための枠組を検討した例として、文献 [Brown 86] のAIR-CYL システムがある。ENGINOUSシステムは、シミュレーションを陽の形で扱ったシェルであることに特徴があり、参考とすべき点も多い。ここでは、シミュレーションを統合したシェルの実現に対する課題について整理する。

(a) 設計改善のための試行のプロセスの支援

シミュレーションを行ないながら設計（計画）案を作っていく例は多い。したがって、個別の設計問題に対応した支援システムを作成する際に、domain independent なシェルが利用可能であれば有用である。その場合に、シェルの側でどのような支援機能をサポートすべきかを明確にする必要がある。例えば、AIR-CYL システムでは、試行に際してのエラーの取扱いに重点を置いている。

(b) 不完全・不正確な知識の使用

ENGINEOUS システムでは、ルールは不完全・不正確なものと割り切って使用している。ルールを使ってもよい結果が得られないとき、ルールを使えないときには、アルゴリズム的な手段でバックアップする方法を採用する。設計案の改善には、基本的には、浅い知識と深い知識を組み合わせて使用している。浅い知識で方向（ローカルな設計目標）を決め、深い知識でどこまで進むか（パラメータ修正量）を決定する。ここで、浅い知識のバックアップには、最適化アルゴリズムという形の深い知識を用いている。

(c) 複数の解の取扱い

ENGINEOUS システムでは、局所最適解を回避する所に重点を置いている。ルーチン設計の場合には少ないかもしれないが、観点の異なる解をいくつか用意し検討す

ることもある。

(d) 設計プロセスの記述と制御

設計改善のプロセスでは、現存の設計案の状況とともに、修正の履歴や修正の方針を的確に表示してエンジニアを支援する必要がある。設計改善の試行にエラーが生じてバックトラックするような場合、この機能は重要であり、前述のように、AIR-CYL システムでは、この点を重視している。ENGINEOUS システムでは、バックトラックはないとしているが、目標の変更という形で同様な処理を行なっている。問題によっては、バックトラックの処理量が大きくなるような場合もあり、シェルの側で効果的な戦略をサポートすることを検討する必要がある。

(e) シミュレーション・プログラムのバージョン管理

このようなシェルでは、シミュレーション・プログラムのモジュール性が高くなり、シミュレーション・プログラムを容易に変更できるようになる。プログラムのバージョン管理に対して、シェルの側でも留意する必要がある。

〔参考文献〕

[科学技術庁　85]　科学技術庁（編）：「知識ベースシステム」、大蔵省印刷局（昭60-12）。

[Kitzmiller 86]　C.T.Kitzmiller and J.S.Kowalik: "Symbolic and Numerical Computing in Knowledge-Based Systems." In J.S. Kowalik (ed.) "Coupling Symbolic and Numerical Computing in Expert Systems," PP.3-15, Elsevier Science (1986).

[Kerchoff 86]　E.J.H.Kerchoff et al.:"General Considerations on AI Applied to Simulation," held at Univ. of Ghent, Belgium, on Feb.25-28, 1985, pp.ix-xii, Society for Computer Simulation (1986).

[福崎　87]　福崎孝治　他：「BWR 炉心運転管理支援システムの開発（Ⅲ）」、日本原子力学会昭和62年年会予行集、D55（昭62-4）。

[Bennet 78]　J.Bennet et al.:"Sacon: a knowledge-based consultant for structural analysis, "Stanford Computer Science Report HPP-78-23, Stanford Univ. (1978).

[Holt 86]　R.H.Holt and U.V.L.Narayana:"Adding Intelligence to Finite Element Modeling, "Proc. of IEEE Computer Society Symposium "Expert Systems in Government", held at Mclean, Virginia USA, on Oct.22-24, pp.326-337(1986).

[大須賀　85]　大須賀節雄：「次世代CAD/CAM のための知識処理の応用」、マグロウヒル（昭60-11）。

[Nicklans 87]　D.J.Nicklans et al.:"ENGINEOUS:A Knowledge Directed Computer Aided Design Shell," Proc. of The Third Conf. on Artificial Intelligence Application sponsored by IEEE Computer Society, held at Kissimmee, Florida USA, on Feb. 22-27, 1987, pp.308-314 (1987).

[Brown 86]　D.C.Brown and B.Chardrasekaran:"Knowledge and Control for a Mechanical Design Expert System." IEEE Computer, Vol.19, No.7, pp.92-100 (1986).

5. 知識システムにおける並列処理

5.1 はじめに

エキスパートシステムに代表される知識システムの実用化検討が進むに従って、大規模な知識システムを高速に実行する計算機への要請が高まっている。

知識システムは膨大な計算量とメモリ資源を消費し、パターン照合やリスト操作を行う記号処理時間が大半を占める。記号列の処理では一般に大量データへ不規則なアクセスが頻繁に行われるので、従来の汎用ベクトル型計算機では飛躍的な高速化は望めない。そこで注目されているのが並列処理による高速化である［小池　86］。

並列処理による高速化のポイントは、対象問題における並列性の開拓にある。知識システムでの並列性の開拓では次の二つのアプローチが注目される。

(1) プロダクションシステムに代表される知識表現モデルに基づく並列性の追求。

(2) コネクションマシンに代表される既存の並列処理向きハードウェアの応用による並列処理実現。

前者においては、表 5-1に示されるような意味ネットワーク、論理型表現、プロダクションシステム、黒板モデル等に対する知識表現モデルから見た並列処理が検討されている。

後者においては、表 5-2に示されるＡＩ向きマシンの中で試行が始まっている。

本報告書では、以下 5.2節でOPS5に基づくプロダクションシステムの並列処理、5.3節でコネクションマシンの応用による並列処理についてそれぞれの現状と問題点及び今後の課題について述べる。

5.2 プロダクションシステムの並列処理

5.2.1 概要

プロダクションシステムの並列処理に関しては、以下の機関で研究が行われている。

(1) Columbia大学

1023個のプロセッサからなる木構造並列計算機DADO 2[Stolfo 87] が、1985年12月から稼動し、OPS 型のプロダクションシステムを並列処理するための各種アルゴリズム実験が行われている。

また、16000 個以上の小規模プロセッサからなる木構造並列計算機NONVON［Shaw 85 ］上でもOPS 型のプロダクションシステムの動作実験が行われている。

(2) Carnegie-Mellon 大学（CMU ）

OPS を用いて開発した各種アプリケーションシステムの分析を基礎に、OPS 型のプロダクションシステムを並列処理する、共通メモリ方式のPSM ［Forgy 86］（Production System Machine ）が研究されている。PSM は32～64個のRISCプロセッサで実現される方向である。

(3) MIT AI Lab

Connection Machineを用いて、MYCIN 型のプロダクションシステムを、並列実行するCIS ［Bielloch 86］（Concurrent Inference System ）が研究されている。

(4) 南カリフォルニア大学（USC ）

グラフ文法に基礎を置くプロダクションシステムモデルを提案し、マルチプロセッサシステム上への効果的なルールの配置アルゴリズムが研究されている。［Tomorio 85］。また、実際にマルチプロセッサシステムの開発も始まっている模様である。

(5) ITT Advanced Technology Center

SIMD型のアレイプロセッサCAP ［Brooks 85 ］（Cellular Array Processor）を用いてOPS 型のプロダクションシステムを並列実行する研究が報告されている。

(6) Haneywell Computer Science Center

データフロープロセッサPESA-1［Ramnarayan 86 ］を用いて、OPS 型のプロダクションシステムを並列処理する研究が報告されている。

本章では上記の研究の内、CMU におけるPSM の研究状況について述べる。なお、OPS 型のプロダクションシステムの、動作アルゴリズムであるRETEアルゴリズム［Fogy 82 ］についての知識を前提とする。

5.2.2 PSM の研究状況

5.2.2.1 並列処理の単位

OPS 型のプロダクションシステムに含まれる並列性については、文献［Gupta 84］に詳しい。以下ではその要約を述べる。

プロダクションシステムは、match, conflict resolution, act の３フェーズを繰り返し実行する。PSM プロジェクトでは、この内、実行時間の約 9割を占め

るmatch フェーズの並列処理に焦点をあてている。ここで、並列処理の単位として、何を独立のタスクとするかが問題となる。以下の２つの代替案を延べる。

方式　1

プロダクションルール毎にタスクを割当てる。即ち、各タスクはそれぞれ１プロダクションルールの条件判定を受けもつ。

方式　2

RETEネットワークのノード毎にタスクを割りあてる。即ち、各タスクは、パターンマッチ、ジョイン等のより小さい単位の処理を受け持つ。

プロダクションルールの条件判定はルール毎に独立の処理可能である。このため、方式１では、タスク間通信が一切不要であるという利点がある。一方、方式２では、タスク間で処理結果の引き渡しを行う必要が生じる。また、小さなタスク（約50～ 100機械語命令を実行）が多数生成されるため、そのスケジューリングオーバーヘッドも無視できない。

方式１の上記の利点にもかかわらず、PSM では方式２が採用されている。その主な理由を以下に示す。

① 方式２を採用し、ノード毎にタスクを割当てると、ルールの条件部に指定された複数のパターンマッチを並列に処理することができる。特に、ルールの実行部での操作（平均して１ルールあたり 5.3個のワーキングメモリエレメントを変更する）を並列に処理できれば、パイプライン的にパターンマッチやジョインを実行することが可能となる。

② RETEアルゴリズムでは、複数のプロダクションルールに共通な条件判定はノードが共有される。ノード共有が性能に与える効果は、約 1.4倍と報告されている。

方式１を採用し、ルール毎にタスクを割当てると、ノードの共有ができないという問題が生じる。

両方式を、CMU で開発された多数のアプリケーションで評価した結果が報告されている［Gupta 84］。32プロセッサを用いた場合、方式１が6.22倍の性能向上であるのに対し、方式２は 13.95倍（注）の並列化効果を達成している。

5.2.2.2 PSM のアーキテクチャ

PSM のアーキテクチャは、前節の検討結果をふまえ、以下に示す主張をもとに設計されている（図5-1 参照）〔Gupta 86〕。これらの主張はSIMD型高並列マルチプロセッサアーキテクチャを採用しているDADO、NONVON、CAP 等のシステムコンセプトと大きく対立するものである。

① プロダクションシステム用計算機は、共有メモリ方式の32～64台のマルチプロセッサシステムであるべきである。

② 個々のプロセッサは、小量のプライベートメモリとキャッシュを持つ高パフォーマンスのもの（例えばRISCプロセッサ）でなければならない。

③ 各プロセッサは共有メモリに複数本の共有バスで結合されているべきである。

④ プロダクションシステム用計算機は、ハードウェアのタスクスケジューラを備えるべきである。

PSM の性能評価結果を図5-2 に示す。単純に並列度を計算すると、32ノードで平均 15.92倍の性能向上が得られるが、種々のオーバヘッドを考慮すると、実際の性能向上は、8.25倍であると報告されている。

5.2.3 まとめ

(1) PSM の一連の研究は、CMU でのOPS の豊富なアプリケーション経験の分析に基礎を置いている。この点が他機関の研究に比べての強みであり、プロダクションシステムの並列処理研究の基点となっているゆえんである。逆に、既に開発したアプリケーションの特性に研究が縛られているという欠点も指摘されている。

(2) PSM の評価結果は、あまり魅力的なものではない。8.25倍という数字は、OPS5（lisp版）からOPS83 への、ソフト的努力の成果（約30倍と報告されている）と比べても大きなものではない。しかし、この結果をもってプロダクションシステムには並列処理は適さないと断定するのは早計であろう。例えば、以下の検討が必要である。

（注）文献[Gupta 86]では 15.92倍と報告されている。

① 複数のプロダクションシステムを並列に発火させること［Bielloch 86］［Tonorio 85］［Ishida 85］は、並列処理可能なワーキングメモリ操作を増大させる。並列発火のモデルとその効果の測定が必要である。

② プロダクションシステムの並列処理は問題の性質によって大きく異なる。並列処理に適する問題領域の明確化が必要である。

5.3 並列処理向ハードウェアConnection Machineの応用

5.3.1 背景

知識システムへの並列処理の応用には色々な可能性が考えられる。既存の知識処理の中で並列処理可能な部分を同定しその高速化を図るという方法もあれば、並列処理のもたらす高速性に着目し全く新しい知識の処理方法を考えるというやり方もある。また、並列処理にも色々あり、並列度の大小、並列に動く処理素子の同質・異質性等によっても話の進め方が変わってくる。ここでは、この多様なスペクトラムの中で、同質処理素子による高度多重並列処理が知識処理にどのような影響を与え得るかについて、比較的具体的な成果が出はじめているConnection Machineに関する最近の研究動向をまとめる。この種の研究分野を広く特徴づけるキーワードとして"Connectionism" という言葉があるが、本稿ではその思想背景［Fahlman 87］等には立ち入らず、単にConnection Machineのようなものがあれば知識システムの世界で何ができそうかという点に話を絞る。

5.3.2 Connection Machine

Connection Machineとは2^{14}から2^{16}個の処理素子が高速通信網で相互に接続されたもので、表5-2 に示すとおり、意味ネットをその基本モデルとする。この通信網により各処理素子は他の任意のあるいは全ての素子と並列に通信することができる。Connection Machineには８個までFront-end Machine をつけることができる。今までに作られた多くのConnection Machineでは、処理素子は4K bitのメモリをもち、逐次ビット処理を行う能力をもつ。即ち、メモリから取り出した２ビットとレジスタから取り出した１ビットの間で演算し、その結果の１ビットをメモリへ、１ビットをレジスタへ書き込む。この処理に約１マイクロ秒かかる。

最近、Connection Machineの応用に関する最初の成果が出始めている。［Belloch 86］［Waltz 87］［Stamfil 86］。それらを本報告書の立場から見ると２通りの利用方法が考えられそうである。１つはCIS のようにMYCIN 流のルール・ベースシステムをConnection Machineの特徴を活かすように並列ネットワークに展開し、その推論をほぼ瞬時に達成しようというものである。もう１つは、Memory-

Based Reasoning と呼ばれるもので既存のエキスパート・システムのように経験をルールとして抽象化しそれを利用するというのではなく、経験を単に事実群として記憶し、問題を記憶された事実と照合し最もよく似ているものを求めることにより解を得ようとするものである。つまり、Memory-Based Reasoningでは、Connection Machine の特徴を、より直接的に活かしている。以下、それぞれについてより詳しく見ていくことにする。

5.3.3 CIS

CIS (Concurrent Inference System) [Bielloch 86] では、図5-3 のようなMYCIN 流のパラメータとルールが与えられると、それらを図5-4 のネットワークに翻訳する。そして、さらに図5-4 のノードをConnection Machineの処理素子に写像する。たとえば、今推論の対象となっているanimalについて、そのcoveringがhairだとわかったとしよう。その情報は、Front-end マシンよりConnection Machine中の図5-4 の1)に対応する処理素子へ値１が送られることで伝えられる。1)に１がくるとそれに確信度に相当する重み１をかけて2)に伝えられる。ここで2)と2)' は背反であるので共に１になっていないことがチェックされたのち、さらに2)から出ている矢印の先に夫々の重みをかけた上で伝えられる。その１つは3)でこれは

if (covering hair) then (animal-class mammal .95)

のルールを前向きに辿るためのノードである。今このルールには条件が１つしかないが、複数の入力がある場合は、それぞれの重みの最小値が出力される。（これはMYCIN のAND の計算規則に対応している。）その出力にこのルールの確信度0.95をかけたものが4)に伝えられる。この結果、animal-classがmammalであるという推論が確信度0.95で為される。今の例では矢印を上から下向に辿ることにより前向き推論を行ったが、下から上への矢印を辿ることにより後向き推論も同時に行ない得る。通常のネットワークはこの例より複雑であるから、値の伝播は常時複数個起り得る。CIS ではこの伝播が並列に処理されるので、推論が高速に処理されることになる。試算では、ルールが100,000 個あっても、事実が１つ変化したときそれをネットワーク全体に伝えるのに２秒以下で済むという結果が得られている。

5.3.4 Memory-based reasoning

Memory-based reasoning[Waltz 87][Stanfil 86]はCIS とは異なり、ルール・ベース・システムのあり方そのものに異を唱え、その欠点を克服しようとするものである。Waltz 等によると、ルールとして抽象化する作業（知識獲得）の困難さ、ルールが１つでも足らないと突如として機能しなくなる問題、常識をルールとして表現することの難しさ等を考え合わせると、この方法の有効範囲は極めて狭いように見える。たとえば、人間の日常的判断の多くは、ルールを試行錯誤的に適用し代替

案を検討するというよりは、記憶に頼り理由無しに瞬時に為されることが多いように見える。このことは計算機上で知的な情報処理を行う際に何か別の（補完的な？）パラダイムが必要であることを示唆しているとも考えられる。この立場にたち、別のパラダイムの１つの候補として提案されたものが、Memory-based Reasoningである。医療診断のシステムQuack の例をとって、この考え方を説明しよう。Quack では過去の患者のデータ（症状、属性、テスト結果、診断等）が各処理素子に１つずつ格納されている。このとき、与えられた新しい患者に関し次の処理が試される。

(1) 新しい患者のデータをすべての処理素子に（Front-end マシンから）放送する。
(2) 各処理素子は自分の持つ患者データと比べてその近さを報告する。
(3) その報告をもとに新しい患者に最も近い患者データをいくつか取り出す。

ここで、次のような場合が存在し得る。

(a) 取り出されたｎ人の患者がすべて同じ診断を受けている
(b) 新しい患者に似た症例が１つもない
(c) 新しい患者に似た症例がほんの少ししかない
(d) 取り出されたｎ人の患者に対し幾つかの異なった診断が為されている

