「人工知能」 1988.8
(慶應義塾) 訂正用原本
中村 明男

5 人工知能言語

5.1 人工知能言語の概要

人間の知的な能力を模擬する人工知能システムにおいては，知能を実現するためのソフトウェア技術が極めて重要である．このようなソフトウェアは，汎用計算機の上に実装され，ソフトウェア主体の人工知能システムとして実現される場合もあるし，あるいは，処理の高速化や低価格化などを目的に，専用ハードウェアと組合せた形で実現される場合もある．また，特殊ハードウェアを用いてハードウェア主体で実現される場合も考えられる．本章では，このような人工知能システムを実現するための基本となるソフトウェアを記述するための言語について述べる．

5.1.1 人工知能システムのモデル

人工知能システムは，人間の頭脳の知的作用の一面をモデル化したものと考えられるので，人工知能言語は，このようなモデルをできるだけ正確に，かつ容易に記述するのが目的である．ここでは，人工知能システムの簡単なモデルとして，図5.1に示す知識処理モデルを用いて説明する．

知識ベースには，

「太郎の母は花子である」

「XがYの母ならば，Xは女である」

などの事実や規則が蓄えられる．推論エンジン（推論プログラム）はこの知識を利用して，与えられた問題を解決するための推論を行う．基本的な推論方式としては，前向き推論と後向き推論がある．前向き推論（forward reasoning）は，

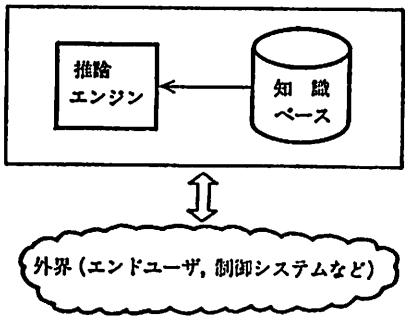


図 5.1 知識処理モデル

「太郎の母は花子である」

という事実と，

「XがYの母ならば，Xは女である」

という規則から，

「花子は女である」

という結論を導く推論方式である．後向き推論（backward reasoning）は，

「花子は女か？」

という問題に対して，上の規則を逆方向に

「Xが女であるためには，XがYの母であればよい」

と解釈することにより，与えられた問題を解く代わりに

「花子は母か？」

という問題を解くことを試み，

「太郎の母は花子である」

という事実から，結局，与えられた問題の正しさを確認する推論方式である．

このような知識処理モデルを記述するためには，次のような言語機能が必要である．

① 事実や規則などの知識の表現形式

② 知識の利用方法を決める推論機構

人工知能システムの記述言語には，プリミティブな基本機能だけを備え，ユーザが開発したいシステムに最適な知識表現形式と推論機構を，この基本機能を

用いて作成するものと，特定の知識表現や推論機構を言語機能として備え，ユーザが比較的容易にシステムを作成できるようにしたものとがある．本書では，便宜上，前者を人工知能用基本言語，後者を知識表現言語と呼ぶ．

5.1.2　人工知能用基本言語

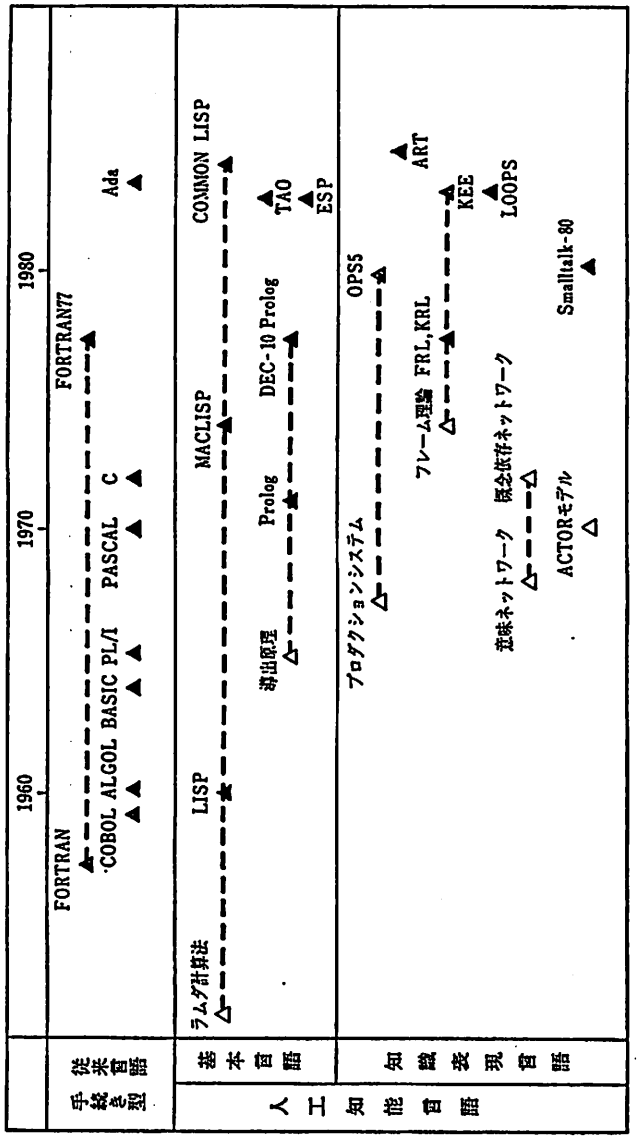
人工知能用基本言語として広く用いられているものに，LISPがある．表5.1に示すように，このLISPと同じ頃に開発され，現在に至るまで情報処理の分野で幅広く用いられているFORTRANと比較してみると，LISPの特徴は，記号の列として表現される知識を，リスト構造という単純なデータ型に対応させて，自由に操作できる点にある．

表 5.1　人工知能用基本言語と従来言語の比較

	LISP	FORTRAN
開発時期	1960年頃	1957年
適用分野	人工知能システム	情報処理システム
言語の種類	非手続き型言語（関数型言語）	手続き型言語
処理の対象	記号データ	数値データ
処理内容	リスト処理	計算処理

もう一つの基本言語として，述語論理に基づいたPrologがある．Prologは，事実や規則などの知識表現形式や後向き推論機構を備えているため，同じ問題をLISPで記述した場合よりも，プログラムが短かくてすむことが多い．前項の分類では，Prologは論理型の知識表現言語と考えてもよいが，実際には人工知能システム用のプログラミング言語として用いられることが多いので，本書では基本言語として位置づける．

図5.2に本章で紹介する人工知能言語の開発の経過を，従来の手続き型言語との対比で示しておく．そのうちのいくつかの言語で，階乗計算プログラムを記述した例を図5.3に示す．2.3.3の図2.9およびこの図から，人工知能言語と従来言語では，言語の構文が大きく異なっていることがわかる．



△：理論　▲：言語，システム

図 5.2　人工知能言語の開発時期

```
      INTEGER FUNCTION FACT(N)
        FACT = 1
        DO 10 I = 1, N
   10     FACT = I*FACT
        END
```

(a) FORTRAN (手続き型言語)

```
      factorial(n)
        int n;
      {
        if(n==0) return(1);
        else return(n * factorial(n-1));
      }
```

(b) C (手続き型言語)

```
      (defun factorial (num)
        (if  (zerop num) 1
                    (* num (factorial (-  num 1)))))
```

(c) LISP (関数型言語)

```
factorial (0, 1).
factorial (N, M): -T1 is N-1, factorial (T1, T2), M is N * T2.
```

(d) Prolog (論理型言語)

```
      factorial: x
        |fact|
        fact ← 1.
        1 to : x do : [:n|fact ← n * fact].
        ↑ fact
```

(e) Smalltalk-80 (オブジェクト指向型言語)