この結果をもとに次の仮説検定フェーズに入る。

(a) →ｎ人と同じ診断が成り立つ可能性が高いと結論する。
（この場合からルールを抽出できる可能性もある）
(b) →診断不能（システムは自分が“わからない”ということを知っている）
(c) →根拠は薄いが仮の診断をする。

実際には(d) のようなケースが多いと考えられるので、この場合についてもう少し説明する。上記(2) で患者データの類似度を計算したが、実はこれは患者データのどこに着目するかで大幅に値が変わる。たとえば、インフルエンザの疑いがあるときは性別は関係ないが、妊娠の有無を診断しようとするときには性別に大きな重みが与えられなければならない。したがって、Quack は患者の診断に対し仮説を立て（一番最初は“健康である”という仮説）その仮説とどの症状やテストが相関が高いかを全データをもとに計算する。（インフルエンザにかかった男性の割合と女性の割合に差があるかというように）。そして相関の高い項目に重みを与えて(1) の放送を行う。多くの場合、最初の仮説は成立せず(d) でいくつかの診断仮説が提案されることになるので、さらにそれらの仮説をもとに(1) ～(3) をやり直す。この過程を繰返すことにより、仮説が自分自身を可能な診断の１つとして返すようなものがいくつか生き残る。その次のステップでは、それらの生き残った仮説のどれが正しいかを検出する能力の一番高い症状やテスト結果に重みを与えて(1) ～(3) を繰返し、結論を出す。

このようにQuack は通常知識ベースと呼ばれるものは何も有しない。あるのはただ膨大なデータと、いくつかのWeak Method だけである。したがって新しいデータが与えられたときも単にそれを格納するだけで済むし、ルールの抽出やその確信度といった問題に悩まされずに済むという利点をもつ。ルールはある種の規則性を抽出したものであるから、そのような規則性が見出せない場合には無力であるが、Memory-based reasoningの場合には類似のデータが１つでもあれば某かの結論を引き出せ、しかも類似のデータがないときには"知らない"ということができる。

5.3.5 まとめ

以上のようにConnection Machineのような高度な並列性をもつ処理機構を知識システムへ利用する方法には色々な可能性がある。ただいずれの場合にもその基本となるのはvon-Neumanマシンの常識では考え付かないような面での高速性である。（Memory-based reasoningがそのよい例である。）この種の並列処理が本当に役立つか否かの判定は今後の研究成果に持たざるを得ないが、人間の情報処理との類比で考えるとかなり有力な機構である可能性が強く、その動向には眼が離せないといえよう。

5.4 おわりに

知識システムにおける並列処理に関して、プロダクションシステムに基づく並列処理と、コネクションシステムに基づく並列処理と、コネクションマシンの応用による並列処理の実現について述べた。

現時点では知識システム向の決定的な知識表現モデルが定まっているわけでなく各種モデルでの並列処理を追求する研究と、並列処理向ハードウェアの研究とが同時進行していると言える。本報告書では、そこでの代表的な例について述べたに過ぎない。今後は、並列処理向き知識表現モデルをベースにした知識システム構築ツールと、その専用ハードウェアの実現を通して、実用規模の応用問題による性能評価の報告が期待される。

〔参考文献〕

[小池 86] 小池誠彦、"人工知能向ハードウェア"、ICOT昭和60年度新世代コンピュータに関する技術開発動向及び適用分野等の調査研究報告書－人工知能（AI）技術・需要の動向－、pp.98-112 (1986)

[Stolfo 87] Stolfo, S.J., "Initial Performance of the DADO2 Prototype", IEEE Computer (January 1987)

[Shaw 85] Shaw, D.E. "NON-VON's Applicability to the AI Task Areas", IJCAI, (1985)

[Forgy 86] Forgy, C. and A. Gupta, "Preliminary Architecture of the CMU. Production System Machine", Hawaii international Conf. on System Sciences (1986)

[Bielloch 86] Bielloch, G.E., "CIS: A Massively Concurrent Rule-Based System", AAAI (1986)

[Tonorio 85] norio, F.M. and D.I. Moldovan, "Mapping Production Systems into Multiprocessors", International Conf. on Parallel Processing (Aug. 1985)

[Brooks 85] Brooks, R. and R. Lum, "Yes, An SIMD Machine can be Used for AI", IJCAI (1985)

[Ramnarayan 86] Ramnarayan R., G. Zimmerman, and S. Krolikoski, "PESA-1: A Parallel Architecture for OPS5 Production Systems", Hawaii International Conf. on System Sciences (1986)

[Fogy 82] Fogy, C.L. "RETE. A Fast Algorithm for the Many Pattern/Many Object Pattern Problem", AI Journal (1982)

[Gupta 84] Gupta, A. "Parallelism in Production Systems: The Source and the Expected Speed-up", Technical Report, Dept. of Comp. Sc., CMU (1984)

[Gupta 86] Gupta, A., C. Forgy, A. Newell and R. Wedig, "Parallel Algorithms and Architectures for Rule-Based Systems", International Symposium on Computer Architecture (1986)

[Ishida 85] T. Ishida and S. J. Stolfo, "Towards the Parallel Execution of Rules in Production System Programs", International Conf. on Parallel Processing (1985)

[Waltz 87] D. Waltz, "Application of the Connection Machine," Computer, Vol.20, No.1, pp.85-97 (1987)

[Stanfil 86] C. Stanfil and D. Waltz, "Toward Memory-Based Reasoning," Comm. of the ACM, Vol.29, No.12, pp.1213-1228 (1986)

[Fahlman 87] S. Fahlman and G. Hinton, "Connectionist Architecture for Artificial Intelligence," Computer, Vol.20, No.1, pp.100-109 (1987)

6．エキスパートシステムシェルのユーザインタフェースに関する一考察

6.1 はじめに

エキスパートシステムに限らずコンピュータシステムのユーザインタフェースの重要性は大いに強調されている。エキスパートシステムにおいても、シュルンベルジャー社のDipmeter Advisorでは推論部が全体に対して8%のコード量なのに対して、ユーザインタフェースが42% のコード量を占めているというのは有名な話である。［Smith 84］エキスパートシステム構築に際しても、ユーザインタフェースの設計には多大な労力を必要とする。しかしその一方でユーザインタフェースはなかなかまとまった議論のしにくい分野である。その理由の一つとしてユーザインタフェースに関する一般論と個別の応用システムに関する議論との間のギャップが非常に大きいことが挙げられる。ユーザインタフェースの一般論に関する議論としては、メンタルモデル［Furukawa 86 ］やユーザモデル［Sleeman 85］がありこれらは特にエキスパートシステムの操作性、説明機能に大いに関係する。しかし、これらの議論はそのままシェルの構造に反映できるところまでには至っていない。一方、応用システムとしてはユーザインタフェースに優れたシステムも多くあるが、それらの経験をシェルの構造にどのようにいかせばよいかという議論はまだ充分にされてはいない。良好なユーザインタフェースは、応用システムの特性、使用するユーザの特性によって大きく異なるためにESシェル機能として取り込むのには難しい面も多い。現在、シェル機能のうち知的インタフェース的なものとしては自然言語テンプレートを用いた説明やグラフィックスアイコンの利用ぐらいであるが、今後より高度な知的インタフェースの出現が望まれる。後に述べるようにATMSやGeneric Taskのように最近注目されているシェルに関する話題はユーザインタフェースとも大いに関連がある。これらの新技術をインタフェースの点から見直し評価してみることも重要である。ユーザインタフェースは何か新しい理論等によって革新的に進歩するものではなく段階的に一歩ずつ進歩するものであるので、地道な研究活動が必要となるだろう。

近年マルチウィンドウやグラフィックス機能を持ったワークステーションの普及が進み、良好なユーザインタフェースをサポートする環境は整ってきた。この結果、マウス、ポップアップメニューなどの利用によって多様な情報の様々な角度からの多重表示、容易なコマンド選択などが可能となり、エキスパートシステムのユーザインタフェースは以前に比べてかなり向上した。しかし、マルチウインドウといっても表示画面の大きさは限られているわけで、この中でいかに効率的に情報の表示を行えばユーザにとって親切なシステムになるかなどの利用技術はまだ充分に進歩してはいない。特にユーザが真に見たい情報だけを表示する知的なフィルター機能が重要である。マルチウインドウ、グラフィックス機能などの環境を考えたときに、どのような知識構造、推論構造ならば良好なインタフェースが構築できるかという観点からの議論が必要である。

個々の応用システムに関してはユーザインタフェースの良好なシステムも多く出現している。応用問題ではこの図があると話がよく通じるというようなものが多くある。例えば、医療診断における医者のカルテ、回路設計における回路図、CAM における工具軌跡図などである。このような応用に固有なインタフェースもシェルのグラフィックスエディター機能によって容易に作成できるようなシェルもある。シェルの機能としては、インタフェース構築を容易にするような仕掛けを多く準備することが重要である。最近のウインドウシステムではユーザが自在にウインドウを設計できるようにパネル、スウィッチ、ボックスなどのパーツを準備するようになってきた。エキスパートシステムにおいてもこのようなパーツという概念（様々な形態の知識や推論の状況などを表示するための部品という意味である）を用いることによって、ユーザが真に見たい情報だけが表示されるようなインタフェースの設計を自在に行えるようになることが望まれる。

6.2 第２世代シェルにおける問題点

KEE [Fikes 85]、ART [Clayton 85] などの第２世代シェルはユーザインタフェースに関しても現在のシェルの中では優れているといわれている。KEE のActive Image、ART のARTISTのようにグラフィックス用のエディターが用意されているものがあり、これらはシェルの一つの大きな特徴になっている。グラフィックス用のエディターを利用すると、アプリケーション固有のアイコンやゲージ類の作成が容易に行え、推論中のデータの変化などを図形的に表示できる。これらは一つの重要な機能ではあるが、インタフェースに関する問題点は多くある。特に推論状況の表示、説明を行うような機能が充分でないと思われる。シェルにおけるユーザインタフェースは汎用性、効率とのトレードオフの問題があって難しい点も多いが、次世代シェルに向けての重要な課題であると考えられる。ユーザにとって親切な工夫を少しでも増やしていく努力を積み重ねていくべきである。第２世代シェルの一つであるART を例にとり、ユーザ側として不満に感じる点について述べる。

① ユーザが真に見たい情報だけを表示することが難しい。

トレース情報の表示やファクトの参照などを行おうとすると非常に多くの情報が表示されることがある。このような場合、ユーザにとって本当に知りたい情報が不要な情報の中に埋もれてしまう。ユーザにとって本当に知りたい情報だけの表示が行えるような知的なフィルター機能が望まれる。

② 常識的な線での表示機能がない。

常識的という判断は難しいが、例えば画面の大きさ、表示時間などの制約から表示が不可能と思われるような表示も実行しようとするために実質的にシステムが止まってしまうことがある。表示量に応じて適当な量の表示だけを行う機能が欲しい。ビューポイント、スキーマの階層構造、ファクト・ルールの依存関係などが図的に表示さ

れるのは分かりやすい。しかし、複雑な構造になると画面上で全部の情報を表示することはできないために、全体的な構造が把握しにくい。ズーミングが自由に行える機能が必要である。

③ 説明機能として履歴情報の表示だけですまそうとしている。

ユーザに対する説明機能としてはファクト間の依存関係などによる履歴情報の表示を主としている。必要な情報の表示機能は全部あるから勝手に見てくれという態度にも思われユーザにとって親切ではない。初心ユーザ・専門家に対してはもっと分かりやすい説明、熟練ユーザ・専門家に対してはもっと深いレベルでの説明が望まれる。

④ 全般的にプロ向きのユーザインタフェースである。

シェルの使い方を覚えるのが非常に難しい。ART の場合、推論制御の大部分はルールによって行われるが、この記述はかなりの高度なテクニックを要する。また、ルールの構造は平坦でありしかもルール中に制御用のファクトなども含まれているために、規模が大きくなったときには知識ベースは非常に見づらいものになる。理解しやすさの点では、パソコン用シェルと天地の差がある感じがする。

6.3 エキスパートシステムの使用形態

ユーザインタフェースを考えるときにはユーザの特性を充分に考慮する必要がある。エキスパートシステムのユーザは使用形態および使用する人のエキスパートシステムに対する習熟度によってその特性が大きく異なり、それに応じて良好なインタフェースも異なったものとなる。シェルとしてはこれらすべてのユーザに対して最適なインタフェースを提供することはできないが、使い方に応じた良好なインタフェースを提供できることが望ましい。シェルとしてこれらのすべての使用形態を対象とするのか、或は対象を限定して一部のみをサポートするのかを明確にする必要がある。但し、汎用シェルを目指すならばプロトタイプ構築には様々なタイプの問題を解決できるような推論方式、知識表現方式を持ち、かつ実システム構築には必要部分をカスタマイズ化することによって効率化できるような機能が必要である。使用形態ごとにどのようなインタフェースが望ましいかを考えてみる。

6.3.1 知識ベース構築時のインタフェース

エキスパートシステムにおけるボトルネックとして知識獲得が挙げられ、知識ベース構築時の専門家支援機能の重要性が強調されている。例えば、プログラミングを支援するＡＩシステム研究としてMIT のKBEmacs [Waters 85] [Anzai 86-2] があるが、ユーザの多様な要求に答えるための知識量は膨大な量となるという問題点が明らかになっている。エキスパートシステムの場合でも、知識ベース構築時の要求は多く

のものがあり、これらすべての要求に答えることは難しい。シェルの知識ベース構築時のインタフェースとしては次のような項目がポイントになる。エキスパートシステムに熟練した専門家に対しては全体的な見通しの良さと専門家が自在に知識を記述できるような枠組みが重要である。エキスパートシステムに熟練していない専門家に対しては分かりやすく入力が容易な表現が重要となる。但し、この場合には知識表現自体は使い方が限定されたものになる。

① エキスパートシステムに熟練した専門家

知識表現としてはAI言語（LISP、PROLOG、オブジェクト試行言語など）に近い形で自由に知識を記述できる方が便利である。また、知識の誤りなどを検出してくれるデバッグ機能が必要である。

② エキスパートシステムに熟練していない専門家

エキスパートシステムに熟練していない専門家でもKEの助けなどを借りずに容易に知識ベース構築が行えるシェルが必要だという議論をよく聞く。しかしこの要求は現実的には実現が難しい。KEなどのエキスパートシステムに熟練した人が問題構造、シェルの構造を良く考慮した上でエキスパートシステムの枠組みを決定するべきである。シェルに熟練していない専門家向けのインタフェースの考慮はそれからである。応用なりにある程度知識表現に制限を設けても専門家が入力しやすいような形式の知識表現機能があったほうが便利であることも多い。例えば表形式からのルール入力などが一例である。専門家用の外部知識表現とシステム用の内部知識表現の区別も必要になってくる。ETS [Boose 84]、MORE [Kahn 85] などで行っているような知識の変換機能を含む知識獲得支援機能が重要となる。専門家としては内部知識表現を意識せずに知識ベースデバッグ等ができることが望ましい。

6.3.2 システム実行時のインタフェース

① エキスパートシステムに熟練したユーザ

何故そのような結果が導かれたかなどの比較的詳細な説明が欲しい。推論の状況、結果などを徐々に詳細化して説明してくれるような機能がいる。エキスパートシステムに熟練したユーザの場合、ユーザがシステムの制御に積極的に関与できるためのインタフェースが重要である。

② エキスパートシステムに熟練していないユーザ

自然言語表現、グラフィックス機能などを用いた表示、ヘルプ機能、対話による説明など分かりやすいインタフェースが重要である。また、ユーザが自分のしたいこと

が容易に行えるような操作性も重要である。ダイレクトな操作感の原則[Shneiderman 83]に乗ったアイコン等による操作性の向上の研究が待たれる。

6.4 知識ベースのデバッグ過程

エキスパートシステムシステムシェルにおけるユーザインタフェースを考えるときに知識ベース構築、問題解決の各フェーズにおいて、どのような構造のシェルであればどのようなユーザインタフェースが可能か、それらはユーザにとって使いやすいものであるかということを洗い出す必要がある。知識ベースのデバッグ機能は、従来のシステムと異なったエキスパートシステム（知識ベースシステム）の大きな特徴であるが、その機能が充分でないために有効に利用されていない場合も多い。ここでは、知識ベースのデバッグ過程を取り上げユーザインタフェースについて考察してみる。知識ベースのデバッグ過程は次のような手順で進められる。

（知識ベース変更）

トレース → ブレーク → 情報参照（説明） → 再実行

エキスパートシステムシェルのデバッグ機能なども実際にはあまり使われていない場合も多い。これは上記の手順中で、トレースやブレークされたときに推論の状況の説明がうまくなされないためである。必要な情報が分かりやすい形で表示されていない、不要な情報が多く表示されているなどの問題点がある。今どういう状況にあるかを説明する推論状況の表示・説明、現在までの推論結果を説明する推論結果の表示・説明、どのような処置をとればいいかを示す操作指示などのインタフェースに問題がある。これらの機能は、グラフィックス機能等の利用によって改善されうる可能性がある。これらの各過程におけるグラフィック機能の有効性は対象問題、問題解決戦略によるが次のような場面で利用可能性がある。なお、グラフィックス機能を用いたインタフェースはオブジェクト指向言語上ではインプリメントが容易である。

6.4.1 推論状況の表示・説明

設計支援や相談型システムなどユーザが推論制御に関するイニシアティブを取りながら推論を行うシステムが増えてきた。これらのシステムでは、今まで以上に推論状況が明確に示されることが重要となってくる。推論状況の表示・説明には、推論モデル自身の表示、推論中に動的に変化するデータの表示、アプリケーション固有の表示などがある。

① 問題解決方法が明示される推論モデル

ユーザに推論状況を分かりやすく示すためには、推論モデル自体が分かりやすく

表示しやすい構造になっている必要がある。ART におけるビューポイント機構などは現在どのような仮説が考慮されているかが良く分かる。ユーザによる代替案の理解、選択指示なども容易である。但し自動マージのデバッグは困難である。BB1 [Hayes-Roth 85] [Mizoguchi 87] は、ドメイン用ブラックボードのほかに制御用ブラックボードの導入を行い問題解決戦略に関する推論も行う。しかも問題解決戦略に関する説明機能も実現されている。また最近、Generic Task [Chandersekaran 86] などの問題構造の類型化手法が知識システム構築の問題として注目されているが、これらはインタフェースの面にも大いに関連がある。定型的な設計問題に対するGeneric Taskでは設計問題における推論構造を、設計におけるパラメータを求めるというゴールを中心に展開している。すでにどのようなゴールが生成されているかとか、どのようなゴールを求めようとしているかなどの問題解決の構造は分かりやすいものになっている。

② 変化が分かりやすいデータ

仮説分類の状況、CF値、時系列変化データなどは表現の決まったものがある。これらの表示インタフェースは表示用のパーツの提供によって設計できるのが望ましい。

③ 視覚的に分かりやすい対象問題

対象問題によってはアイコンの使用などによって容易に状況が理解できるような問題がある。一部のシェルではグラフィックスエディターを用いてこれらのアイコンが容易に作成できるようになっている。シミュレーション実験、設計・組立ての問題などが該当する。FFAST [Friedman 86] は書式に基づくアプローチを特色とする業務審査を行うシステムである。書式に基づいたユーザにとって分かりやすいインタフェースを提供することによって、ルール、フレームなどのシステム内部の知識表現が表に出てこないようにしている。また、FFAST ではベースとしているART の機能をインタフェースに生かしており、ビューポイント機構にもとづく代替案の選択や、論理的依存関係に基づく制約などを利用している。

6.4.2 推論結果の表示・説明

① 推論結果の正当性の説明

現在のシェルでは、事実の支持関係の表示などの履歴情報によって推論結果の因果関係の説明を行っている。ART の"justification network" のような図形的な表示は分かりやすい。しかし、ユーザにとっては単なる履歴表示だけでなくより親切な説明機能が求められ、知識内容の正当性に関する説明機能も重要である。エキス

パートシステムに熟練していない専門家・ユーザにとっては、自然言語などを用いたもっと分かりやすい説明が望まれる。エキスパートシステムに熟練している専門家・ユーザにとっては、知識ベース中の誤りなどを発見することを支援する機能が望まれる。XPLAIN〔Swartout 83 〕〔Furukawa 87 第6章〕では、ルールの正当性（ルールが何故正しいかを深い知識を用いて説明する）をユーザに示す説明機能を実現している。

② 推論結果に関する状況の説明

推論結果に関してその結果に関する状況の説明も必要である。例えば、ART のビューポイント機構を用いると、代替案が一つのビューポイントとして明示される。また、これらの代替案同士の比較（"alternative comparison"）も容易であるので、代替案をユーザが選択するようなインタフェースの設計が容易である。ユーザ主導の推論制御を行うためには、推論結果に関する状況の説明、ユーザの選択に関するインタフェースはますます重要になる。

6.4.3 操作指示

特にエキスパートシステムに熟練していない専門家・ユーザにとっても、システムと対話しながら推論制御が行えるようなインタフェースが望まれる。メニュー選択、アイコン選択などによるダイレクトな操作性が必要である。また、ユーザが自分がやってみたい部分だけの実行が行えるような、部分実行、再実行指示を自在に設定できるようなインタフェースも重要である。

6.5 推論モデルとユーザインタフェース

6.4節でも述べたが最近のエキスパートシステムの動向として、診断問題、設計問題などのタイプにかかわらずユーザが積極的にシステムと対話しながら推論の制御を行うようなシステムが多くなってきた。このようなシステムでは、ユーザインタフェースの重要性はますます増加し、特に推論状況をいかにユーザにわかりやすくするかが重要となる。そのためには推論モデル自体がわかりやすいことが必要である。

例えば、ART におけるビューポイント機構を考えてみる。ART ではビューポイントのブラウズ機能の中に指定したリレーションだけを表示する機能がある。この機能を用いるとビューポイントとアサンプションだけを図形的に表示することが可能で、代替案の生成状況などがよくわかる。但し何故そのような状況になったのかという原因はわかりにくく、問題解決戦略自体を説明する機能はない。ユーザが積極的に主導権をとって推論を進めるうえではこの点は不十分である。BB1 などの問題解決戦略自体に関する説明機能は興味深い。

また、Generic Taskなどは問題を類型化することによって知識ベース構築を容易にさせるという利点だけではなく、問題構造が明確化されるという利点もありインタフェースとも大いに関連がある。定型的な設計（routine design）を行うDSPL［Brown 86］というツールでは設計パラメータを求めるというゴールを中心に制約条件や設計が失敗したときの代替案の選択方法などを記述できる枠組みとなっている。同様のアプローチがXerox PARCのPRIDE［Mittal 86］という設計支援システムでも実現されている。これらのシステムでは定型的な設計問題に特有の問題解決戦略が組み込まれている。定型的な設計問題に特化したために推論制御構造は明快なものとなっており、推論の状況の把握が容易なためにユーザが推論制御を行いやすい。Teknowledge 社のABE［Nikkei86］ではGeneric Taskなどを一つのレイヤーと考える知識システム開発環境を提供することによって、ドメインの専門家がKEなどの手を借りなくてもシステムを開発できることを目指している。

〔参考文献〕

[Anzai 86-1]　安西，"インタフェースとコミュニケーション"，bit，Vol.18，No.7，(1986)．

[Anzai 86-2]　安西，"インタフェースからみた人工知能"，bit，Vol.18，No.9，(1986)．

[Boose 84]　J.H. Boose，"Personal Construct Theory and the Transfer of Human Expertise," AAAI-84，pp.27-33．

[Brown 86]　D.C. Brown and B.Chandrasekaran，"Knowledge and Control for a Mechanial Design Expert Systems"，IEEE COMPUTER，Vol.19，No.7，pp.92-100，(1986)．

[Chandrasekaran 86]　B.Chandrasekaran，"Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design"，IEEE EXPERT，FALL，pp.23-30，(1986)．

[Clayton 85]　B.D.Clayton，"The ART programming tutorial"，Inference Corp.，(1985)．

[Fikes 85]　R.Fikes and T.Kehler，"The Role of Frame Based Representation in Reasoning"，Comm. of the ACM，Vol.28，No.9，pp.904-920，(1985)．

[Friedman 86]　J.Y.Friedman and A.Jain，"Framework for Prototyping Expert System for Finantial Applications," AAAI-86，pp.969-975．

[Furukawa 86]　古川，溝口編，"メンタルモデルと知識表現"，共立出版，(1986)．

[Furukawa 87]　古川、溝口編，"インタフェースの科学"，共立出版，(1987)．

[Hayas-Roth 85]　B.Hayes-Roth，"A Blackboard Architecture for Control," Artificial Intelligence，Vol.26，No.3，pp.251-321，(1985)．

[Kahn 85]　G.Kahn，S.Nowlan and J.McDermott，"MORE: An Intelligent Knowledge Acquisition Tool，IJCAI-85，pp.581-584．

[Mittal 86]　S.Mittal，C.L.Dym and M.Morijaria，"PRIDE: An Expert System for the Design of Paper Handling Systems"，IEEE COMPUTER，pp.102-114，(1986)．

[Mizoguchi 87]　溝口，角所，"エキスパートシステムにおける新しい研究動向"，情報処理，　Vol.28，No.2，pp.207-217．

[Nikkei 86]　［ＮＡＩレポート］，　次世代シェル開発へ動き始めた米国ＡＩツールベンダー"，日経ＡＩ，(1986.9.1.)

[Shneiderman 83]　B.Shneiderman，"Direct Manipulation: A Step beyond Programming Languages"，IEEE Computer，pp.57-69，(1983)．

[Sleeman 85]　D.Sleeman，D.Appelt，K.Konolige，E.Rich and B.Swartout，"User Modelling Panel"，IJCAI-85．

[Smith 84] R.G.Smith. "On the Development of Commercial Expert System". AI MAGAZINE. FALL. pp.61-73. (1984).

[Swartout 83] W.R.Swartout. "XPLAIN: A System for Creating and Explainint Expert Consulting Programs". Artificial Inteligence Vol.21. No.3. pp.285-325. (1983).

[Waters 85] R.C.Waters. "The Programmer's Apprentice: A Session with KBEmacs." IEEE Trans. Software Eng.. Vol.SE-11. No.11. Nov.. pp.1296-1320. (1985).

7．エキスパートシステムシェルのカスタマイズ機能

7.1　はじめに

最近、エキスパートシステムシェル（以下では単にシェルという）の種類は我国で入手できるものに限っても、パーソナル・コンピュータ用からメインフレームの大型計算機用にいたるまで、数十種類を越える状況になっており、機能的にも性能的にもさまざまなレベルのものが混在している。そして、単純なプロダクション・ルールのみをサポートするOPS5［Brownston 1985］のような第一世代のシェルにかわって、ART［Clayton 1985］、KEE［Fikes 1985］、Knowledge Craft［Popper 1986］などに代表される豊富な機能をもつ第二世代のハイブリッドシェルが注目を集めている。さらに、対象領域を限定してエキスパートシステム開発を支援するツールの開発もいくつか試みられている。その結果、知識ベースばかりでなく推論エンジンの開発までをシステム開発者が行っていた数年前の状況とは異なり、エキスパートシステム開発にこれらシェルのはたす役割も次第に重要になりつつある。シェルの定義にはいろいろなものが考えられる［情報処理1987］が、本章では、これを広くとらえ、"知識ベースと推論機構を供えたソフトウェアー知識ベースシステムーの開発を支援するためのツール"［寺野 1987a］、［寺野 1987b］と定義する。

本章では、汎用のシェルを利用者が固有の目的で改良するーカスタマイズするー際の問題について考察する。これは、特定分野の応用システムの実現作業にはじまり、各分野向きのシェルの開発にまでつながる幅の広い課題である。本章では、まず、シェルのカスタマイズが、知識技術者、各分野の専門家、エンドユーザのそれぞれに対して、どのような意味をもつのかを考察し、ついで、シェルのカストマイズの事例を紹介する。

7.2　カスタマイズの必要性

現在普及しているシェルの機能から判断するとルールを利用して経験的な知識を記述し、フレームで表現した対象分野の構造的知識と組合せて利用するという手法は、新しいプログラミング・パラダイムとして確立してきたように思える。実際、シェルをうまく利用すれば、知識処理技術の特徴である宣言的知識の記述性のよさ、プロトタイプ手法の適用容易性などとあいまって、狭い範囲の適切な問題に対してはシステム開発の効率は著しく改善することができる。

ただし、実用的なエキスパートシステムを開発しようとする場合、現在のシェルが提供しうる人工知能の知識表現のレベルが不十分であること、また、専門家が日常問題解決に利用する知識とシェルが提供しうる知識表現との差異は非常に大きいことが、しばしば問題となる。そのため、シェルで表現できるレベルまでに専門家の知識を整理するには、人工知能の概念にあわせた知識の強引な変換を行うとともに、シェルそのものの

機能を問題にあわせて変更する必要も生じてくる。シェルをカスタマイズするための視点は、そのターゲットが、知識技術者、各分野の専門家、エンドユーザのどれかであるかによって異なる（図7-1 ）。

知識技術者にとってシェルをカスタマイズする必要が生ずるのは、シェルが提供する基本的な知識表現ではシステム開発に不十分な場合、システムの実行性能を改善したい場合、他言語とのインタフェースをとらなければならない場合などである。これらは、図7-1 に示したように、いずれも、シェルの推論機構ならびに知識表現に手を加えることを意味する。

第１の場合では、特に、シェルが陽に提供していない人工知能のパラダイムを実現することが課題となる。これは、複雑なタスクにおいて知識を表現する際には不可欠である。第２の場合は、推論機能を問題向きに特化して高速化する際に必要となる。このような作業は、プロトタイプの開発が終了し、その実用化をはかるときには重要である。第３の場合は、“深い”システムを作成する際に対象モデルの作成を容易にし、また、数値・記号モデルを利用するシミュレーション機能を実現し、そして、記号処理にふむきな情報処理を行うときに重要である。これらの要件を実現するには、シェルの機能として、基本的な人工知能技法が簡単に扱え、シェルと人工知能用プログラム言語と容易に統合化できることが重要である。そして、知識表現の問題と知識の手続き的な扱いがポイントとなる。

シェルを用いて開発されるエキスパートシステムでは、シェルの提供する知識表現・推論機構などの本来の人工知能研究が提供する技術よりも、専門家の知識のはたす役割のほうが事実上重要である。各分野の専門家にとってシェルをカスタマイズする必要性は、与えられたシェルでは機能が不十分で、エキスパートシステムの行うタスクの内容を（専門家が理解しやすいように）表現し、対象とする問題を適度なレベルで抽象化する場合に生ずる。これには、専門知識の獲得の問題と（エキスパートシステム技術の特徴である）宣言的な知識の扱いとがポイントとなる。これは、図7-1 に示したように、対象とする問題を表現する知識ベースのプロトタイプを開発することに相当する。さらに、上で述べたのと同様に、既存のプログラムをエキスパートシステムで扱う手段を作成するものとなる。

現在のシェルを提供するインタフェースは、他のソフトウェア・システムと同様、不完全であり、シェルを用いて開発したプロトタイプをそのまま使いこなすのは難しい状況にある。したがって、完成したシステムを利用するエンドユーザにとってシェルのカスタマイズが必要となるのは、図7-1 に示すように、システムの利用者インタフェースを改善しエンドユーザにとって真に使いやすいシステムを実現する場合、ならびに、エンドユーザが自らシステムを開発しうる環境を実現する場合に集約される。第１の目的は当然ながらすべての計算機システムに共通のものである。ただし、エキスパートシステムの立場からは、エンドユーザとどのように知的なコミュニケーションを行うか、ま

た、特にプレゼンテーションのための知識をどう扱うかがポイントとなる。また、第２の目的からは、基本的な人工知能技法が簡単に扱え、簡易ソフトウェアなどと統合化できるようにすることが重要である。

7.3 カスタマイズの事例

以下では、上で述べた３つの視点からシェルをカスタマイズした事例を示す。最初の２つは、ともに、プロダクション・システムを対象とした知識技術者むけのカスタマイズ例である。３番目の例はドメインモデルを知識ベース内に構築することによって、専門家むけにシェルをカスタマイズした例である。最後の例では、利用者インタフェースを改善するために、シェルで不足する機能を補っている。

7.3.1 OPS5に基づくカスタマイズ

OPS5はよく知られているようにrecognize-act の概念を忠実に実現した高速のプロダクション・システムであり、多くの実用的なエキスパートシステムで利用されている。しかし、OPS5の基本機能のみでは複雑な処理を行うことが難しいので、プロダクション・システムの枠組みの中で、さまざまな人工知能パラダイムの実現をはかる試みがなされている。その中で、注目すべき例は、IBM ワトソン研究センターで開発されたオペレーティングシステム監視用エキスパートシステムYES/MVS [Griesmer 1984] とCMU で開発された電力系統運用エキスパートシステムTOAST [Talukdar 1986] の２つである。

YES/MVS では、OPS5を利用してTMS (Truth Maintenance System) [Dolye 1978] の機能を実現している [Schor 1984] 。これはオペレーティングシステムの状態を示すワーキングメモリー内の情報の変化によって、従属関係を制御するルールを適用させるものである。この方式では、従属関係に基づくバックトラッキングを行うかわりに。ワーキングメモリー内の変化にともなって再推論を行っており、とくに推論機構の高速化が重要なポイントとなっている。

一方、TOAST では、エキスパートシステムの複数の構成要素を協調して動かすために、ブラックボードモデルを採用している（図7-2 ）。この実現にはOPS5を拡張したCOPS (COncurrent Procudtion System) を利用して実現している [Rychener 1984] 。このアーキテクチャは図7-3 に示すとおりであり、複数の推論機構が互いに相手のワーキングメモリを参照／変更できるようになっている。

7.3.2 OPS83 の考え方

OPS5は、比較的単純なシェルであり、上に述べたようにいろいろな改良を行ってから、実際のエキスパートシステム開発に適用する場合もある。それに対し、一般にハイブリイドシェルは、多様な推論方式を推論機構の機能として予め組み込んでいるた

め、個別の問題によっては、与えられた推論方式では不十分であったり、逆に、一部しか必要でないなどといったことがしばしばある。

OPS83 は、シェルのもつこのような欠点をへらすために、機能を増やすかわりに、推論機構を作るための基本関数を公開するというアプローチをとっているのが特徴である［Forgy 1984］。エキスパートシステム開発者は、この基本関数を組合せて、自分に必要となる推論機構を作るものである。従って、このタイプは、ツールというよりプログラミング言語としての性格が強く、個々の問題に応じて推論の制御を詳細に行いたい場合などに適している。実際、OPS-83は、PascalやModular-2 風の言語に、推論機構を構成するための基本関数などが追加された形になっている。このため、数値計算などは、OPS-83上で従来のプログラムのように記述すればよい。また、Fortran 、CやAssembler などの言語も容易にリンクすることもできる。

OPS-83で、推論機構を構成するための機能は次の７つである。

1)作業記憶（Working Memory）

2)作業記憶（WM）上のデータ（要素：element とよぶ）に対するタイプ宣言

3)作業記憶を書き換える関数、手続き

4)作業記憶上の要素についての情報を得る記述

5)ルールの記述

6)競合集合の管理

7)競合集合に対する関数

すなわち、OPS83 は知識技術者によるカスタマイズを前提としたシェル（プログラム言語）である。このタイプのシェルは現在のところ数は少ないが、今後、複雑なシステムに人工知能技法を組み込むための言語を提供するには有用と考えられる。たとえば電力設備などに関する大規模な人工知能システムの多くは既存プログラムとの結合融合で成り立っており、これを実現するには、OPS83 の考え方にしたがった高速かつ柔軟な推論システムが必要である。