図 5.3 いくつかのプログラミング言語による階乗計算プログラムの記述例

5.1.3 知識表現言語

現在，実用化が進んでいる知識処理を主体とした人工知能システムでは，知識をどのような形で記憶し，利用するかが重要な課題である．このような知識表現言語としては，今までに数多くの提案があるが，ここでは，そのうちの代表的な次の5種類について述べる．

① プロダクションシステム：ノウハウ的知識を「if～then～」というルール形式で簡潔に表現

② フレーム：概念的知識のまとまりを単位とし，概念間の関係を階層構造で簡潔に表現

③ 意味ネットワーク：概念や実体をノード（節点）で表現し，それらの間の関係はネットワーク構造で自由に表現

④ オブジェクト指向言語：概念や実体をその機能に着目したオブジェクトで表現し，それらの間の静的関係を階層構造で表現するとともに，それらの機能実現上の動的関係をメッセージ送信で表現

⑤ ハイブリッド型言語：複数の知識表現形式を複合して，多種多様な知識を柔軟に表現

図5.2には，これらの知識表現言語の開発時期も合わせて示した．

5.2 リスト処理言語LISP

1960年頃に米国マサチューセッツ工科大学 (MIT) のJ. McCarthyらが開発したリスト処理言語LISP (LISt Processor) は，その後，人工知能における各種の研究分野で幅広く用いられてきた．LISPには，少しずつ言語仕様の異なる方言が多いが，代表的なものとしてMacLISPとInterLISPがある．1984年にはそれらの標準化を目的としたCommon LISPが開発された．

LISPの一般的特徴は次のようなものである．

① リストというデータ構造とそのリストに対する3種類の操作 (car, cdr, cons) が基本である．

② プログラムはS式と呼ばれる表現形式のリストで記述される．

③ プログラムは関数として記述され，ラムダ計算法に基づいて実行される．

さらに，Common LISPでは，データ型およびモジュール化機能の充実や，変数の有効範囲規則の単純化を行い，プログラミング機能の向上を図っている．

以下では，これらの特徴をCommon LISPプログラムの例を用いて具体的に説明する．

5.2.1 リ　ス　ト

LISPの基本となるデータ構造はリストであり，その例を以下に示す．

① 学生名のリスト
(John Richard Martin George William)

② 学生の点数のリスト
(82 76 68 95 74)

③ 名前と点数のリストのリスト
((John 82) (Richard 76) (Martin 68) (George 95) (William 74))

上の例はリストの要素がそれぞれ記号，数，リストになっている．LISPのデータの最小単位である記号と数をまとめてアトム (atom) と呼ぶ．リストの一般形は，

(S式　S式　・・・　S式)

の形をしており，S式とはアトムまたはリストである．したがって，上記③のようにリストの要素がリストでもよいし，リストとアトムが入り混ってもよい．

5.2.2 関数と演算

LISPのプログラムは，関数を定義し，それを呼び出すように記述されるので，関数型言語 (functional language) といわれる．この関数の定義方法は2種類あり，その一つはラムダ式 (lambda expression) を用いるものである．通常，われわれは関数を$f(x)$と表現することが多いが，これではxを変数とする関数自身を表しているか，あるいはxにおけるfの値を表しているかがあいまいになる．そこで，LISPの基礎となるラムダ計算法 (lambda calculus) では，関数自身を表す場合は，$\lambda x f(x)$のようにλの直後に変数を並べたラムダ式を用いる．Common LISPではこのラムダ式を

(lambda　変数リスト　本体)

の形で記述する．

もう一つの関数の定義方法は，関数に名前をつけて，

(defun　関数名　変数リスト　本体)

の形で記述するもので，この方法による関数定義の例を④～⑧に示す．

④ 合計点と人数から平均点を求める．

```
(defun average (sum num)
  (/  sum num))
```

これは，averageという関数は，入力として合計点sumと人数numをもち，sumをnumで割った値を返すことを示している．

⑤ 上記②の点数リストの合計を求める．

```
(defun sum (marklist)
  (if (null marklist) 0
                (+ (car marklist) (sum (cdr marklist)))))
```

この関数の本体は，

(if　条件　式1　式2)

の形をしており，条件が真ならば式1，条件が偽ならば式2の値を返す．条件の(null marklist)は，入力の点数リストが空リストか否かを判定しており，空リストの時は，式1すなわち0が関数sumの値となる．空リストでなければ式2の値となる．式2において，carはリストの第1要素を取り出す関数，cdrはリストの第1要素を除いた残りのリストを返す関数である．関数呼び出しの一般形は，

(関数名　引数　・・・　引数)

である．上記④および⑤でnull，＋，／で始まるリストも関数呼び出しになっている．特に⑤においては，sumという関数定義の本体の中でsum自身が呼び出されているが，このような関数を再帰関数 (recursive function) と呼ぶ．たとえば，上記②の点数リストの合計点を求める場合，この関数を用いて，

(sum　(82　76　68　95　74))

を実行すると，内部でsumが5回呼び出され，右から順に

(82＋(76＋(68＋(95＋(74＋0)))))

という計算が行われ，395が返される．すなわち，ifの式2のところでは，点数リストの左端の値に，左端の値を除いたリストの合計点を加算する計算が行われる．sumが5回目に呼び出された時は，入力リストが空になっているのでif

の式1すなわち0が返されている.

⑥　上記②の点数リストの要素数を求める.

```
(defun num (marklist)
  (if (null marklist) 0
                  (1+  (num (cdr marklist)))))
```

これは⑤とほぼ同じ構造をしている. (1+　引数) は (+　1　引数) の簡略形式である.

⑦　点数リストの合計と要素数を同時に求める.

```
(defun sumnum (marklist &optional (sum 0) (num 0))
  (if (null marklist) (list sum num)
  (sumnum (cdr marklist) (+  sum (car marklist)) (1+  num))))
```

これは上記⑤, ⑥で別々に求めていた合計点と人数を同時に計算し, (合計点 人数) というリストの値を返す関数である. ただし, 合計点の求め方は⑤とは逆に左から順に加算する方式にしてある.

1行目の&optionalは, sumnum関数の第2引数sumと第3引数numは, 関数呼び出しの時に対応する値が指定されなくてもよく, その時は各々0になることを示している. 2行目のlistは, 引数を要素とするリストを返す関数である.

⑧　点数リストの平均点を求める.

```
(defun average (marklist &aux (work (sumnum marklist)))
  (if (null marklist) 0
                  (/ (car work) (cadr work))))
```

1行目の&auxは, この関数内だけで用いられる局所変数workを宣言しており, その初期値は関数呼び出し (sumnum marklist) の値となることを示している. この場合, ⑦で述べたように第2, 第3引数が省略されているのでsumnum関数の実行開始時は引数sum, numはともに0となり, ⑦の3行目のsumnumの再帰呼び出しが5回実行された後, (395　5) というリストを返す. これが⑧のworkに代入され, ⑧の3行目で395/5が計算され, 79がaverageの値とし

て求まる. (cadr work) は ((car (cdr work))) と同じ処理で, workの第2要素の5を返す.