7.3.3 KEE を用いたドメインモデルの作成

従来、大規模プラントは個別に設計されることが多く、設計情報をデータベースに蓄えても、それをさまざまな視点から利用することは難しかった。しかし、今後は、経済性の観点から、小規模な設備ばかりでなく、原子炉のような大規模設備に至るまで標準化の努力がなされることと考えられる。そのような場合、プラントに共通の情報と汎用のプラントモデルとを供えた深い知識をもつプロトタイプの知識ベースを開発しておけば、個々の業務のエキスパートシステム化に役立つものとなろう。これによって、各分野の専門家がみずから必要なエキスパートシステムを開発することが容易になることが期待される。

このような、プロトタイプ知識ベースの考え型にしたがって、現在EPRI（米国電力

研究所）では原子力プラント共通のエキスパートシステムの開発を行っている。これは、知識工学社が不足している状況で、米国の電気事業で人工知能の応用を活発化するには、ドメインエキスパートシステムがみずから使いこなせるツールを開発しなければならないという考えによる［Faught 1985 ］。

その結果として開発されたツールが、PLEXSYS (Plant EXpert SYStem) である。PLEXSYS は、次の機能をもつ原子力分野向けのエキスパートシステム開発ツール／プロトタイプ知識ベースシステムである（図7-4 ）。

(1) 高度な利用者インタフェース

電力会社の担当者が自ら知識ベースを開発できるようにプラントモデルを記述する手法を提供する。これはマルチーウインドウ機能、グラフィック機能を用いて作られており、プラントの構成部品をさまざまな視点から接続していくだけで知識ベースを作成することを可能とする。なお、これには、KEE のもつSimkitの機能を利用している。

(2) プラント構成表現機能

上記のインタフェースを用いることで、配管・構成図に含まれる構成部品と同等の複雑なシステムを表現でき、また、部品間の接続情報やバルブの状態などを表わせる。さらに、これらの構成は階層的になっているため、適切なレベルでサブシステムを表現できる。

(3) 推論機能

プラントのサブシステムの状況を各構成部品から推論できる機能（"readiness status" module）、ならびに、フォールトツリー分析機能が、通常のKEE の推論機構に追加された。

7.3.4 利用者インタフェースを重視したカスタマイズ

エキスパートシステムのエンドユーザインタフェースについては、他のソフトウェアシステムと同様、汎用のシェルでサポートできる範囲は少ない。特に、エンドユーザインタフェースとして評価の高い直接操作(Direct Manipulation) の概念を実現しようとすると、シェルを使う使わないにかかわらず膨大な開発作業が必要となる。

この方向で、努力をはらっているのがシュルンベルジュ社の石油探査データの解析システムとして有名なDipmeter Advisorである［Smith 1983］（図7-5 ）。Dipmeter Advisor は初期のバージョンはすべてInterlisp-D を使って記述されていたが、最近では、シュルンベルジュ社独自のシェルであるSTROBE、ならびに、その上のエディタ Impulse を使って記述されている。このシステムの特徴は、専門家である地質学者にとって使いやすいように非常に高度な利用者インタフェースを供えていることであり、システムの利用にあたって知識ベースのはたす役割はさほど大きくない［森 1987 ］。

現在のシェルは、一般に、日本語やグラフィックス、ポップアップメニューなどを供えたきめの細かいインタフェースを実現するには、従来型のプログラムに頼らなければならない面が多い。したがって、このように高度な利用者インタフェースが開発できるようにシェルをカスタマイズしていくのも重要な方向と考えられる。

7.4 おわりに

本章では、シェルのカスタマイズに関して、それが、知識技術者・各分野の専門家・エンドユーザのそれぞれに対してどのような意味をもつのかを考察し、ついで、シェルのカスタマイズの事例を紹介した。本章の議論は、はじめにも述べたように、各問題向きに特化したシェル、ならびに、問題向きの知識システムの開発方法論とも関連が深い。これらの研究はまだ開始されたばかりであり、発表された資料はあまり多くはない。その中で、次世代のシェルを研究していくためには、たとえば、[Chandrasekaran 1986]、[小林1986]などに見られる議論が参考となろう。

〔参考文献〕

[情報処理1987]　情報処理：“特集：エキスパート・システム．” Vol.28，No.2，(1987)．

[寺野1987a]　寺野隆雄：知識システムの構築環境．　計測自動制御学会講習会資料（エキスパートシステム：方法論と応用），pp.83-96，（1987年 3月）．

[寺野1987b]　寺野隆雄：エキスパートシステムシェルのユーザインタフェース．人工知能学会誌，Vol.2，No.2，pp.166-173，(1987年 6月)．

[Brownston1985]　Brownston，L.S.，Farrel，R.G.，Kant，E.，and Martin，N.：“Programming Expert Systems in OPS5”，Addison-Wesley，(1985)．

[Clayton1985]　Clayton，B.：“ART Programming Primer.” Inference Corp.，(1985)．

[Fikes1985]　Fikes，R.，and Kehler，T.P.：The Role of Frame-Based Representation in Reasoning，Comm．ACM，Vol.28，No.9，pp.904-920，(1985)．

[Popper1986]　Popper，J.，and Kahn，G.：Knowledge Craft：An Environment for Rapid Prototyping of Expert Systems．Proc．SME Conf．on A.I．for the Automative Industry，(1986)．

[Forgy1984]　Forgy，C.L.：“OPS83 Report.” Technical Report CMU-CS-84-133，Dept．Computer Science，Carnegie-Mellon University，(1984)．

[Doyle1979] :Doyle，J.:A Truth Maintenance System．Artificial Intelligence，Vol.24，No.3，pp.231-272，(1979)．

[Faught1985]　Faught，W.S.：“Functional Specifications for AI Software Tools for Electric Power Applications.” EPRI Report NP-4141，Research Project 2582-1，(August 1985)．

[Griesmer1984]　Griesmer，J.H．et al.：YES/MVS：A Continuous Real Time Expert System，Proc．AAAI-84，pp.130-136．

[Schor1984]　Schor，M.：“Using Declarative Knowledge Representation Techniques:Implementing Truth Maintenance in OPS5.” IBM Research Report RC 10455，(1984)．

[Talukdar1986]　Talukdar，S.N.，et al.：Toast：The Power System Operator's Assistant，IEEE Computer，Vol.19，No.7，pp.53-60，(1986)．

[Rychener1985]　Rychener，M.D．et al.：A Rule-Based Blackboard Kernel System：Some Principles in Design，IEEE Proc．Workshop on Principles of Knowledge-Based Systems，pp.59-64，(1984)．

[森1987]　森俊二：エキスパートシステムのユーザインタフェース，人工知能学会誌、Vol.2，No.2，pp.174-181，1987．6．

[Smith1983] Smith, R.G., and Baker, J.D.: The Dipmeter Advisor System: A Case Study in Commercial Expert System Development. Proc. 8th IJCAI, pp.122-129, (1983).

[小林1986] 小林重信：知識工学. 昭晃堂, (1986).

[Chanderasekaran1986] Chandrasekaran, B. : Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. IEEE Expert, Vol.1, No.3, pp.23-30, (Fall 1986).

8．知識システム構築技法

8.1　はじめに

エキスパートシステムが、近年多く開発され始められてくるに従って、エキスパートシステム構築環境の充実が求められてきている。現在、エキスパート構築ツールとして提供されているものの大部分は、エキスパートシェルであり、システム開発における上流工程をサポートするものは少ない。エキスパートシステムと言えども、一つのシステムであることには間違いはない。従って、従来のシステム工学的アプリケーションが全く利用できないわけではないであろう。しかし、そこにエキスパートシステム独自の特殊性があるとすると、それはいったい何なのであろうか？

従来のシステムは定型業務がその対象であった。ＳＥ（システム・エンジニア）は、現在行なわれている業務を分析し、何がルーチンワークで、何が例外処理なのかを見極め、例外処理を極力ルーチンワーク内で行なえるように計算機化のルーチンを作り、どうしても残る例外処理をシステム運用でカバーというシステム全体の枠組みを作るのが仕事であり、それをいかに矛盾なく作り上げるかがＳＥの能力であった。しかし、エキスパートシステムが対象とする領域は、より人間の知的活動に近い、いわゆる非定型業務（診断、設計、計画etc.）である。ここでの扱う処理は例外処理の塊で、業務の流れ、思考の流れを、可視的に表現しにくく（従来は事務分析図、フローチャート等で可視化した）、システム全体が把握しにくいことが、従来型システムとの大きな相違点であろう。このような領域に対し、ＳＥの能力にプラスして、何らかの能力と何らかの開発環境が必要となってくる。

ここでは、システムの要求定義からシステム構築に至るまでの工程に必要な機能、考え方をまとめてみた。ただし、この過程において非常に重要で、中心的役割を果す知識獲得においては、それだけで大きな広がりを持つ分野であるため、別に譲ることにする。(KAS.WG)

8.2　知識システム構築手順

エキスパートシステムといえども１つのシステムであり、一般のシステム構築手法が生かされるべきであるが、さらに、エキスパートシステムという特殊性を反映した手順をふむ必要がある。

一般のシステム構築手順は、フェーズド・アプローチと呼ばれ、システム計画、システム設計、システム制作といったフェーズに分け、開発が進められる。フェーズ分けすることにより、システム開発方針の妥当性が検討され軌道修正が可能となる。各フェーズの中では、システム案の技術的実現可能性や性能を評価するためにプロトタイプ的な

試作が行われる。

このようなシステム構築手順をエキスパートシステムの開発に適用し、そのプロトタイピング的側面を強調すると、図8-1 に示す如く、３フェーズに分けた標準的エキスパートシステム開発手順が設定できる。

第１フェーズのFeasibility model の作成フェーズでは、ばく然としたユーザの要求から、システムの果すべき目標を明確化し、技術的な実現可能性を検討する。ユーザの要求は不明確で矛盾を含んでおり、断片的なものであるのが通常である。また、エキスパートシステムに過度の期待を持っている場合が多く、ユーザに妥当な実現イメージをもたせる必要がある。従来手法によるシステムではなく、エキスパートシステム化することの意味も十分つめておかなければならない。また必要であれば試験的に、知識の抽出を行ない、計算機にインプリメントし、技術的可能性をチェックする。その結果を評価し、最初のユーザの要求を明確化すると共に、実現の無理な部分は変更をする。

第２フェーズのPrototype の作成フェーズでは、前フェーズでの結果をもとに、実際にエキスパートシステムとして機能するプログラムを作成する。知識の量、性能、マンマシン機能等は実用レベルに達していなくてもよい。技術的に核となる知識、推論機能がともかく実現されておればよい。このプロトタイプを評価し、ユーザの要求を一層明確化し、より妥当な仕様に変更する。

第３フェーズのReal-use modelの作成フェーズでは実用に耐えられるエキスパートシステムを作成する。知識の量は実際に使用しながら、増していくが、ともかく、エンドユーザが現場で使えるシステムとする必要がある。

各フェーズ内における作業ステップは次のように分けられる。

問題定義ステップでは、これから開発しようとするエキスパートシステムへのユーザニーズおよび問題点を明らかにし、システムの目的、エキスパート化すべき範囲を設定する。専門知識を出してもらう専門家を誰にするかも重要な決定である。

知識収集・整理ステップは、初期知識の獲得をするステップである。専門家自身がＫＥの役割を兼ねる場合を除き、ＫＥは予め専門分野の基礎的知識を修得しておく必要がある。

知識の入力・推論の決定ステップでは、問題のタイプに合った知識表現、推論法の選択をする。

知識の入力・編集ステップでは、収集整理された知識を、前ステップで決定された知識表現法に変換し、コンピュータに入力する。また、運用に入ってから、新たに追加す

べき、もしくは変更すべき知識が見つかった場合も入力する。

知識の検証ステップは、知識ベース中に蓄積された知識に誤りはないか、各知識は正しそうに見えても連結された場合、おかしな結論になっていないか、知識間に矛盾はないか、抜けはないか、等のデバッグをする。

運用・改善ステップは、プロトタイプであれ、実用システムであれ、当初は知識ベースは不充分なものであるから、実際に問題を与え、運用をしてみる。運用をしながら、次第に知識を改善していく方法をとる。

8.3 問題定義と知識収集・整理

(1) 問題定義

まず開発するエキスパートシステムの果すべき目的を明らかにする。目的は単一とは限らず、複数の目的からなるのが通常であり、これらを階層的に整理し、これから開発するエキスパートシステムが、どの範囲をねらうかを明確にすることが第一歩となる。

次に、これらの目的を達成するのに関連するニーズを列挙する。ニーズの中には機能、性能的要求、制約条件、解決すべき問題点、評価要素が混在している。これらを項目ごとに分類し、整理することにより、システムの理解ができる。

ニーズを抽出する方法として、問題に関連した文献の調査、ユーザを交えたブレーンストーミング、ユーザへのアンケート、あるいは専門家へのインタビュー等がある。ブレーンストーミング等では、例えば、縦軸に関与者、横軸に場所のように、５Ｗ１Ｈ（Why. What. Where. When. Who. How）から適当な２つを選び縦と横に並べた表を用い、各枡目にニーズを記入する。現状システムの改善の場合は、現状調査を行なう。それにより、現状の不満に対する改善ニーズ、現状の機能のうち真に必要なニーズは何かといった観点からニーズ抽出ができる。

(2) 知識の収集

知識ベースに蓄積すべき知識は、エキスパートシステムを実現する立場から見て必要な機能、制約等を表すものである。即ち、(1) 問題定義で列挙したニーズがユーザの立場からの言葉であるのに対し、ここでは専門家・開発者の立場からの言葉で知識を収集する。

収集の方法としては、対象分野の文献調査、専門家へのインタビュー、専門家の行動記録、専門家の発話記録等を通し、キーワード、キーセンテンスを抽出し列挙する。これらが知識の核となる。この時、５Ｗ１Ｈが知識抽出のヒントになる。

(3) 知識の整理

収集された知識の核の間の関係を分析し、エキスパートシステムの目的達成に必要な知識のシステム全体構造を求めることが知識整理である。分析の手法としてはシステム構造化技法を用いる。代表的なシステム構造化技法には自己相関分析と相互相関分析がある。

(i) 自己相関分析による構造化

これは、知識の核の集合をシステムの表現と考え、その知識の核間にある関係を定義するものである。

x ： 知識の核

s ： システムsを表現する述語

として、

$s = \{x \mid s(x)\}$

であるとき、x，y間に二項関係Rを考え推移律を仮定するものである。即ち、

$R(x, y), R(y, z) \rightarrow R(x, z)$

xを構造化するには、この推移律を使った同値関係（グラフ論的には、強連結成分）でグループ化を行なう。図8-3 に例を示す。

しかしながら当然のこととして実際的に関係Rの定義が明確でなく、推移律が一般的には成立しない場合が多い。そのため、Rを多値化して利用する場合が多い。またRは一意的でなく、いくつもの関係が考えられるため、構造化の一つの例として考えた方が良い。

二項関係Rの例としては、因果関係、制約関係、推奨関係等、種々に考えられるが、エキスパートシステムのタイプ、分野により選択する必要がある。

(ii)相互相関関係による構造化

これは、一つのシステムsを二つの観点X，Yから記述し、それらの記述間の関係の全体をシステムsを表す概念と考える考え方である。すなわち、一つのシステムを記述する方法を二つ考え、それらを結ぶ共通概念がシステムを表現すると考える。

$s = \{R \mid R(x, y), x \in X, y \in Y\}$

グラフ論的には二部グラフに対応し、システムは

$$(X1, Y1) : \forall R \exists x1 \in X \quad \exists y1 \in Y$$
$$\exists x \in X1 \quad \exists y \in Y1$$
$$R(x, y1) \vee R(x1, y)$$

で表現されるとする。(ϕ, Y)，(X, ϕ) もシステムの表現であり、

（X｜，Y｜）も一つの表現である（図8-4）。この手法の特徴は、システム自体というよりはシステムの表現の構造化に主眼を置いてあり、上記のさまざまな表現に制約を課し構造を抽出しようとするものである。特にX，Yの大きさ｜X｜，｜Y｜に制約を置き｜X｜＋｜Y｜が最小となるように構造化を行なうと、二部グラフの既約表現になり、X，Yがグループ化される。この方法によれば、Xとしてシステムの目的、Yとしてそれを達成するための機能とすることにより、直接システムの機能分解が可能となる。相互相関関係を使ったほかの構造化手法としては、一方の観点の言葉を他の観点の言葉と関係を使って多次元空間内の平面に対応づけ位相空間として解析する手法も研究されている。

8.4 知識表現・推論法の決定

エキスパートシステムの概要がある程度明確になってくると、次は知識をどのような構造で体系化していったら良いか、また、目的とするoutputを得るためには、どのような推論メカニズムで行ったら良いかの検討を行う段階になる。ここで、次の前提で、大きくそのアプローチが異なってくる。その前提とは、エキスパートシステムを構築するためのシェルを決めているか否かである。構築するためのシェルが事前に決まっている場合は、そのシェルが持っている特殊性を充分に考慮に入れ、そこで提供している知識表現、推論エンジンの制約に沿って構築することになり、その制約が逆に設計時の思考の発散を防いでくれるため、ある意味では、構築しやすくなることも事実である。しかし、システム工学的見地からすると、やはり対象とするアプリケーションにおいては知識の表現、構造はどうあるべきか、推論メカニズムは、どのような機能を持たせ、どう構造化すべきかというトップダウン的な思考過程の結果、どのようなシェルを使うべきかという結論に達するのが常道であると思われる。