5.2.3　リスト処理

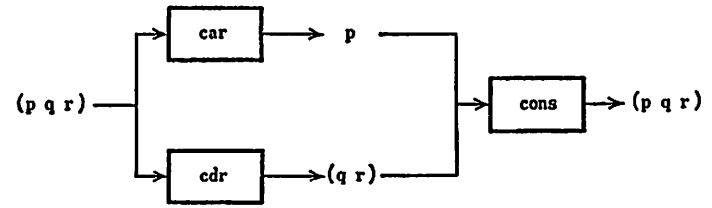
LISPに特徴的な, リスト処理を中心としたプログラム例として, 上記③の名前-点数リストの要素を, 点数の高い順に並べ替える関数を図5.4に示す. 並べ替えは, 挿入ソート法 (insertion sort) という簡単な方法を用いている.

```
(defun nmsort (nmlist)
  (if (null nmlist) ()
                  (insert (car nmlist) (nmsort (cdr nmlist)))))
(defun insert (student sortlist)
  (cond ((null sortlist) (list student))
        ((< (cadr student) (cadar sortlist))
          (cons (car sortlist) (insert student (cdr sortlist))))
        (t (append (list student) sortlist))))
```

図 5.4　成績順にソートするCommon LISPプログラム

nmsortはリストの第1要素を除いた残りをまず並べ替えて, その後, 第1要素を適当な位置へ挿入する再帰関数になっており, 実際にはリスト要素を右から順に取り出して並べている. このプログラムおよび⑤～⑧で用いられたリスト処理関数をまとめると次のようになる.

(a)　基本関数：car, cdr, cons



(car (p q r)) ⇨ p
(cdr (p q r)) ⇨ (q r)
(cons p (q r)) ⇨ (p q r)

図 5.5　リスト処理の基本関数

（b） 合成関数：cadr, cadar

（c） 操作関数：list, append

基本関数は図5.5に示す関係をもつ．合成関数はcarおよびcdrの組合せを示し，(cadar p) は (car (cdr (car p))) と同じである．リストを組み立てる関数としては，consが第1引数のリスト要素を，第2引数のリストの先頭に加えるのに対し，listは，任意の個数の引数をすべてリスト要素としたリストを生成する．appendは，第1引数のリストと第2引数のリストを一つのリストに統合する．各々の例を次に示す．

(cons (p q)(r s)) ⇨ ((p q) r s)
(list (p q)(r s)) ⇨ ((p q)(r s))
(append (p q)(r s)) ⇨ (p q r s)

図5.4ではこれらのリスト処理関数が多く用いられているので，上記③の名前-点数リストがnmsort関数の入力となった場合の，並べ替えの過程を読者自ら確めて欲しい．

なお，2行目の () は，入力が空リストの時に空リストを返すという意味の定数で，nilと書いてもよい．5行目のcondはifと似ているが，

```
(cond (条件1　式1)
      (条件2　式2)
      (条件3　式3)
          ・・・　)
```

のように条件が複数指定でき，上から順に見て，最初に真となる条件に対応する式が実行される．図5.4の例で条件2の<は値の比較演算関数で，挿入する学生の点数が，ソート済みリストの先頭の学生の点数より小さいか否かを判定している．条件3のtは，常に真であることを意味する定数であり，条件1も条件2も偽の時は必ず式3が実行される．

最後に，名前-点数リストを読み込み，図5.4のnmsort関数を用いて，成績順に並べ替えた結果を出力するプログラムを以下に示す．

```
(defun listout ( )
  (print "Input a list of (name marks)")
  (setq inlist (read))
  (setq outlist (nmsort inlist))
  (print "A sorted list of students")
  (print outlist))
```

setqは変数に値を代入するもので，変数inlistに入力リストを代入したり，並べ替えられたリストを変数outlistに代入している．

5.3 論理型言語 Prolog

1972年に，フランスのマルセーユ大学のA. Colmerauerが開発した論理型言語Prologは，当初は構文解析ツールや，導出原理に基づく定理証明ツールとして用いられていた．1974年になって，ロンドン大学のR. Kowalskiがこれにプログラミング言語としての解釈を与え，また1977年には，エジンバラ大学で実用的な性能の処理系が開発された．さらにその後，日本の第5世代計算機プロジェクトの基本言語として採用されたことから，人工知能の研究分野を中心に広く使われ始めている．

Prologの一般的特徴は次のようなものである．

① プログラムは述語論理の一形式であるホーン節で記述され，導出原理に基づいて実行される．

② ユニフィケーションと呼ばれる強力なパターンマッチング機能がある．

③ 自動的なバックトラックによる探索機能がある．

以下では，これらの特徴をはじめとするPrologのプログラミング機能について，例を用いて具体的に説明する．なお，言語仕様は，広く用いられているエジンバラ大学のDEC-10 Prologに従うが，記号として日本語も使用することにする．

5.3.1 論理型知識表現

Prologでの知識表現の基本となる事実，規則，質問の記述例を図5.6に示す．

```
〔事実〕
(f1)　父（信虎，信玄）.
(f2)　父（信玄，義信）.
(f3)　父（信玄，勝頼）.
(f4)　父（信玄，盛信）.
(f5)　父（勝頼，信勝）.

〔規則〕
(r1)　祖父（X，Z）:－父（X，Y），父（Y，Z）.
(r2)　祖父（X，Z）:－父（X，Y），母（Y，Z）.

〔質問〕
      ?－祖父（X，信勝）.
```

図 5.6　Prologによる家族関係の知識表現の例

```
〔ホーン節の一般形〕
A：-A1, A2, …, Am.

〔ホーン節の右辺が空〕
A.

〔ホーン節の左辺が空〕
?-A1, A2, …, Am.
```

図 5.7　Prologの構文

左端の番号は本書での説明用に便宜的につけたものである．父(a, b)の形式の5個の表現は，「aはbの父である」という事実を表している．1番目の規則は，「XがZの祖父であるためにはXがYの父であり，YがZの父であればよい」ことを表現している．最後の質問は，「信勝の祖父は誰ですか？」を意味し，「X＝信玄」という回答が出力される．ここで，XやYのように大文字の英字で始まる文字列は，変数を意味する．

これらの例に対応するホーン節，または節と呼ばれるPrologの基本構文を図5.7に示す．AおよびAiは項（term）と呼ばれ，以下のいずれかである．

① 変数：X，Y，Data，Rest，Routeなど

② 定数：信玄，父，511，pathなど

③ 構造：父（信玄，勝頼），path(X，Y，[X，Y])など

ホーン節の一般形は，右辺のすべての項Aiが成立すれば左辺の項Aが成立することを示しているので，

結論：-条件1，条件2，…，条件m.

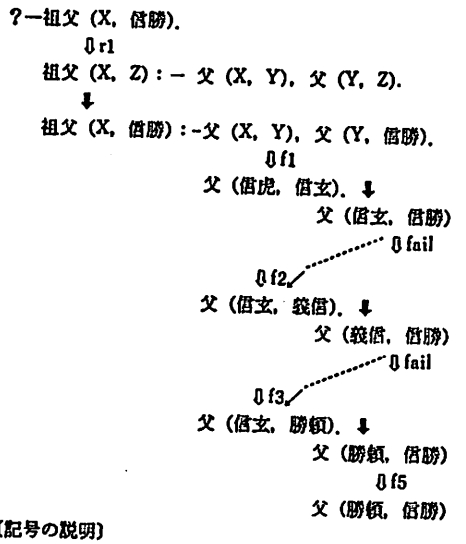
を意味している．したがって，右辺がない場合は結論が無条件に成立していること，すなわち事実を意味することになる．また左辺がない場合は，右辺の条件が成立するか否かを調べよ，ということを意味している．

5.3.2　プログラム実行方式

図5.6の家族関係のプログラムの実行過程を，図5.8を用いて説明する．「信勝の祖父は誰ですか？」という質問を意味する祖父（X，信勝）は，Prologではゴールと呼ばれ，このゴールすなわち「信勝の祖父はXである」が正しいことを証明する過程が，プログラムの実行過程である．その過程で変数Xに代入された値が，この質問の答となる．

まずゴールが与えられると，図5.6のプログラムを上から順に調べて，そのゴールと対応がとれる節を探す．祖父（X，信勝）と規則r1の左辺は，変数Zを信勝に対応させると一致することがわかる．この操作をユニフィケーション（unification）と呼ぶ．これは，ゴールが成立するためには，規則r1の右辺が成立すればよいことを意味するので，次に右辺の項をサブゴールと考えて，同じ操作を繰り返す．