しかし、現在開発されているエキスパートシステムの多くは、前者のアプローチをとっているものと思われる。その原因は、トップダウン的思考を行う上での、その判断基準、テンプレートがないことに起因していると考えられる。つまり、ある要求仕様が与えられた時に、全くゼロから「どうあるべきか？」を考えるのではなく、「このような要求仕様はブレークダウンすると、××問題と△△問題との複合問題である①と考えられる。従って、××問題においては、このような推論形式があり②…」という何らかの指針となるべきパターンが存在することが重要である。下線①は、問題の分類、構造化であり、下線②は推論のパターン化である。このように問題を定式化し、パターン化していく研究の成果はここ１～２年でようやく顕在化されはじめている。そこで、このような考え方の代表的なものを紹介する。

8.4.1 Generic Tasks

Generic Tasks はOSU(Ohio State University)のB. Chandrasekaran 教授らのグル

ープによって提唱された考え方である。Chandrasekaran教授らの問題意識は次の通りである。現在提供されているツールは、汎用的なものであり、問題向ツールになっていない。従って、エキスパートシステムを構築する場合、問題の構造を反映させたツールの利用というよりも、ツールに合わせた問題構造にしてエキスパートシステムを構築しているというのが現状である。問題を単に診断型とか設計型だとか、非常にグローバルな分類をするのではなく，それをもっとブレークダウンし、よりプリミティブな問題に分解し、そのプリミティブな問題に合った問題解決器を与えるべきである。

アプリケーションの問題は、このプリミティブな問題の複合問題と捉えることができる。このような考え方に基づいて、次の６つのGeneric Tasks を定義している。

- Hierarchical classification （階層的分類）
- Hypothesis matching, of assessment（仮説のマッチング／評価）
- Knowledge-directed information passing（知識指向型情報伝達）
- Abductive assembly（発想的組立て）
- Hierarchical design by plan selection and refinement
  （プランの選択と洗練による階層的設計）
- Statea bstraction （状態の抽象化）

(1) Hierarchical Classification

分類に関しては、階層の各ノードが仮説に対応する。そのノード１つ１つはspecialistと呼ばれ、そのノードの下位のノードは、それをよりブレークダウンするspecialistに相当する。

図8-5 においては、internist （内科医）が病気を発見し、専門医（肝臓、心臓）に情報を渡す。各専門医はその情報を見て、自分に該当するかを判断できる機能を有している（establish ／reject）。もしestablish されると、それをよりブレークダウンするために下位のノードに問いかける（refine）。このような問題解決方式をestablish-refine type として特徴づけることができる。

(2) Hypothesis matching, or assessment

前期Hierarchical Classification で分類され階層化された各ノード（spesialist）は、その仮説がマッチするかどうかを評価するためのメカニズムを与えるものである。また、このメカニズムは独立した、それ自身でgeneric な性質を持っている。（計画的問題にも用いることができる）

(3) Knowledge-directed information passing

ある問い合わせに関して、それに対する答えが直接的には存在しないことがある。この時、現在ある事実から、その答えを推論してやる必要がある。

「もし麻酔を受けた経歴があるなら、肺炎を考えろ」

↓　　質問

「麻酔を受けた経歴があるか？」 → ・麻酔履歴に関する情報（情報ナシ）

回答

yes ← ・大きな手術の履歴情報（yes ）

- default
- demon
- inheritance

(4) Abductive assembly

ある現象を説明するのに、最も妥当な仮説を選び出し、それらを合成して複合仮説を生成する。

< deduction >

$\forall x \ (P(x) \rightarrow Q(x))$

↓

$P(a) \rightarrow Q(a)$

< induction >

$P(a) \rightarrow Q(a)$

↓

$\forall x \ (P(x) \rightarrow Q(x))$

< abduction >

$\forall x \ (P(x) \rightarrow Q(x))$

↓

$Q(a) \rightarrow P(a)$

(5) Hierarchical design by plan selection and refinement

設計問題は3つのクラスに区別できる。

Class 1 ： 設計対象物の構成要素すら分らないレベル

Class 2 ： 設計対象物の構成要素は分っているが、まだ手本となるものがないレベル

Class 3 ： 設計対象物の構成要素は分かっており、設計の各段階において利用できる手本があるレベル

いわゆるClass 3 を"routine plans" と言う。

各構成要素は、それぞれのspecialistになっており、その下部には、その構成要素をブレークダウンするsub-specialistがいる。（図8-6 ）

各specialistは次のような構造になっている。

specialistは、プランを選び出し（Plan Selector ）、そのプランを実行させる。その実行されたプランが更に他の詳細な部分を設計するために必要なspecialistを呼び出すという型で再帰的に設計が完了するまで続けられる。

(6) State abstraction

「もし、このような状態になったら何が起るか？」（What will happen if ～）という予測問題である。このタスクの必要な推論は定性的なシミュレーションである部分の変化が、全体にどのような変化を及ぼすかを、ある状態変化が起ると予測される振りを仮定し、更に、その過程のように動作したとすると、またどうなるかを再帰的に情報を伝播させながら推論していく。

8.4.2 その他の提案

1) Heurististic Classification

スタンフォード大学のClancey によって提唱されているものである。Clancey の問題意識も、Chandrasekaranとほぼ同様で、多くのエキスパートシステムが世の中に、たくさんあるけれども、それらのシステムが、どのようなメカニズムで動いているかをきちんと表現することは難しい。それは、そのシステムの作成者すらもきちんと説明できない。しかし、これらのシステムをよく分析してみると、驚くほどの共通点がある。このようにそれぞれの推論メカニズムを分析し、定式化できると、新たにある問題を持ってエキスパートシステムを構築しようとするものにとって、非常に有用なものになるという認識である。

MYCIN, SACON, GRUNDY, SOPHIEなどの推論形態は非常に良く似ている。

<Heuristic Msatch>

エキスパートシステムのタイプは大きく２つに分けることができる。

- 解を選ぶもの …… selection
- 解を作り出すもの …… construction

Heuristic Classficitation は、selection 問題の有力な手法である。Building Expert System（F. Hayes. Roth等の著書）によると、エキスパートシステムは、INTERRETATION （解釈）、PREDICTION（予測）、DIAGNOSIS （診断）、PLANNING（計画立案）、DESIGN（設計）、MONITORING（モニタリング）、DEBUGGING （デバッギング）、REPAIR（修理）、INSTRUCTION （教育）、CONTROL（制御）というように分類されているが、それぞれの定義が明確ではない。これらをもう少し体系的にエキスパートシステムのGeneric な操作を分析すると図8-7 のようになる。

このようにoperatorを定義することにより、推論メカニズムは定式化することができる。

例）　MICIN ＝Monitor （患者の状態）
　　＋DIAGNOSE（病気の範疇）
　　＋IDENTIFY（バクテリア）
　　＋MODIFY （体、組織）
　　BRUNDY＝IDENTIFY（人間のタイプ）
　　＋PLAN （読書プラン）
　　SACON ＝IDENTIFY（構造のタイプ）
　　＋PREDICT （数値モデル）
　　＋IDENTIFY（詳細な予測のための分析のクラス）
　　SOPHIE＝MONITOR （回路の状態）
　　＋DIAGNOSE（モジュール／部品の欠陥）

2)　Ontological Analysis(SUREOSPOONS)

これはTektronix のAlexander らが提唱しているものである。従来の知識は、シェルに合った表現で収集されていた。これは、今のツールが非常に低いレベルでサポートされているため、直接的な知識内容の表現になっている。Ontological Analysisは、知識要素が持っている基本的構造を明確に分析しようとするものである。つまり、モデルにおける概念を領域（ドメイン）方程式として明確にしていこうというものである。これをサポートするツールとしてSUPE-SPOONS がある。

9.5　まとめ

知識システムの開発過程における上流工程は、未だ明確な手法が確立されていない（従来型システムにおいても未確立であるが）。システム計画－システム設計－システム製作といった開発フェーズにおいて、システム設計フェーズでGeneric Tasks のような概念が、提唱されはじめたことは非常に注目すべきことであり、一部ツール化されて

きていることは歓迎すべき傾向である。またシェルを提供しているメーカも、そのシェルを使うという前提で、その前工程をサポートするツール、方法論も提供しようとする動きが活発化してきているようである。

今後とも、このような研究開発が一層盛んに行なわれていくものと見られるが、よりプリミティブなレベルでの問題表現、知識構造表現が行なえる共通語の確立が急務である。

〔参考文献〕

[新井　87]　新井政彦、本位田貞一、"診断型エキスパート・システム"、
情報処理　vol.28, no.2, pp.177-186, (1987).

[河野　85]　河野毅、田村信介、"システム計画技法"、
電気学会雑誌　vol.105, no.6, pp.54-548, 1985.

[Arai 81]　M.Arai, et.al : A Method for structural Modeling of Complex Systems, Proc. IEEE Int'l. Conf. on SML (1981).

[Chandrasekaran 86]　B.Chandrasekaran, "Generic Tasks in Knowledge Based Reasoning: High-Level Building Blocks for Expert System Design", IEEE EXPERT, vol.1, No.3, pp.23-30, (Fall 1986).

[Brown 86]　D.C.Brown and B.Chandrasekaran, "Knowledge and Control for a mechanical Design Expert System", IEEE COMPUTER, vol.19, no.7, pp.92-100, (July 1986).

[市吉　86]　市吉伸行、"エキスパート・システム構築ブロックとしてのGeneric Task"
第５回ＫＳＳワーキンググループ資料　KSS5-1, (1986).

[Clancy 85]　William J.Clancy, "Heuristic Classification", Arfiticial Intelligence, 27, pp.289-350, (1985).

[福永　86]　福永光一、"W.ClancyのHeuristic Classification",
第４回ＫＳＳワーキンググループ資料　KSS4-1, (1986).

[Alexander 86]　James H.Alexander, "Knowledge Level Engineering :Ontological Analysis", AAAI'86, pp.963-968.

9．知識ベース管理・最適化について
－設計問題にとっての知識ベース管理・最適化－

9.1　知識ベースの２つの側面

管理・最適化の対象となる「知識ベース」には、次の２つの側面がある。

(1) ノウハウを表現するルール等の知識
- 一通りの実行中でのほ普遍
- 獲得、学習等の対象

(2) 実行中に生成されたデータ
- 無矛盾性維持
- ＴＭＳ，　ＡＴＭＳの対象

具体的には、(2) を設計について見てみると・・・

設計という作業は、試行錯誤のプロセスを伴う。これは、ひとつには、設計者が個々の決定（design decision ）の及ぼす影響を正確に予測することができないことによる。繰り返しは、ひとつの設計過程（design stage）内で起る場合もあれば、何段階か前の設計過程に戻ってやり直すこともある。いずれの場合も、やり直しを管理するメカニズムが必要であろう。具体的には、適当なところからのやり直しをサポートすること、決定に理由を示す情報（justification ）を保存しておくこと、がある。後者は、ユーザが前に下した決定について忘れてしまった時にも有効であるし、システムがユーザに説明をする時にも役立つ。

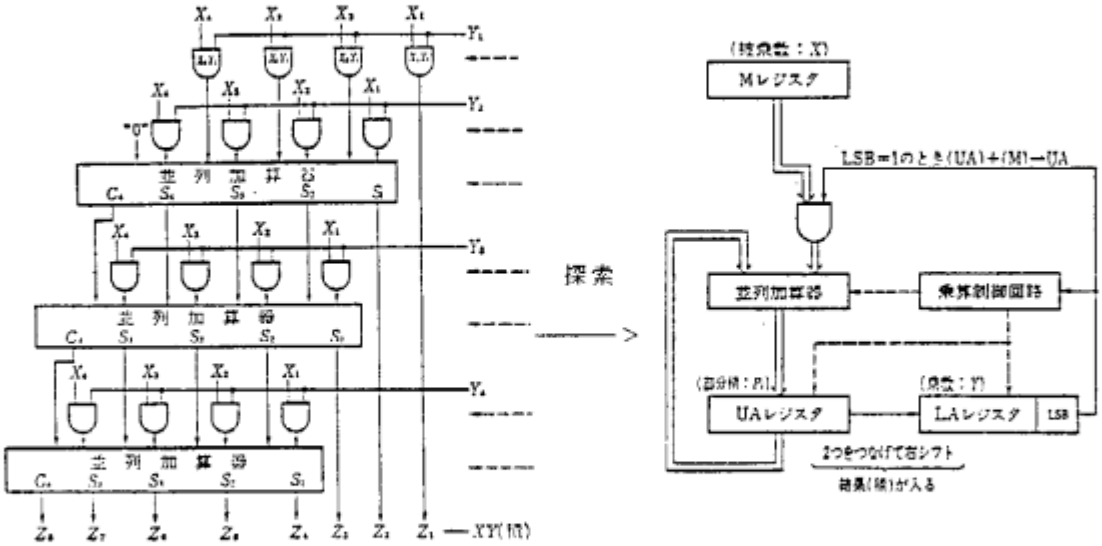
9.2　仮説推論

上で述べたような、試行錯誤を伴いやり直しが頻繁に起こる状況では、個々の決定を仮説と見ることもできる。知識ベースの管理・最適化と仮説推論の接点はここにひとつある。また、セル・ライブラリを利用して設計を行う場合、個々のセルを直接仮説と考えてしまうこともできる。これが第二の接点である。

(1) 個々の決定を仮説と見た、仮説推論としての設計

以下にひとつの例を示す。

最初の選択　　次の選択



justification <Z<-X*Y,面積に余裕>　　justification <Z<-X*Y,時間に余裕>

↓　↑

引き続く　↓　↑　やり直し

設計過程　↓　↑

面積オーバー！

(2) 個々のセルを仮説と見た、仮説推論としての設計

この場合には、次々と仮説を選択していく探索問題の様相が濃くなる。

昭和61年度　知識システムシェル
ＫＳＳ　ＷＧ報告書

図　面　集

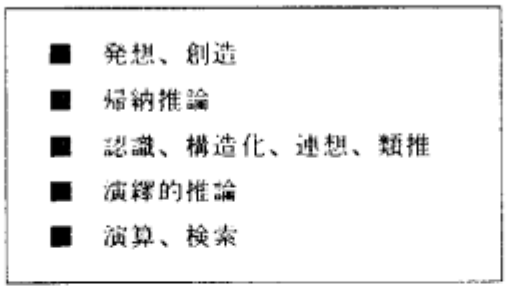


図 1-1 知能のレベル

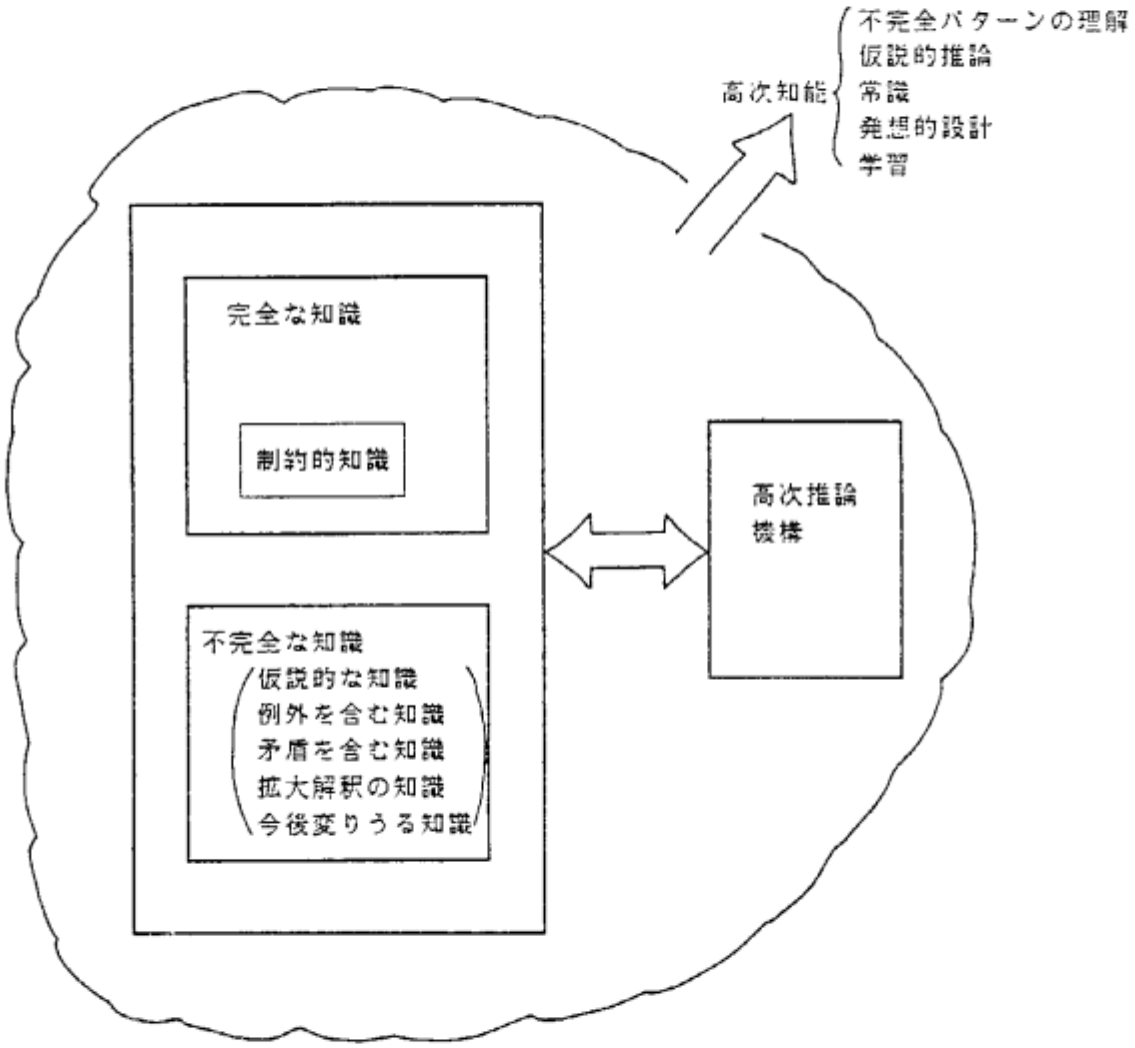


図 1-2 完全な知識に加え不完全な知識も含む知識ベースを
操作することにより高次知能を実現する概念図

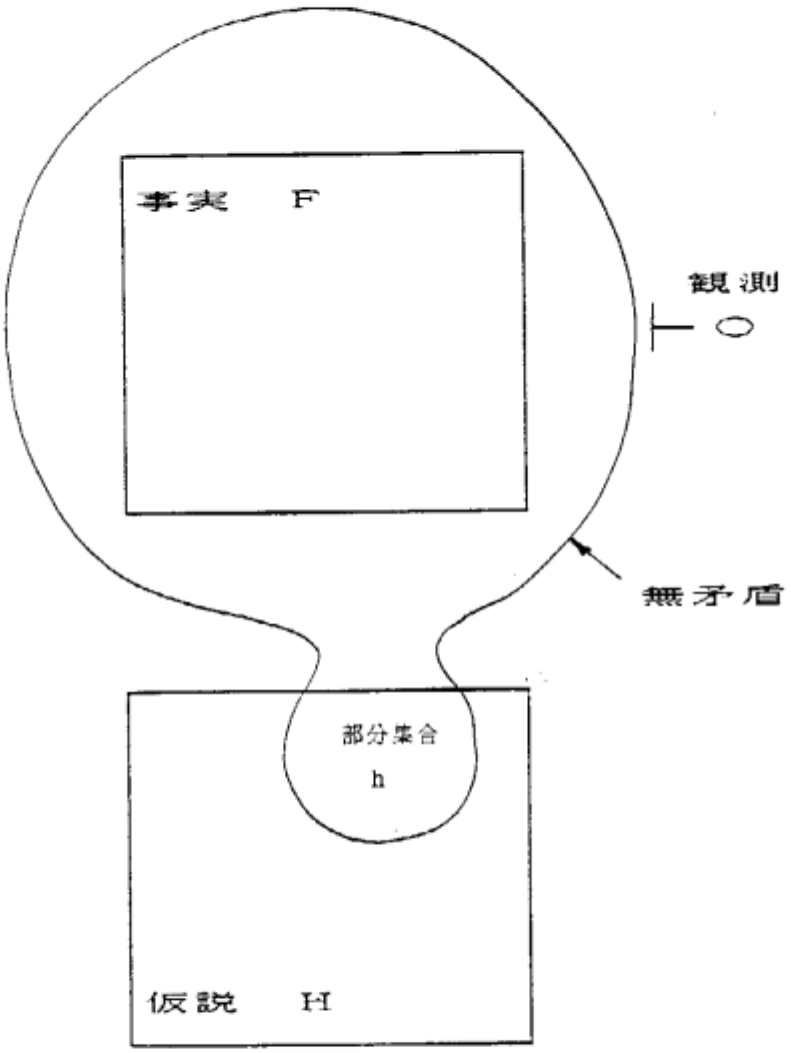


図 1-3　仮説推論システム

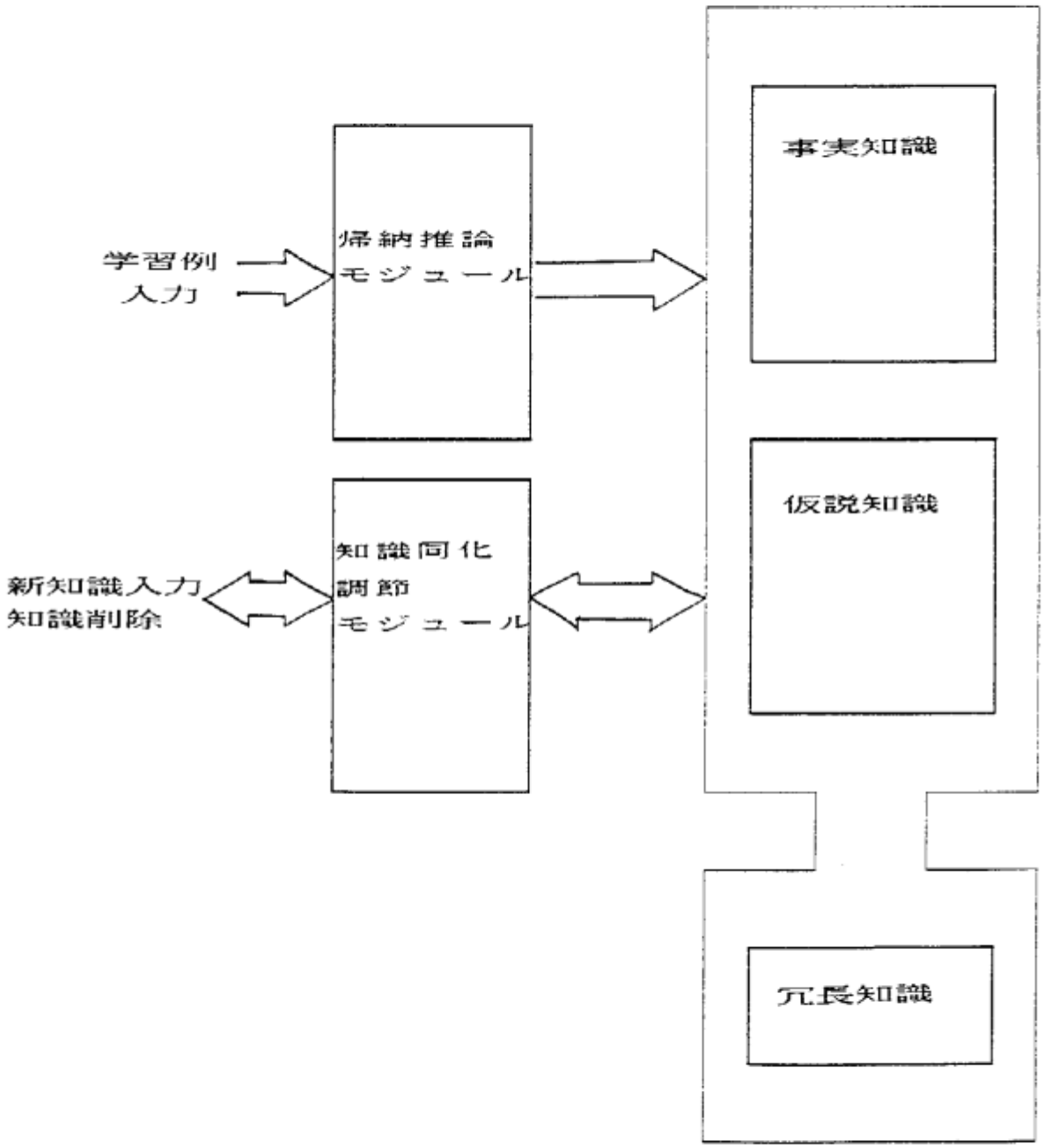


図 I-4　仮説を含む知識ベースと知識獲得モジュール

ＴＥＷＰメータ

Role:

冷却水の温度を測定

冷却水

Role:

Ｘから熱（Ｑｉｎ）を吸収

Ｙに熱（Ｑｏｕｔ）を放出

Structure:

シリンダとＸで接続

ラジエータとＹで接続

Environments

力Ｆを受けて循環している。

Attribute:

液体：l

流量：f

温度：T

比熱：c

図4.1-1　Device Worldの例

熱力学の公式1

適用条件

流体：l

吸収する熱量：Q

流量：f

比熱：c

温度変化：T1 →T2

物理式

$$T2 = \frac{Q}{c\,f} + T1$$

図4.1-2　Physical Worldの例

（メカの状態）計器Xが測定する物理パラメータ
Yの値が基準外である。
（徴候）計器Xの示す値が基準外である。
（メカの状態）部品Xの役割（role）が否定されている
（故障仮説）部品Xが故障している
（メカの状態）経路Xの断面積が基準値以下
（故障仮説）
1．経路Xに物が詰まっている
2．経路Xが細くなっている
3．外力によって経路Xが細くなっている
4．経路Xの構成物質が変形している