右辺の1番目のサブゴールである父（X，Y）はXを信虎，Yを信玄に対応させることにより，事実f1と一致する．次に2番目のサブゴールである父（Y，信勝）のYを信玄で置き換えた父（信玄，信勝）と一致する節を探す．ところ



〔記号の説明〕
◊a　：事実または規則aとユニフィケーションに成功
←…◊fail：ユニフィケーションに失敗（バックトラック発生）
⬇　：変数をその変数に結合した値で置換

図 5.8　Prologプログラムの実行過程

が，プログラム内にはないのでこのユニフィケーションは失敗する．そこで，もう一度，1番目のサブゴールに戻って，事実f1以外で一致する節を探す．このようにユニフィケーションに失敗した時に，これまでのゴールの証明過程を後戻りしてやり直すことを，**バックトラック** (backtrack) という．

サブゴール父 (X, Y) は事実f2とも一致するが，これはf1の時と同様に2番目のサブゴールで失敗する．ここでもう一度バックトラックが生じ，1番目のゴールをf3と一致させた時，2番目のサブゴールはf5と一致し，r1の右辺のサブゴールがすべて成功して，この証明過程は終了する．この時のXに対応した値玄が答となる．

このようにPrologではユニフィケーションとバックトラックが基本的処理である．

5.3.3 手続き的解釈

このようなPrologの実行過程を従来のPascalやFORTRANなどの手続き型言語に対応させると，

$A : -A_1,\ A_2,\ \cdots,\ A_m.$

という節は，

procedure A; begin $A_1,\ A_2,\ \cdots,\ A_m$ end;

と解釈できる．すなわち，節の左辺が手続き定義の頭部，右辺が手続き呼び出しの列からなる手続き本体に対応する．

一方，手続き型言語との大きな違いは，ユニフィケーションとバックトラックの機能があることである．そのため，前述の家族関係のプログラム例でいえば，次の4種類の質問は一つのプログラムで処理できる．

① ?-祖父 (倌玄, 倌勝)…「倌玄は倌勝の祖父ですか？」

② ?-祖父 (X, 倌勝)…「倌勝の祖父は誰ですか？」

③ ?-祖父 (倌玄, Y)…「倌玄は誰の祖父ですか？」

④ ?-祖父 (X, Y)…「XがYの祖父であるようなXとYはありますか？」

これは，Prologのゴールの引数が定数でも変数でもよく，定数の時は入力パラメータ，変数の時は出力パラメータとみなされるためである．手続き型言語では，一般に上の4種類の質問に対して，別々の処理プログラムが必要になる．


```
average([],0).
average(Marklist,ANS) :- sumnum(Marklist,0,0,Sum,Num),
                         ANS is Sum / Num.
sumnum([],Sum,Num,Sum,Num).
sumnum([Data|Rest],S,N,Sum,Num) :- S1 is S + Data,
                                   N1 is N + 1,
                                   sumnum(Rest,S1,N1,Sum,Num).
```

図 5.9 平均点を求めるPrologプログラム

5.3.4 プログラミング機能

次にPrologのもつ，その他のプログラミング機能について説明する．まず，5.2.2の⑦，⑧で述べた平均点を求めるLISPプログラムに対応する，Prologプログラムを図5.9に示す．アルゴリズムは両者ともほぼ同じであるが，次の点が異なる．

① LISP関数は自分自身が値をもつが，Prologの項は値をもたないため，計算結果を返す出力パラメータが付加されている．averageの第2引数とsumnumの第4，第5引数がそれである．

② LISP関数の本体の定義の最初に，ifを用いて入力リストが空リストの時の処理が記述してあるが，このような処理はPrologでは，節の左辺の引数部分のユニフィケーション機能を用いて簡単に記述できる．averageおよびsumnumの最初の節は，第1引数の入力リストが空リストの時にユニフィケーションに成功し，averageでは第2引数の0を返す．sumnumでは入力パラメータである第2，第3引数の値をそのまま第4，第5引数の値として返す．

③ LISPのsumnum関数の (car marklist) と (cdr marklist) にあたるリスト処理は，Prologでは [Data|Rest] という記述でよい．

なお，isはその右側の算術式の値を左側の変数に代入する演算子である．

```
listout :- write('Input a list of [name,marks]'),
           read(Inlist),
           nmsort(Inlist,Outlist),
           write('A sorted list of students'),
           write(Outlist).

nmsort([],[]).
nmsort([Student|Rest],Out) :- nmsort(Rest,Out1),
                              insert(Student,Out1,Out).

insert(S,[],[S]).
insert([N,M],[[N1,M1]|Rest],[[N1,M1]|Out]) :- M < M1,
                                               insert([N,M],Rest,Out).
insert(X,In,[X|In]).
```

図 5.10 点数順にソートするプログラム

次に，5.2.3で述べた，点数順にソートするLISPプログラムに対応したPrologプログラムを図5.10に示す．上記③で述べたように，リスト処理が非常に簡単に記述できる．

5.3.5 縦型探索による推論

Prologの基本的な問題解決方法は，縦型探索によるものである．その説明のために，図5.11のような飛行ルートを選択するプログラムを考える．事実f1～

```
(f1) flight (札幌, 東京).
(f2) flight (札幌, 名古屋).
(f3) flight (札幌, 大阪).
(f4) flight (東京, 名古屋).
(f5) flight (名古屋, 大阪).
(f6) flight (東京, 沖縄).
(f7) flight (名古屋, 沖縄).
(f8) flight (大阪, 沖縄).
(r1) path (X, Y, [X, Y]) : -flight(X, Y).
(r2) path (X, Y, [X|SubRoute]) : -flight(X, W), path(W, Y, SubRoute).
```

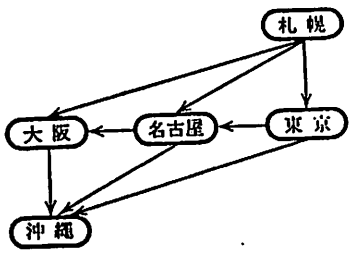


図 5.11 飛行ルート選択のPrologプログラム

f8は五つの都市の間を結ぶ8種類の航空路を示し，規則r1，r2は都市Xから都市Yへ行く経路が存在する条件を示している．pathの第3引数はその経路を経由する都市のリストを示す．なお，問題を簡単にするためにflight(a, b)はaからbへの1方向の航空路を示し，bからaへの航空路は含まないとする．

このプログラムを用いて，札幌から沖縄へ行く経路を求める問題を考える．先の図5.8の家族関係プログラムの実行過程と同じように，図5.12のような実行過程で［札幌，東京，沖縄］という経路が求まる．

次に2番目の解を要求すると，図5.12の最後のf6とのユニフィケーションに成功したところが，失敗した場合と同じ動きをする．すなわち，そこでバックトラックが生じ，一つ手前のr1のユニフィケーションのところに戻り，r1の代わりにr2とのユニフィケーションが行われる．以下同様にして，第2の解［札幌，東京，名古屋，沖縄］という経路が求まる．

結局，Prologではこの問題の六つの解を次の順序で見つける．

① ［札幌，東京，沖縄］
② ［札幌，東京，名古屋，沖縄］
③ ［札幌，東京，名古屋，大阪，沖縄］
④ ［札幌，名古屋，沖縄］
⑤ ［札幌，名古屋，大阪，沖縄］
⑥ ［札幌，大阪，沖縄］

このように経路探索の途中で複数の選択が可能な時に，その一つを選んで先へ進み，行き詰まれば一つ手前に後戻りして，別の選択をするようなバックトラックによる解法を，縦型探索または深さ優先探索（depth-first search）と呼ぶ．