図4.1-3　Interpretation Worldの例

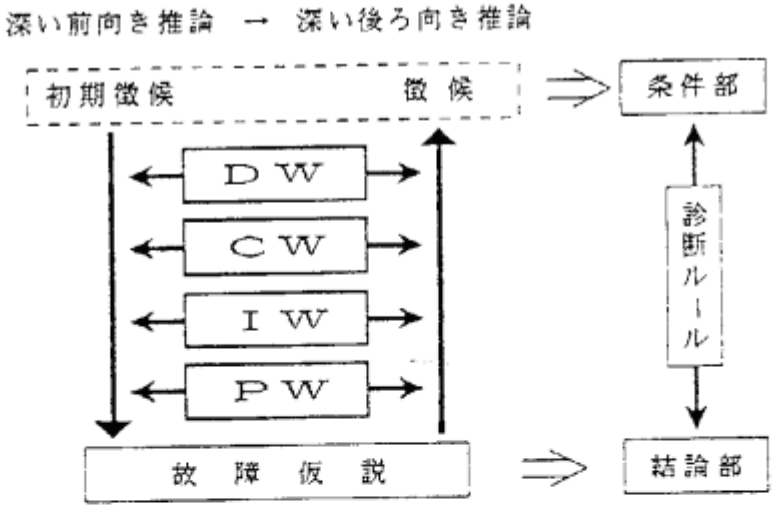


図4.1-4 知識コンパイラの概観

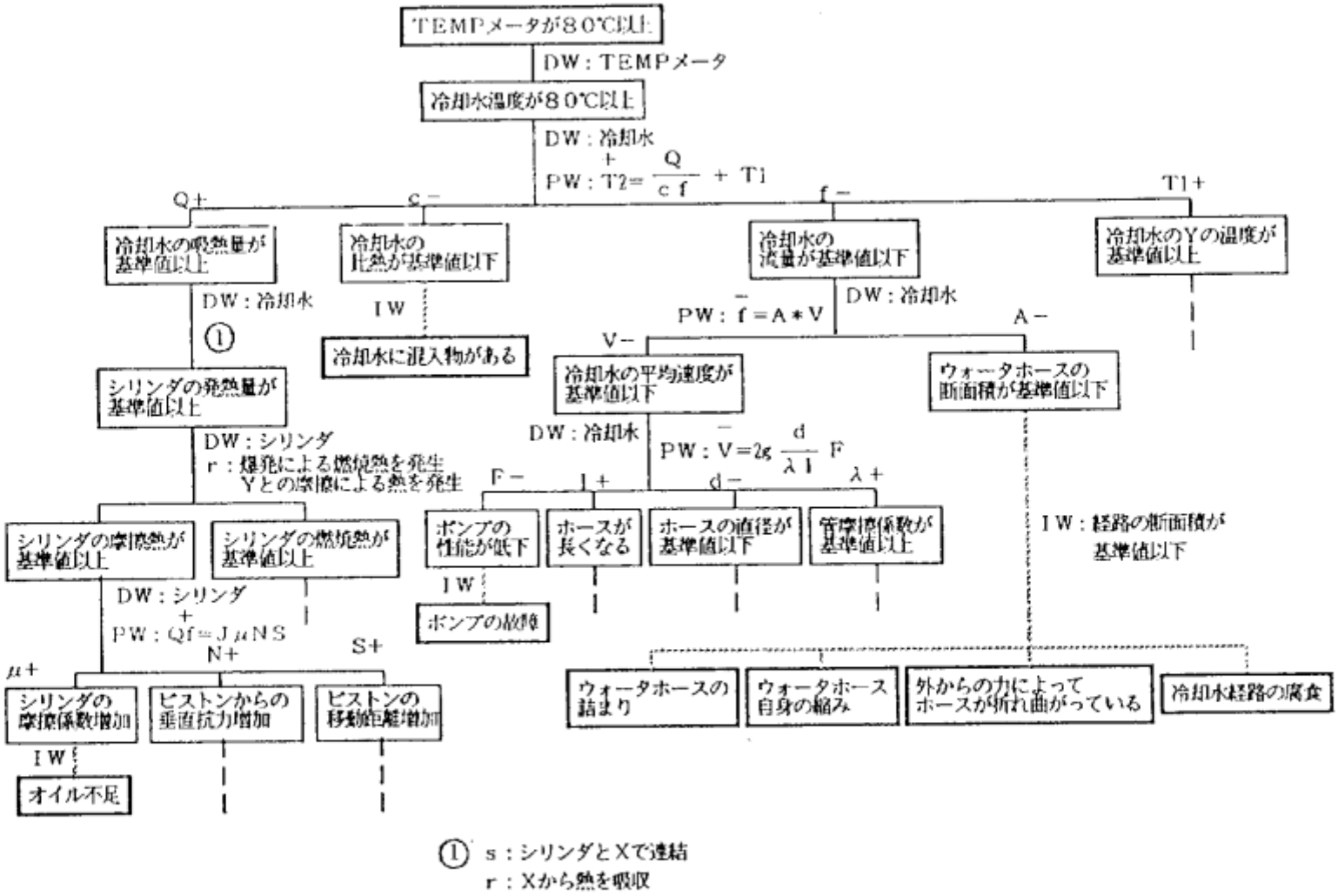


図4.1-5 深い前向き推論による故障仮説の生成

図4.1-6 深い前向き推論により生成された不完全なルール

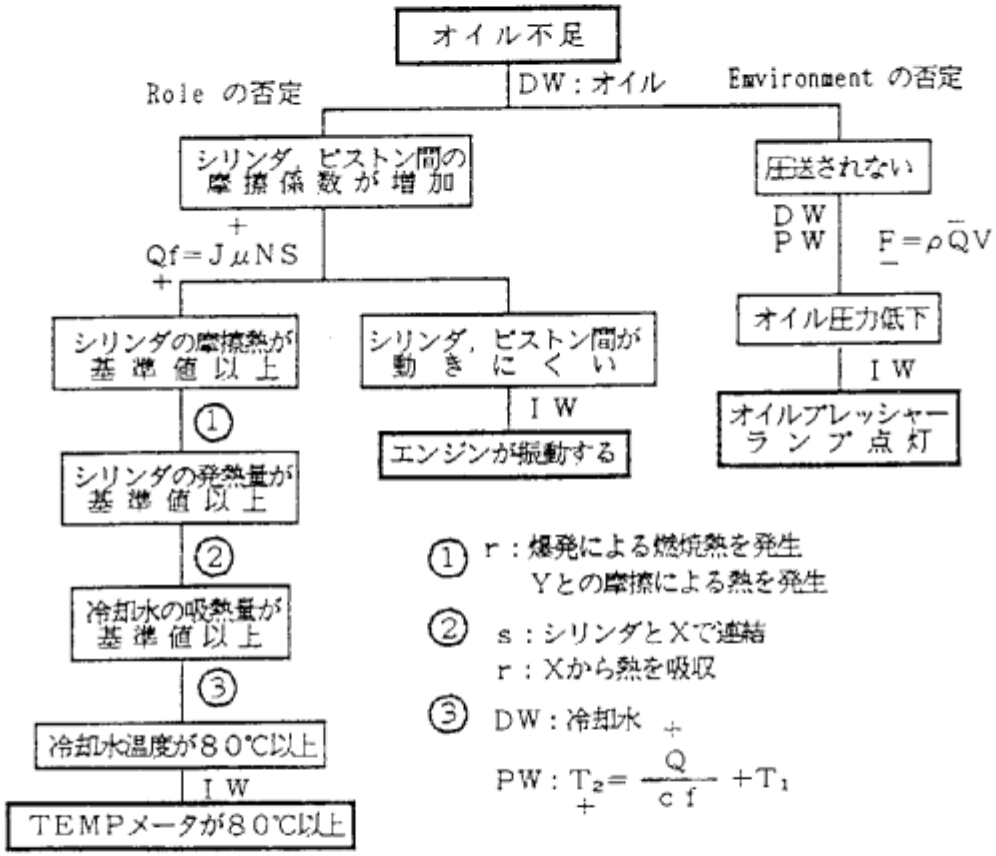


図4.1-7 深い後向き推論による徴候の生成

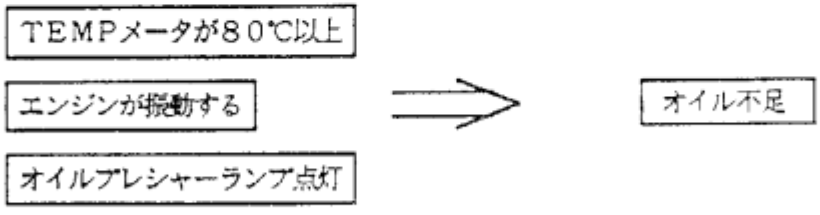


図4.1-8 深い後向き推論により生成された完全なルール

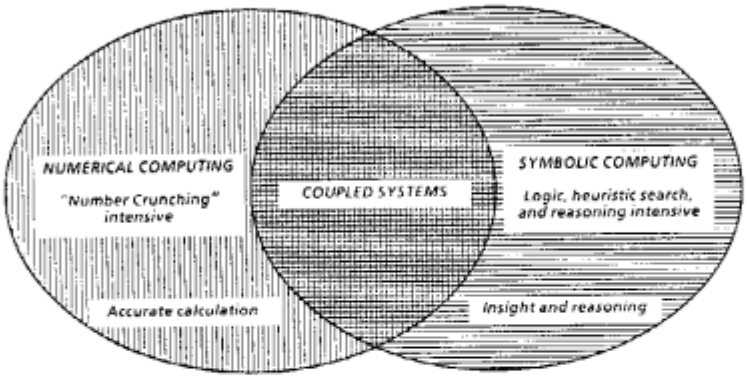


図4.2-1 数値演算と記号演算の関連

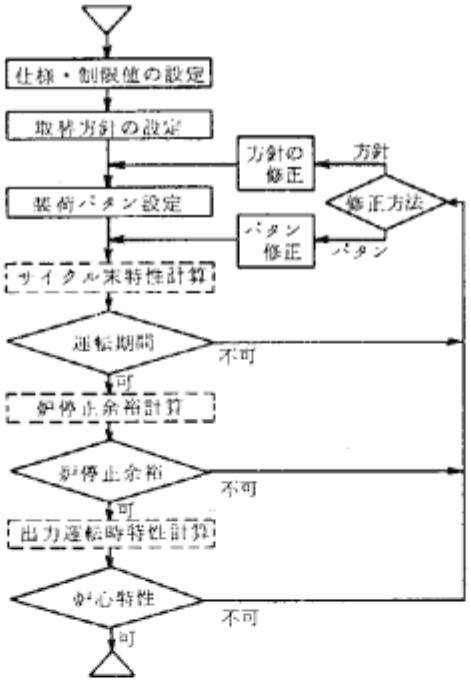


図4.2-2　燃料取替計画作成の手順

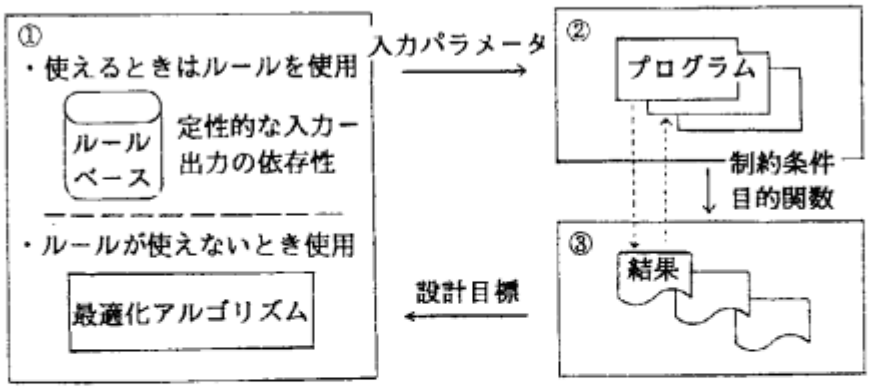


図4.2-3　設計支援システムの基本構成

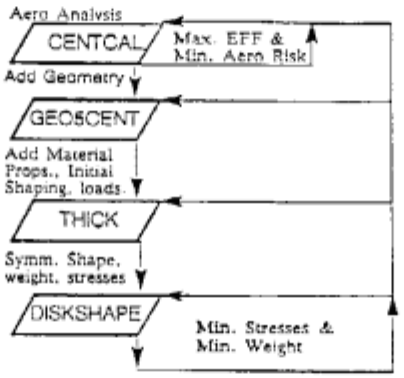


図4.2-4　コンプレッサ設計の手順

表5-1　知識表現モデルから見た並列処理〔小池　86〕

知識表現モデル	アルゴリズム・並列性	並列性の開拓/アーキテクチャ	システム例
意味ネットワーク	－同時ノード探索 －データフロー処理 －連想	－ノードとＰＥを{1：1 / n：1}に対応	－コネクションマシン －SNAP －CAP －IX
		－プロセッサアレイ，キューブ（SIMD）	
論理型表現	－AND並列 －OR並列 －ストリーム並列	－リダクション －データフロー	－PIM －PLM
		－結合ネットワーク	
プロダクションシステム	－Reteアルゴリズム －並列コンディションマッチング	－Rete nodeを分散 －プロダクションを分散 －WMを分散	－DADO －NON－VON －PSM －RISCF
		－トリー	
黒板モデル	－黒板の階層化 －黒板の分割 －複数知識源の分散化		－（スタンフォード大）
		－（プロセッサアレイ）	

表5-2　ＡＩ向きマシンの例〔小池　86〕

システム名	形　　態	モ　デ　ル	開発機関
(1) コネクションマシン	プロセッサメシュ+キューブ	CMLisp, 意味ネット	MIT/TM
(2) ボルツマンマシン	?	熱平衡過程, 意味ネット	CMU
(3) ＩＸ	階層型	意味ネット	ETL
(4) SNAP	プロセッサアレイ	意味ネット	USC
(5) DADO	トリー型	並列プロダクション	Columbia
(6) Non　Von	トリー+接続ネット+メシュ	並列プロダクション, DB	Columbia
(7) CAP	プロセッサアレイ	(意味ネット)	ITT
(8) PLM	専用プロセッサ+マトリクス	Prolog	UCB
(9) Transputer	プロセッサアレイ	Occam	INMOS
(10) ＦＡＩＭ−Ｌ	専用プロセッサ六角アレイ	メッセージベース	Schiumberger
(11) Fuzzy Chip	?	Fuzzy 推論	Bell
(12) BBN−バタフライ	汎用プロセッサ+接続ネット	Lisp	BBN
(13) IBM−RP3	RISCプロセッサ+接続ネット		IBM
(14) Rediflow	プロセッサメシュ	リダクション+データフロー	ユタ大
(15) ALICE	トランスピュータ+接続ネット	リダクション	インペリアルカレジ大
(16) ＰＩＭ−Ｒ	専用プロセッサ+接続ネット	リダクション	ICOT
(17) PARK	汎用プロセッサ+階層バス	AND/OR並列	神戸大
(18) KRP	専用プロセッサ+トリ型ネット	リダクション	京　大
(19) ＰＩＥ	専用プロセッサ+接続ネット	ゴール書き換えモデル	東　大
(20) Symbolics 3600	専用プロセッサ	Lisp	シンボリクス社
(21) PSI	専用プロセッサ	Prolog	ICOT
(22) CHI	専用プロセッサ	論理型	日　電
(23) 1100SIP	専用プロセッサ	Infer Lisp・D/ Smalltalk	ゼロックス社

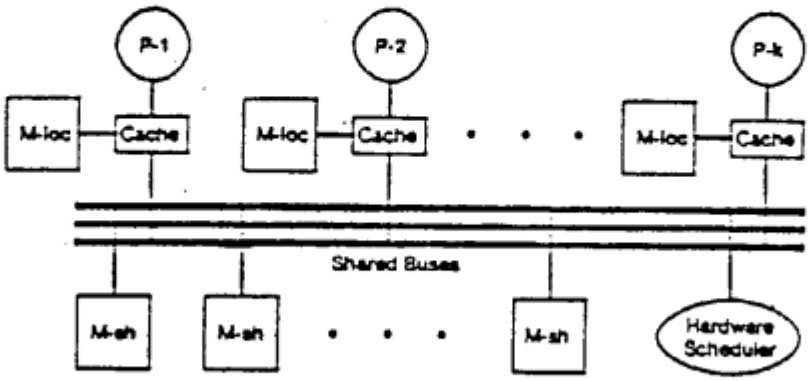


図5-1　PSM のアーキテクチャ

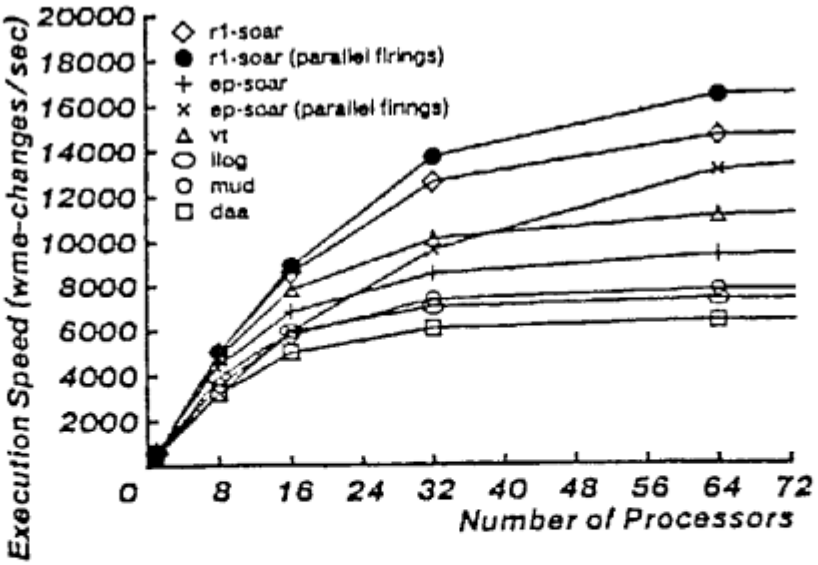


図5-2　PSM による性能向上

```
(make-parameter 'covering '(feathers hair))
(make-parameter 'animal-class
                '(mammal bird reptile))
(make-parameter 'eating-class
                '(ungulate carnivore))
(make-parameter 'ped-type '(claws hoofs))

(make-rule 'animal-rule-1
  '(if (covering hair))
  '(then (animal-class mammal .95)))

(make-rule 'animal-rule-2
  '(if (animal-class mammal) (ped-type hoofs))
  '(then (eating-class ungulate .9)))
```

図5-3　animalのルール及びパラメータ例 [Bielloch 86]

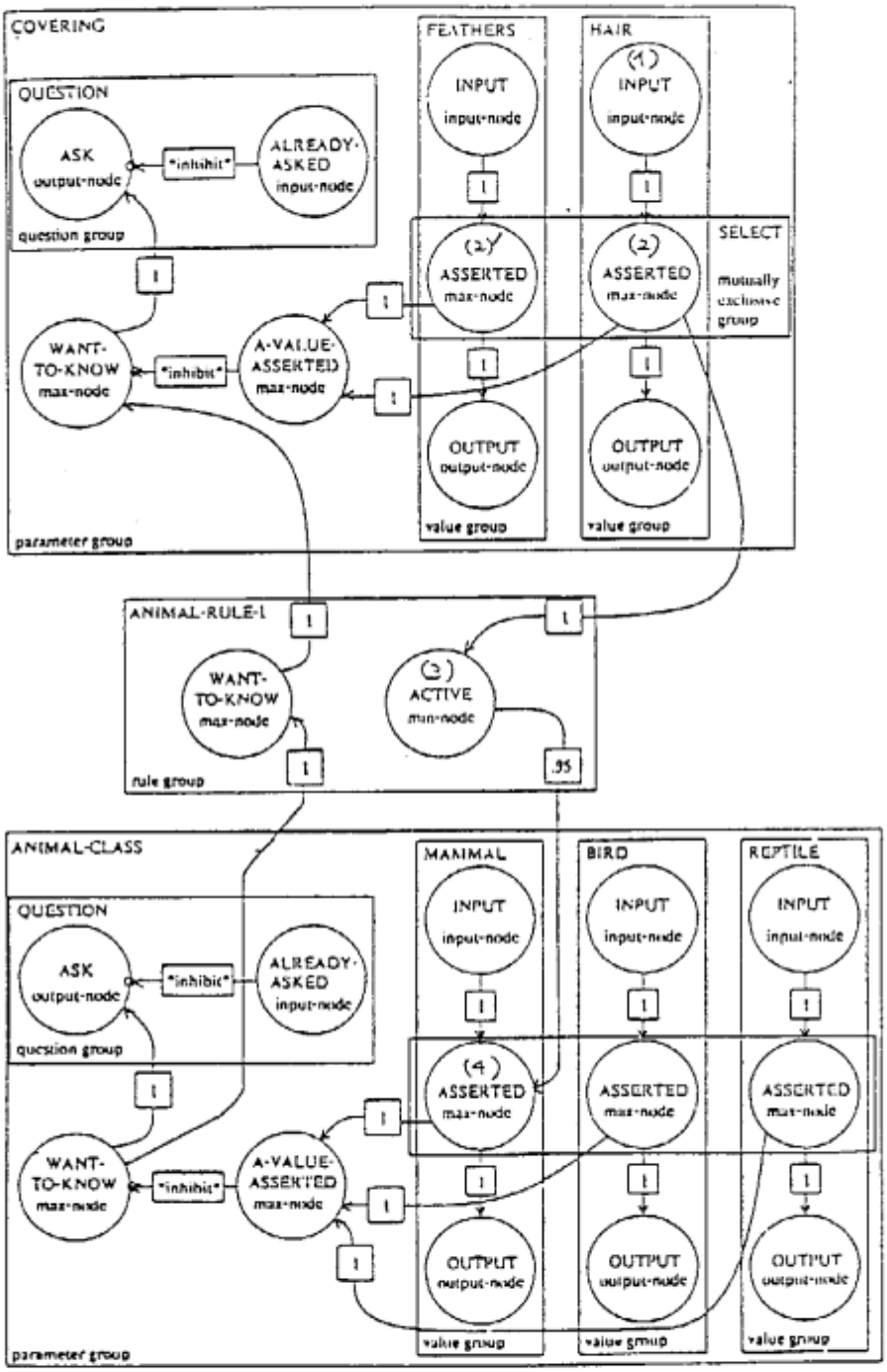


図5-4　animalに対するネットワーク [Bielloch 86]

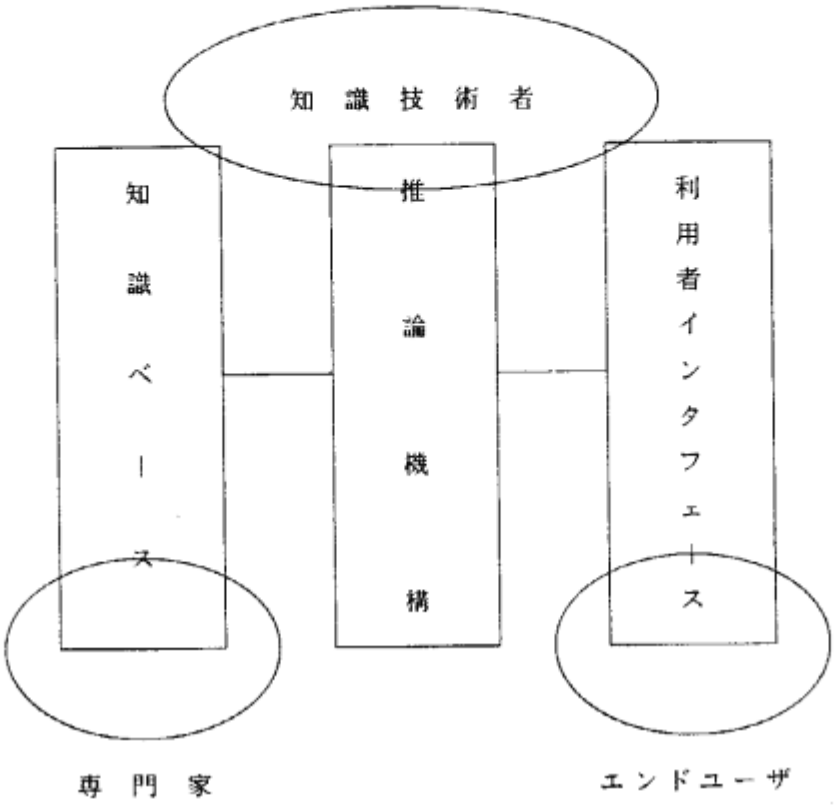


図7-1　シェルのカスタマイズの視点

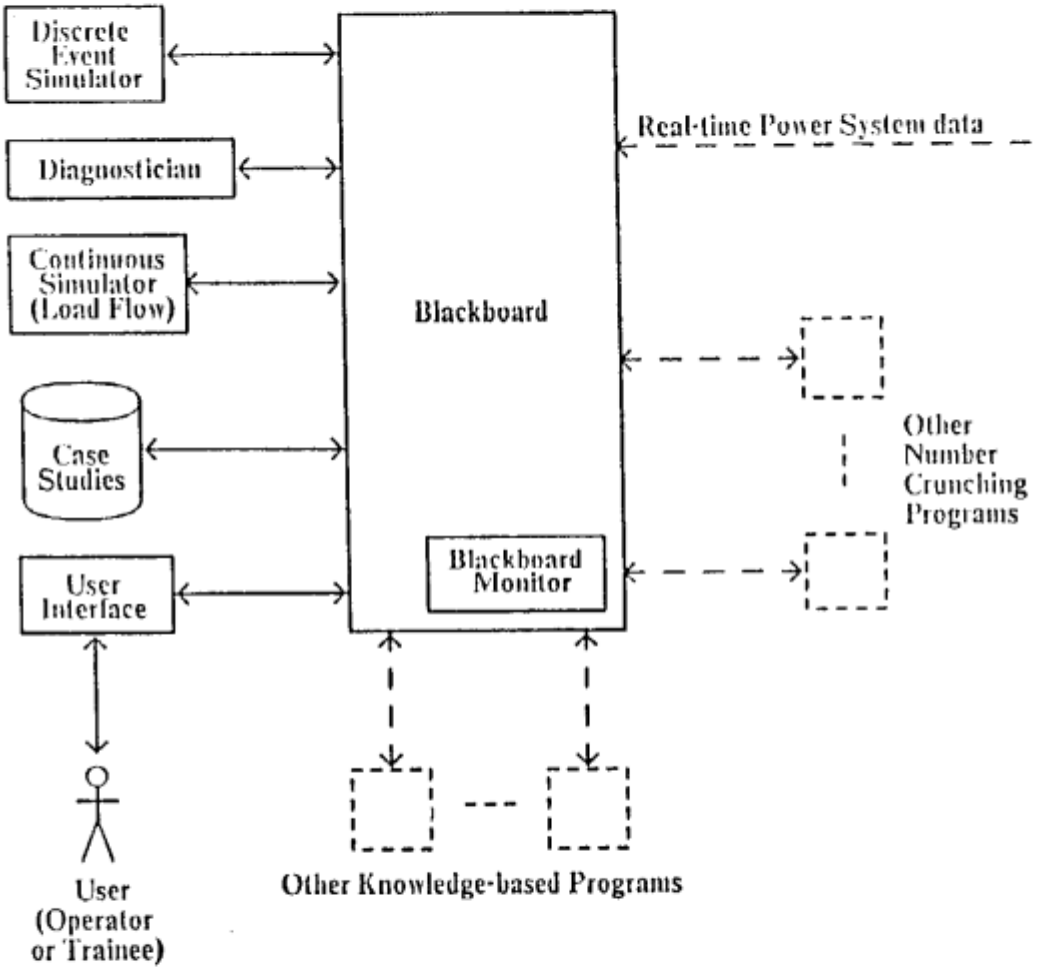
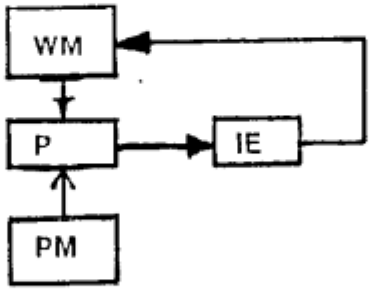
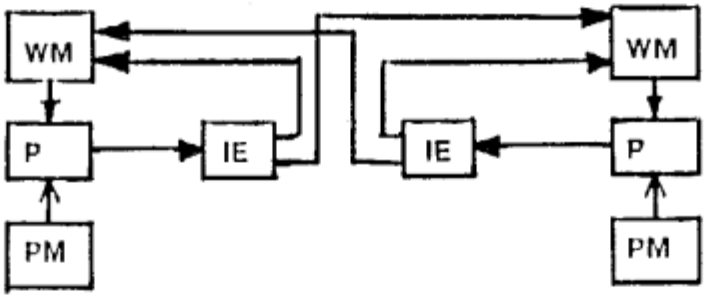


図7-2　TOAST の構成



The components of an OPS5 program.



COPS' way of implementing communications among OPS5 programs.

図7-3　COPSによるブラックボードの実現

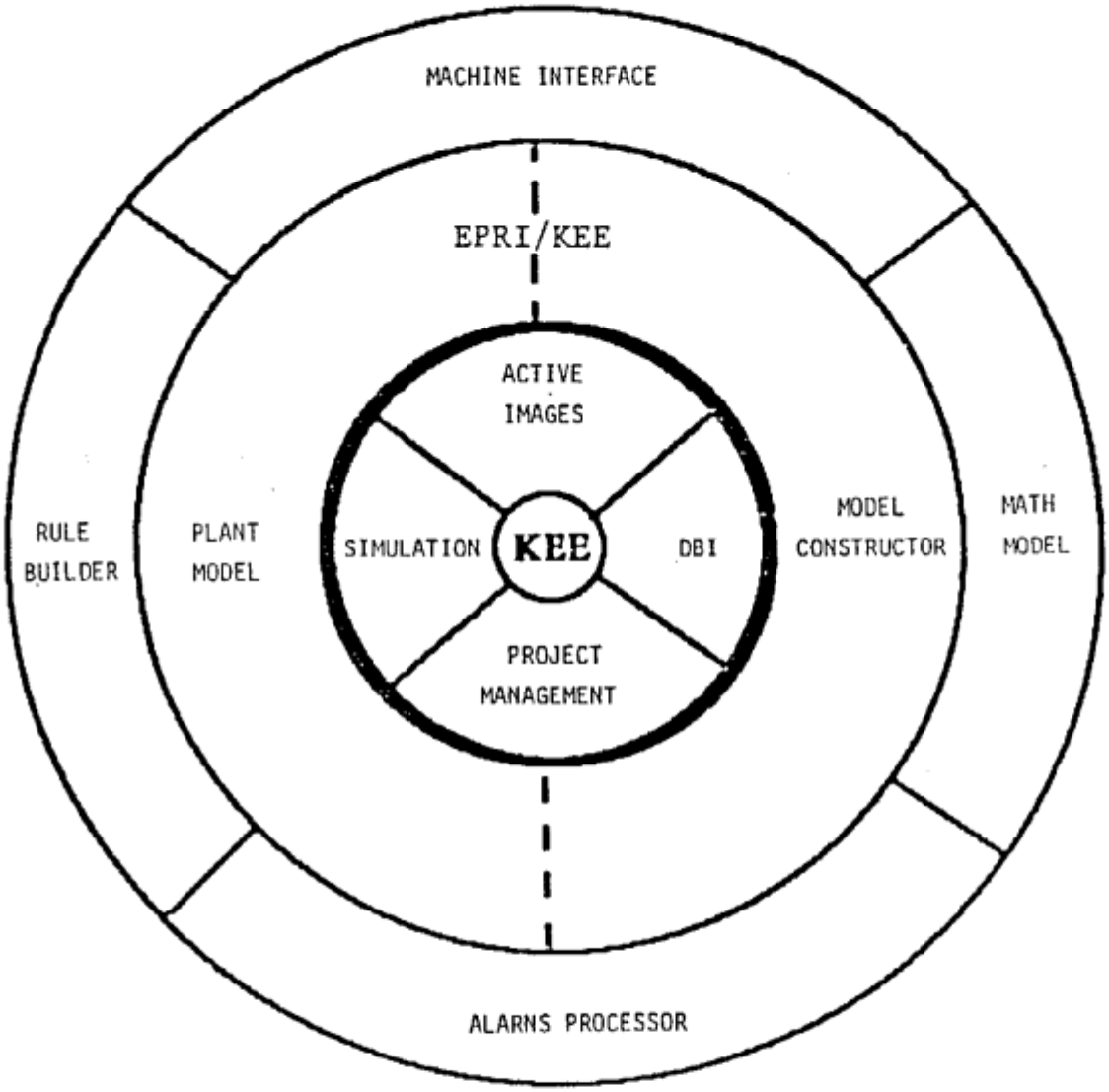


図7-4　KEE に基づいた専用シェルの概念

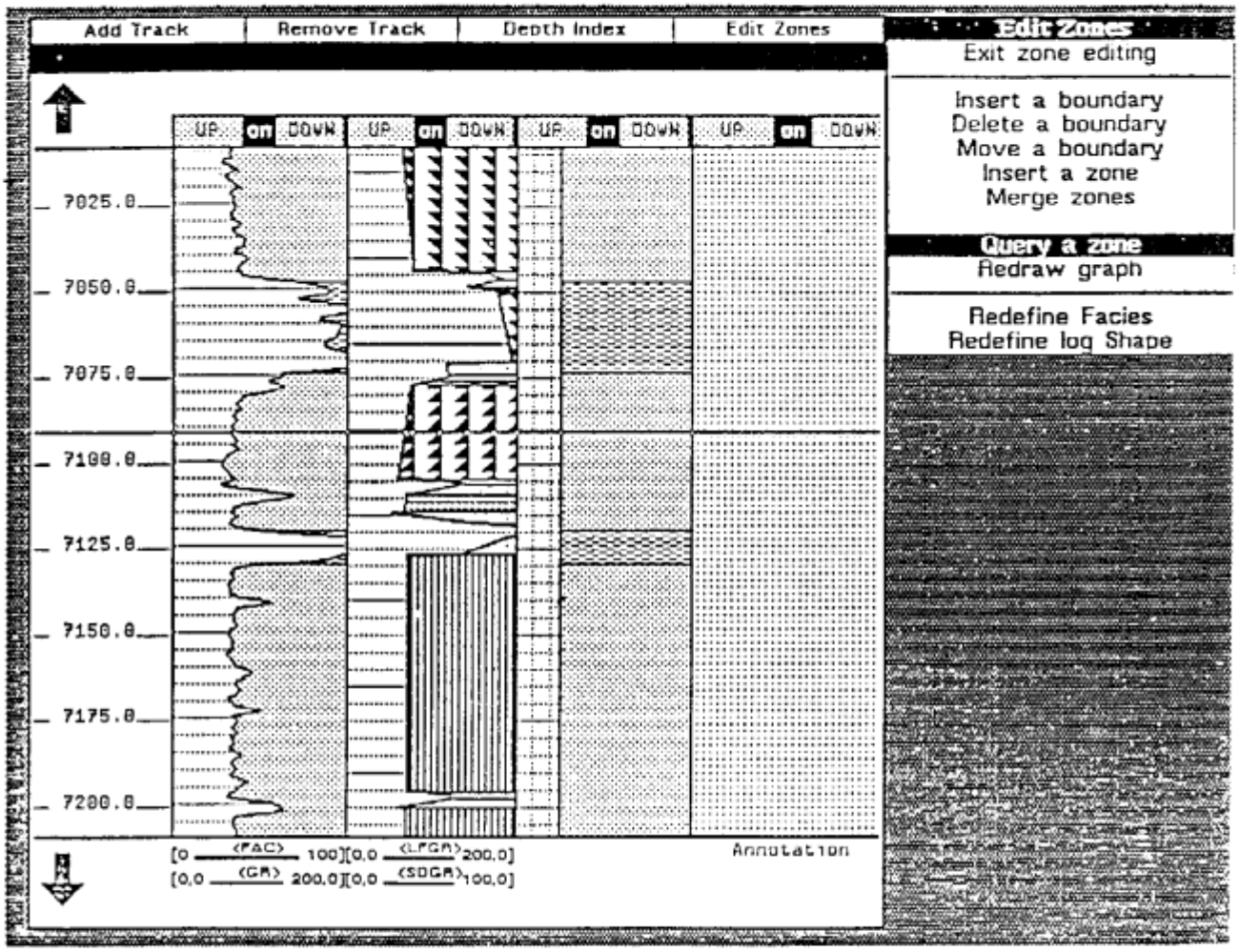


図7-5　Dipmeter Advisorの画面例

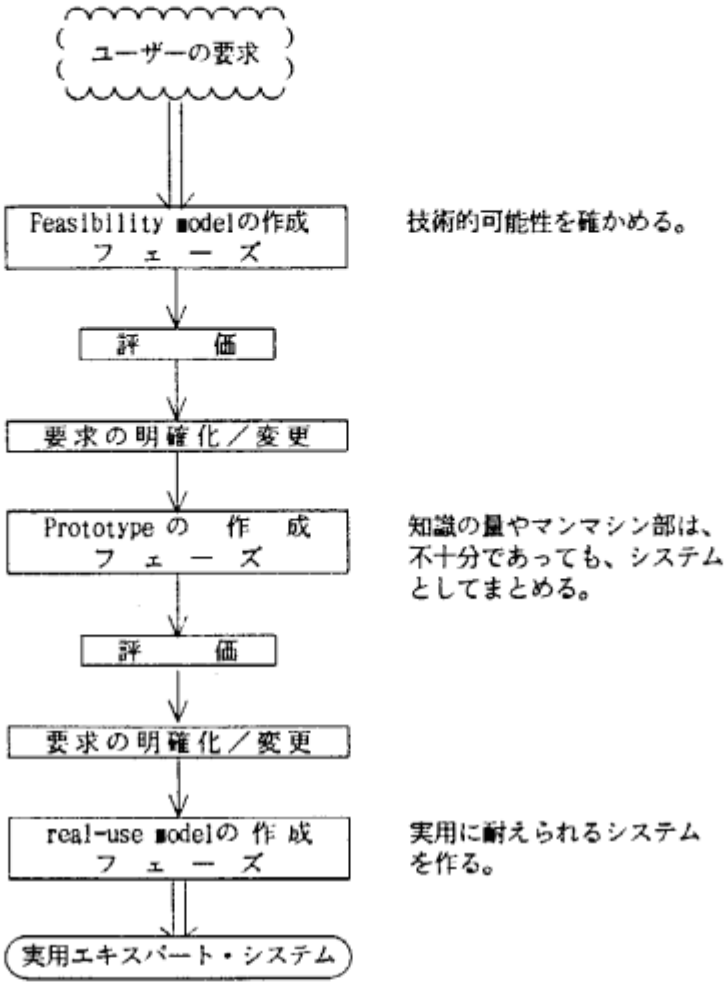


図8-1　エキスパートシステムの開発手順

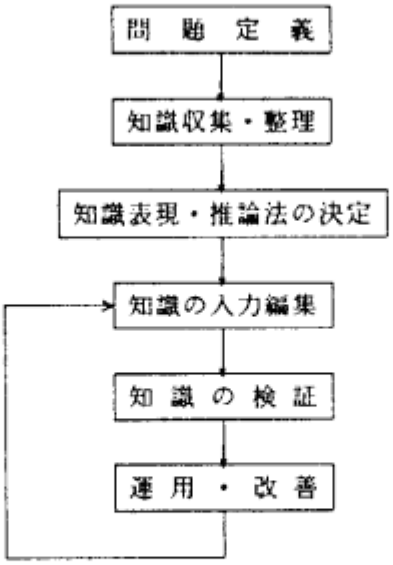


図8-2　フェーズ内の作業手順

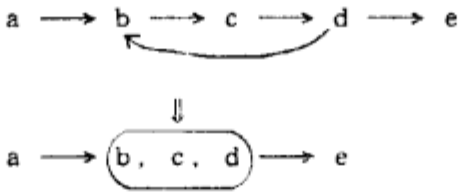


図8-3　推移律によるグループ化

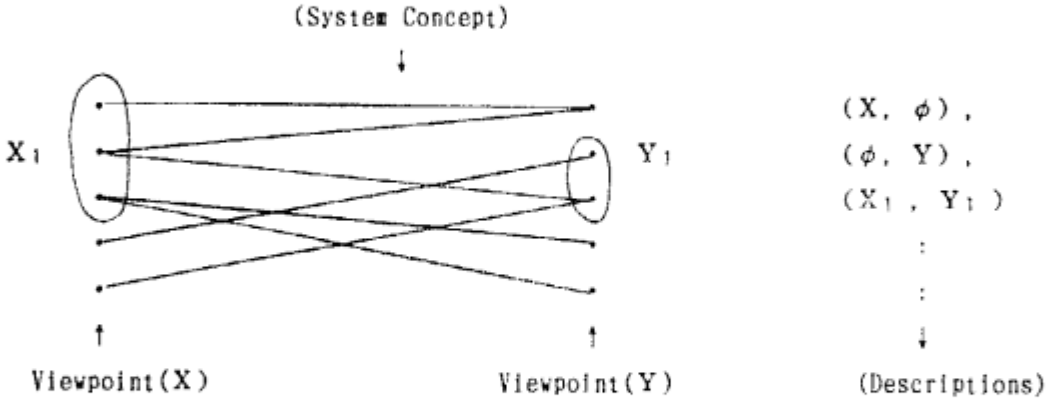


図8-4 二部グラフによるシステム構造化

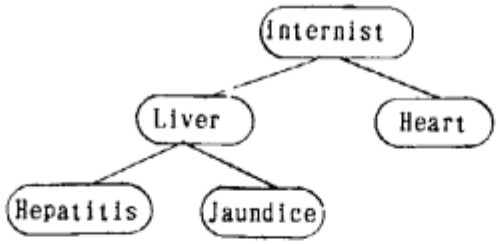


図8-5

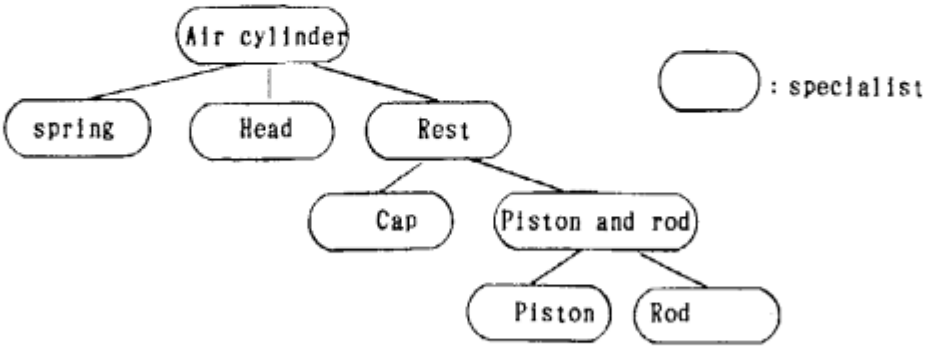


図8-6

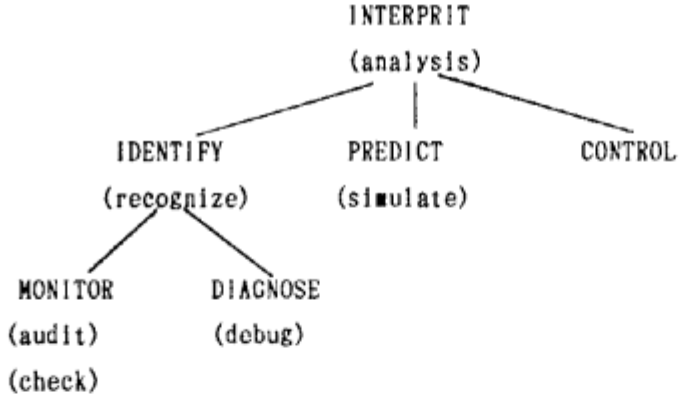


図8-7a

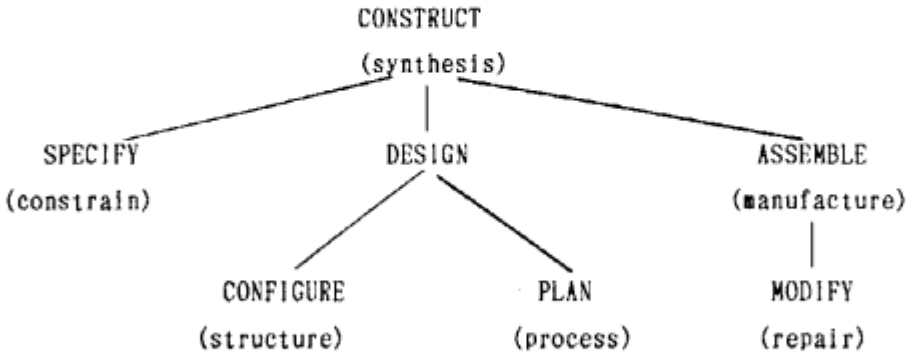


図8-7b