これに対し，複数の選択が可能なものは，すべて同時に調べながら進む方法を，横型探索または広さ優先探索（breadth-first search）と呼ぶ．この場合は，乗換え回数の少ない飛行ルートが優先的に選ばれるため，解の求まる順は，①④⑥②⑤③となる．

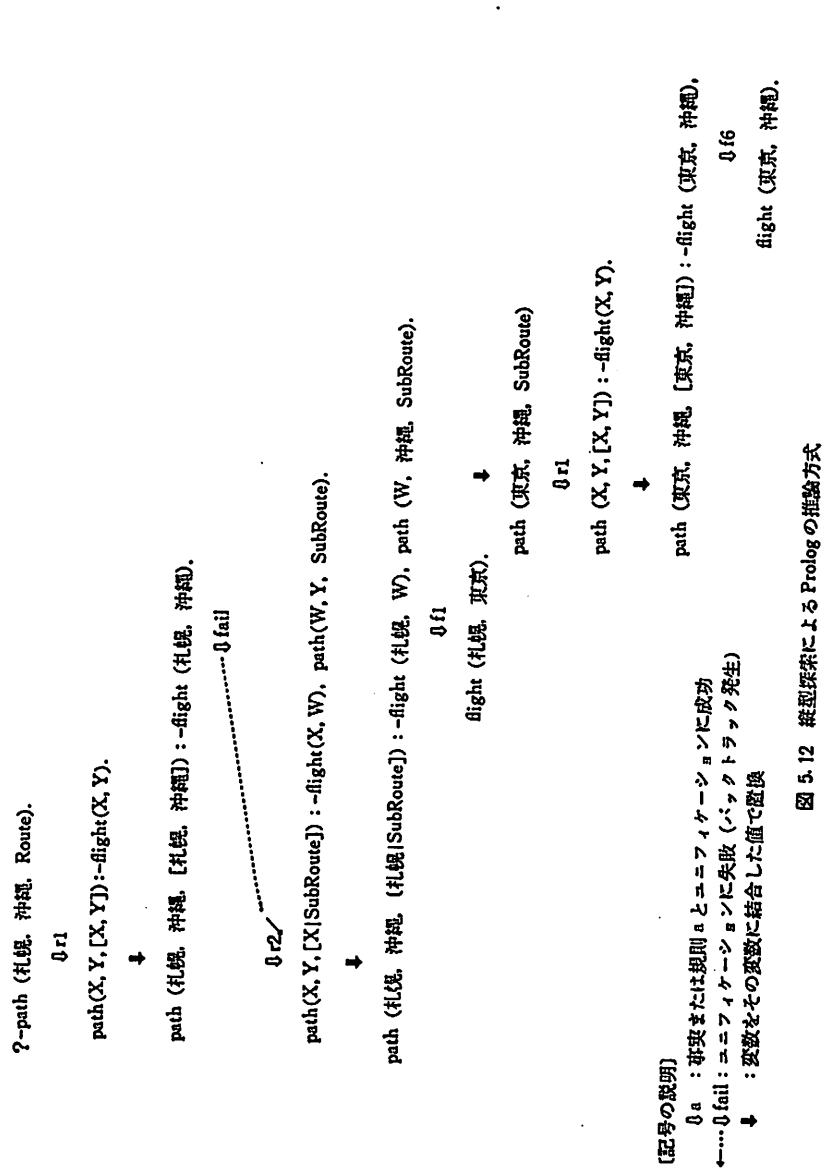


図 5.12　縦型探索によるPrologの推論方式

5.4　プロダクションシステム

1967年に米国カーネギーメロン大学のA. Newellらが，人間の認知モデルに基づく問題解決の方法として考案したプロダクションシステム（production system）は，知識工学応用システムに最もよく利用されており，6章で述べるエキスパートシステムや，その構築ツールの多くがこの機能を備えている．

プロダクションシステムの主な特徴は，次のようなものである．

① 知識は「if条件then行動」というルール形式で簡潔に表現されるため，コンピュータに不慣れな人にとっても自然で理解しやすい．

② 条件を満たすルールの実行を繰返すことにより，初期条件が与えられて結論を導く前向き推論を行うことができる．

③ 要求された行動を指示するルールの条件が満たされているか否かの判定を繰返すことにより，後向き推論を行うことができる．

以下では，これらの特徴を中心に例を用いて具体的に説明する．

5.4.1　ルール形式の知識表現

人間の世界には多種多様な知識が存在するが，「if条件then行動」のようなルール形式で表現できるノウハウ的な知識が，重要な役割をもつ場合が多い．

たとえば，自動車を運転する時の知識の一部として，次のような例を考えてみる．

① “前方に信号機があり，それが赤信号または黄信号ならば手前で停止する．青信号ならば進む”

② “そこが交差点Aならば右折する”

③ “右折する場合は，方向指示器の右ランプを点灯し，ハンドルを右へ回す．ただし，対向車または歩行者がある場合は手前で停止する”

この知識をルール形式で表現したものが，図5.13である．この知識を用いたプロダクションシステムの推論は，次のように行われる．

[初期条件]

場所　　：交差点A
信号機の状態：青信号
対向車の有無：なし
歩行者の有無：なし

(r1) if 赤信号 or 黄信号 then 停止
(r2) if 青信号 then 進行
(r3) if 交差点A then 右折
(r4) if 右折 then 右方向指示ランプ点灯
(r5) if 右折 and 進行 then ハンドル右回転
(r6) if 右折 and 対向車 then 停止
(r7) if 右折 and 歩行者 then 停止

図 5.13　ルール形式の知識表現の例

[推論過程]

ルールr2を用いて「進行」と判断．ルールr3を用いて「右折」と判断．次に「右折」の判断をルールr4に適用して「右方向指示ランプ点灯」を実行．さらに「進行」と「右折」の判断を，ルールr5に適用して「ハンドル右回転」を実行．

5.4.2 推論方式

このような推論を行うためのプロダクションシステムの基本構造は，図5.14のようなものである．ルールの集合はプロダクションメモリに保持する．「青信号」のような外界の状態や「右折」のような推論の中間結果は，ワーキングメモリに書き込む．推論エンジン（推論プログラム）は，ワーキングメモリの状態を観察し，プロダクションメモリの中に，条件部を満足するルールがあれば実行するとともに，その実行結果をワーキングメモリに反映する，という一連の動作を繰返す．このような図5.15に示す実行方式は，認知-行動サイクルと呼ばれ，人間の認知モデルに対応している．

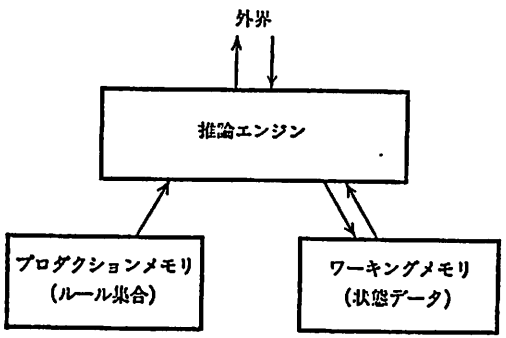


図 5.14　プロダクションシステムの基本構造

先の例では，まず初期条件の「交差点A」と「青信号」がワーキングメモリに書き込まれ，最初の実行サイクルでは，照合過程で実行可能ルールとしてr2，r3が選び出され，そのうちのr2が実行され，ワーキングメモリに「進行」が書き込まれる．以下同様にして，表5.2に示すように5回目の実行サイクルで実行可能ルールがなくなり，推論を終了する．なお，この場合は同じルールが2度は実行されないものとする．

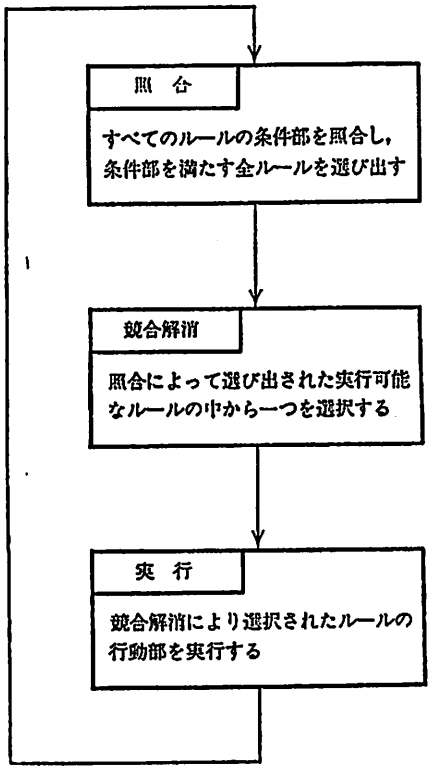


図 5.15　プロダクションシステムの推論実行方式

5.4.3 競合解消戦略

今の例では，1番目と3番目の実行サイクルにおいて，実行可能ルールが複数個あり，いずれもルール番号の小さい方を優先して実行した．このように，実行可能な複数のルールの中から一つを選択することを，競合解消 (conflict resolution) といい，その方法を競合解消戦略という．これには次のようなものがある．

① ルール記述順：ルール番号の小さい方を優先
② 最新優先：ワーキングメモリに書き込まれた最新のデータとの照合を条件部にもつルールを優先
③ 詳細優先：条件部が複雑なルールを優先
④ 重要度優先：各ルールにつけた重要度の重みの大きい方を優先

このほかにもいろいろ考えられるが，適切な競合解消戦略を設定することは，

表 5.2　プロダクションシステムの実行例

サイクル	ワーキングメモリの内容	実行可能ルール	実行ルール	出　力
1	青信号，交差点A	r2 r3	r2	
2	青信号，交差点A 進　行	r3	r3	
3	青信号，交差点A 進　行，右　折	r4 r5	r4	右方向指示 ランプ点灯
4	青信号，交差点A 進　行，右　折	r5	r5	ハンドル 右回転
5	青信号，交差点A 進　行，右　折	なし	なし	

一般に，容易ではない．

たとえば，上の例で用いたルール記述順の戦略では，初期条件に「対向車」が加わっても，4番目の実行サイクルの時にr6よりもr5が優先して実行されるために，事故が発生してしまう．この例では，r5とr6，r7の記述順を変える必要がある．

次に最新優先の戦略では，初期条件を「交差点A」，「青信号」，「対向車」の順に認識した時はr5よりもr6が先に実行されてうまくいくが，「対向車」の認識の後で「青信号」を認識した場合は，r5が先に実行されて事故が生じる．この例では，r5の条件部を詳細化して，

if右折and進行and not (対向車) and not (歩行者) then ハンドル右回転

とする必要がある．

さらに詳細優先の戦略では，3番目の実行サイクルでr4とr5が競合した時，条件部の複雑なr5が優先されるため，「右方向指示ランプ点灯」をしないで右折をしてしまう．これを防ぐには，r5の条件部の「右折」を「右方向指示ランプ点灯」で置き換えることにより，r4，r5の実行順序を固定化すればよい．

最後に重要度優先の戦略の場合は，ルールの行動部が「停止」のr1，r6，r7の重みを最大とし，r4の重みをr5より大きくすることにより，これまで述べたような不都合が生じないようにできる．

5.4.4　実用上の課題

プロダクションシステムは知識表現が簡単で保守も容易なため，広く利用されているが，ルール数の増加に伴い，次のような実行上の課題がある．

① 適切な競合解消戦略の設定が難しい．

② 照合処理に時間がかかる．

第1の問題については，すでに前項で述べたが，ルール数が増加すればさらに難しくなる．そこで実際には，ルールをグループ化してグループ単位で実行制御をしたり，グループ内のルール間の実行順序を制御するための知識を，メタ知識 (meta knowledge) として，記述できるような機能を備えたものが多い．

第2の問題は，プロダクションメモリ内のルール数，およびワーキングメモリ内のデータ数の積に比例して，照合処理の時間が組合せ的に増大するということである．そのため，多くのシステムでは，次のような方法で照合処理の高速化をはかっている．

(1) ルール間の条件部の共通部分は，1回の照合処理でまとめて1度だけ計算すればよいように，あらかじめルールをコンパイルしてネットワーク構造にしておく．

(2) 実行サイクルごとにすべてのルールの条件部を計算する代わりに，前回の実行サイクル中に変化したワーキングメモリ内のデータを条件部にもつものだけを計算し，残りについては前回の計算結果を用いる．

自動車の例では，(1)の処理はr4，r5，r6，r7の条件部が共通に有する「右折」の有無の判断を，1回ですませるようにすることである．(2)の処理は，表5.2の2番目の実行サイクルにおいて，7個のルールの条件部の照合の代わりに，1番目の実行サイクルによってワーキングメモリに加えられた，「進行」というデータを条件部にもつr5だけを照合することである．

5.4.5　拡張機能

プロダクションシステムの基本機能は以上に述べた通りであるが，実際には，

実用上の拡張機能がいろいろ考えられている.

まずルール形式の知識表現は，認知モデルに対応した「if条件then行動」のほかにも，「if前提then結論」という一般的な命題の表現にも利用できる．これはPrologでの

結論：-前提

に対応する．したがって，プロダクションシステムにおいても，条件から行動を決定したり，前提から結論を導く前向き推論のほかに，Prologのような後向き推論，すなわち，結論が与えられて，その結論を導く前提の成否を確かめる推論も可能である．先の例では，「ハンドル右回転」というゴールが与えられて，ルールをr5，r3，r2の順に探索し，「交差点A」で「青信号」の状態ならばゴールの成立を出力し，そうでなければ不成立を出力することになる．

その他の拡張機能として，不確かな知識を利用できるようにするための確信度の指定，ルールの実行制御のためのメタ知識の導入，条件部の記述に変数を使用，などがある．

5.5 フレーム

1974年に米国マサチューセッツ工科大学（MIT）のM. Minskyは，人間の記憶構造や推論過程の説明のためにフレーム（frame）の概念を提案した．これは，先に述べたプロダクションシステムや論理型言語では不十分であった知識の構造化や，知識間の関係の記述が容易なため，多くのエキスパートシステムに利用されている．

フレームによる知識表現の主な特徴は次のようなものである．

① 実世界の概念に対応する枠組みを，フレームとして自然な形で定義できる．

② その概念がもっている種々の属性は，フレームの中に記述する．この属性には手続きを付加することもできる．

③ フレーム間に階層構造を導入し，下位のフレームは上位のフレームの属性を継承できるため，知識の整理が容易である．

以下では，これらの特徴を例を用いて具体的に説明する．

5.5.1 フレームの知識表現

人間が推論による問題解決を行う時，過去の記憶を大まかな枠組みごとに想起していると考え，これをフレームに対応づける．たとえば，「会議」という言葉からは，

"ある会議室で何人かの人達がテーブルを囲んで議論をしている．中央に議長がいて，黒板には会議名と議題が書かれている."

といった情景が想起される．そこで，「会議」のフレームを図5.16のように表現する．これは，「会議」というフレームが「会議名」，「場所」，「出席者」，「議題」，「議長」という属性（attribute, property）をもっていることを示している．

会議
会議名：教授会
場　所：第2会議室
出席者：池田，田中，鈴木，佐藤
議　題：カリキュラム編成
議　長：田中

図5.16 「会議」のフレーム

フレーム名
スロット名：値
スロット名：値
⋮

図5.17 フレームの基本構造

図5.17はフレームの基本構造を示し，属性を入れる場所をスロット（slot）と呼ぶ．フレーム表現の基本は，ある対象がもっているある属性はある値であるという知識を

（対象，属性，値）

という三つ組の形式で表現することである．たとえば，会議フレームの例では，

（会議，会議名，教授会）
（会議，場所，第2会議室）
⋮

のように分解できる．

5.5.2 階層構造と属性の継承

フレーム表現では，より抽象的な概念を表すフレームと，より具体的な概念

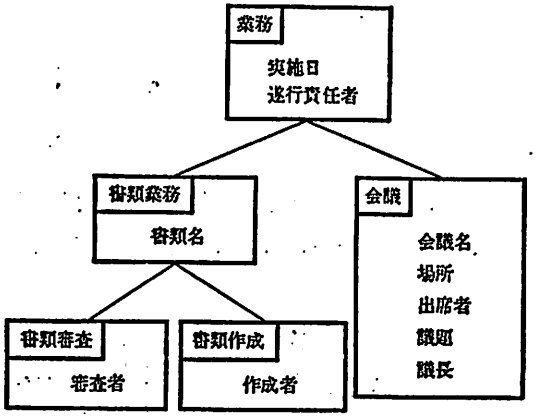


図 5.18　フレームの階層構造

を表すフレームの間で図5.18のような階層構造を構成できる．下位フレームは上位フレームをより具体化したものであり，この上下間のリンクをIS-Aリンクあるいは AKO (a-kind-of) リンクと呼ぶ．これは「会議　is a　業務」あるいは「会議　is a kind of　業務」を意味している．これにより，フレーム間の属性継承 (property inheritance) の機能を導入することができる．すなわち，下位フレームは上位フレームの属性を継承する．図5.18の例では，会議フレームは5個の固有の属性のほかに，業務フレームから継承した「実施日」と「遂行責任者」という属性ももっていることになる．

したがって，一つの概念を表すフレームから生成される具体的事例であるインスタンス (instance) は，そのフレームが上位フレームから継承した属性を，すべて有している．図5.19には会議フレームのインスタンスの例を示す．

インスタンス名：第3回教授会
親フレーム：会議
実　施　日：5月11日
遂行責任者：田中
会　議　名：教授会
場　　所：第2会議室
出　席　者：池田，田中，鈴木，佐藤
議　　題：カリキュラム編成
議　　長：田中

図 5.19　会議フレームのインスタンス例

5.5.3 拡張機能

フレームを用いた知識表現言語では，以上に述べた基本機能に加え，属性記述の拡張機能を設け，詳細かつ柔軟な知識表現を可能としている．すなわち，

フレーム名：会議
親フレーム名：業務
会　議　名：type (string)
時　　間：type (list of integer)
場　　所：type (string)
出　席　者：type (list of string), if-removed (出席者数変更)
欠　席　者：type (list of string), if-added (出席者リスト削除)
出席者数：type (integer), range(1..8), if-needed (出席者リストカウント)
議　　題：type (string)
議　　長：type (string), default (業務.遂行責任者)

図 5.20　会議フレームの定義

属性スロットには，以下のような種々の性質をもつファセット (facet) を記述できる．

① 値ファセット：スロットの値
② タイプファセット：スロットのデータ型
③ レンジファセット：スロットの値の範囲
④ デフォルトファセット：スロットの値が未定の時に用いるデフォルト値
⑤ デモンファセット：スロットアクセス時に呼び出される付加手続き

これらのファセットを用いた会議フレームの定義を，図5.20に示す．たとえば，出席者数スロットのタイプファセットとレンジファセットは，出席者数が整数であり，1から8の間の値をとることを示している．議長スロットのデフォルトファセットは，議長が未定の時は，会議フレームの上位の業務フレームから継承した遂行責任者スロットの値を，議長スロットの値とすることを示している．

デモンファセットで指定される付加手続きは，推論の過程で必要となる手続き型知識の記述を可能とするものである．この手続きは，一般の手続き型言語のようにプログラム内に手続き呼び出し文を明記することはせず，あるスロットがアクセスされた時に必要に応じて自動的に呼び出されるため，デモン (demon) と呼ばれる．代表的なデモンとして次のようなものがある．

① if-neededファセット：スロット値が必要になった時にその付加手続き

が実行され，スロット値が設定される．

② if-addedファセット：スロットに値が書き込まれた時に，その付加手続きが実行される．

③ if-removedファセット：スロットの値が削除された時にその付加手続きが実行される．

図5.20の例では，出席者数が必要となった時に，if-neededファセットで指定された出席者リストカウントの手続きが呼び出され，出席者スロットに書き込まれている人の数を数えて，出席者数スロットの値とする．欠席者スロットに欠席者名を追加した時にはif-addedファセットで指定された出席者リスト削除の手続きが呼び出され，出席者スロットから欠席者名を削除する．その時，出席者スロットのif-removedファセットで指定された出席者数変更の手続きが呼び出され，出席者数スロットの値から1を差し引く．

5.5.4 推論方式

フレームで表現された知識を利用した推論は，すでに述べた機能のうちの以下のものが用いられる．

① 属性の継承：会議の開催日は会議フレームの中では定義されていないが，上位の業務フレームから継承した実施日スロットがその代わりになると推論する．

② デフォルトファセット：議長が未定の時にその業務の遂行責任者が議長を兼任すると推論する．

③ デモンファセット：出席者数が未知でも出席者リストを数えればわかること，欠席者リストに追加した人は出席者リストから削除しなければならないこと，出席者リストを変更した時には出席者数も変更しなければならないこと，などを推論する．

これらの機能を実際にどのように組み合わせて実用システムを組み立てるかは，対象とする問題によって異なる．

5.5.5 多重継承

フレームの階層としては，下位のフレームは，親フレームを一つだけもつという木構造が基本であるが，複数の親フレームをもつのが自然な場合もある．たとえば，図5.21に示すように，議事録作成という業務は書類作成の一種と考えられるが，会議業務の一種とも考えられる．実際，議事録作成には会議フレームの各スロットの値が必要である．このように複数の親フレームとIS-Aリンクを結び，双方の属性を継承する機能を多重継承 (multiple inheritance) と呼ぶ．

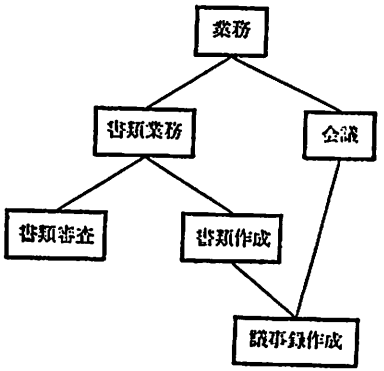


図 5.21　多重継承の例

5.6 意味ネットワーク

1968年にM. R. Quillianは言語理解のモデルとして意味ネットワーク (semantic network) を考案した．この技法はその後，前節で述べたフレームによる知識表現や，7章で述べる自然言語理解における意味表現に利用されてきた．

意味ネットワークの主な特徴は次のようなものである．

① 実世界の対象や概念の間の種々の関係を，2項関係を用いて自由に表現できる．

② この2項関係の集合をグラフ化したネットワーク上での探索と照合操作により，推論が実行できる．

以下では，これらの特徴を例を用いて具体的に説明する．

5.6.1 意味ネットワークの知識表現

図5.22に，桃太郎の昔話の一部分を意味ネットワークで表現した例を示す．丸で囲んだノードは，表現しようとする世界における対象，概念，状態などを表し，矢印で示すリンクはそれらの間の関係を示す．この例では，対象を表す

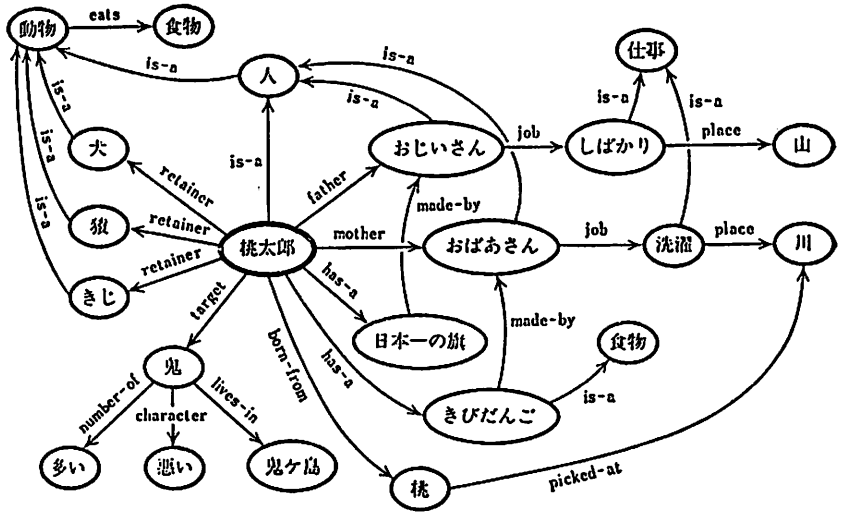


図 5.22　桃太郎に関する意味ネットワーク

ノードは「桃太郎」,「猿」,「きびだんご」などであり，概念を表すノードは「人」,「動物」,「食物」などである．状態を表すノードとしては「多い」と「悪い」がある．ノード間のリンクでは，「桃太郎は人である」というis-a関係，桃太郎はきびだんごをもっているというhas-a関係，桃太郎の家来は猿であるというretainer関係などがある．

5.6.2　基本構造

意味ネットワークの基本構造は，二つのノードとその間のリンクによって表される2項関係である．先に述べたリンクの例は，

is-a（桃太郎，人）
has-a（桃太郎，きびだんご）
retainer（桃太郎，猿）

という2項関係を表現している．この2項関係を

属性（対象，値）

とみなせば，これは5.5.1で述べたフレームの場合と同じように

（対象，属性，値）

という三つ組形式の表現が基本となっていることがわかる．

種々のリンクのうちのIS-Aリンクに着目すれば図5.23に示すような階層構造が表現されており，フレームの場合と同様に属性の継承機能をもつ．また，もう一つの基本的リンクであるhas-a関係はpart-of関係とも呼ばれ，全体と部分という階層構造を表現している．

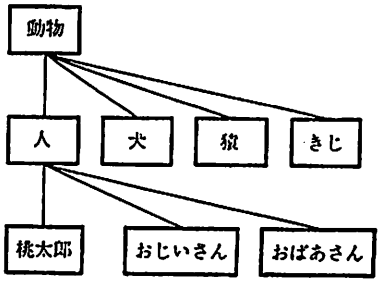


図 5.23　桃太郎のIS-A階層図

5.6.3　推論方式

意味ネットワークは，桃太郎の例からもわかるように，種々のノードやリンクを自由に導入して簡単に作成できる．その反面，この意味ネットワークを利用して推論を行うためには，個別に推論用の手続きを必要とする．

比較的簡単な推論方式としては，変数を含む2項関係が与えられて，意味ネットワークの中からそれに合う部分を探し出す方法である．たとえば，

eats（桃太郎，X）

すなわち，「桃太郎はなにを食べるか？」という質問に対し，IS-Aリンクの継承機能を利用して

is-a（桃太郎，人）
is-a（人，動物）
eats（動物，食物）

という探索を経て，「桃太郎は食物を食べる」という答を得ることができる．これは，フレームにおける属性の継承を利用した推論と同じである．

しかし「桃を拾ったのは誰？」という質問に対して，「桃が川で拾われた」という事実と「川に行くのはおばあさんだけ」という事実から，おばあさんという答を得るには，さらに別の知識が必要である．同様に「鬼を退治したのは誰？」という質問に，犬や猿の答を得るには「主人の敵は家来の敵」という知識がい

る．「きびだんごを食べたのは誰？」という質問に対しては

is-a（きびだんご，食物）

eats（動物，食物）

という関係から「動物とis-a関係にあるもの」という可能性が示せるが特定はできない．

5.7　オブジェクト指向言語

計算機で取扱う問題に関連した概念的対象物を，そのままプログラムの基本単位として記述できるオブジェクト指向言語（object-oriented language）は，従来の情報処理分野ばかりでなく，人工知能の分野の知識処理にも広く用いられている．

このオブジェクト指向言語は次のような特徴をもち，問題解決の計算モデルの自然な表現に適している．

① データとその操作手続きを一体化したオブジェクトを基本要素とし，外部からはその操作手続きを起動するメッセージを送れるが，データにはアクセスできない．

② オブジェクトはフレームと同様に階層化でき，上位オブジェクトから下位オブジェクトへの属性継承機能がある．

③ 同じ機能をもつオブジェクトを動的に複数個生成できる．

このような特徴的機能のもとになった基本概念は，1967年にシミュレーション言語として開発されたSimula 67に見られる．また，1970年代初めにMITのC. Hewittが提案したACTORモデルは，メッセージ送信に基づくオブジェクト指向型計算モデルの形式化とみなすことができる．さらに，1981年に発表されたSmalltalk-80（Xerox社）は，代表的なオブジェクト指向言語として利用され，その後の研究に大きな影響を与えた．

以下では，オブジェクト指向言語の一般的特徴を例を用いて具体的に説明する．

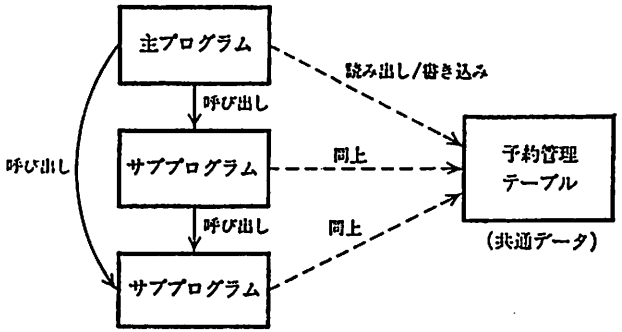


図 5.24　従来の手続き型プログラミングにおける共通データのアクセス方式

5.7.1　オブジェクトの基本機能

会議室などの予約管理をするプログラムの例を考えると，従来の手続き型言語を用いたプログラミングでは，図5.24に示すように，予約管理テーブルを共通データとして，プログラムの任意の場所からアクセス可能とする方式が一般的である．

これに対し，オブジェクト指向プログラミング（object-oriented programming）では，図5.25に示すように，データとその操作手続き群を一体化したオブジェクトとして定義する．これらの手続きはメソッド（method）と呼ばれ，外部から見たオブジェクトの機能は，このメソッドの集合で表現される．データそのものは外部からは隠ぺいされ，直接アクセスすることはできない．

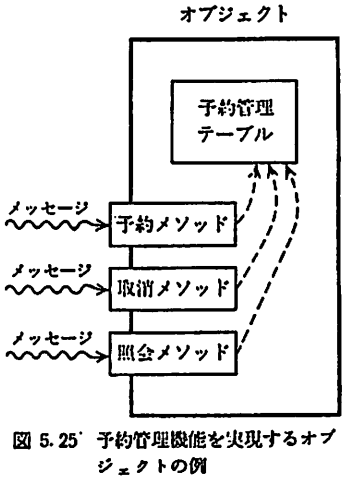


図 5.25　予約管理機能を実現するオブジェクトの例

このようなオブジェクトは独立性が強く，相互作用は相手オブジェクトのメソッドを起動するためのメッセージ（message）を送信することにより行う．その様子を示したのが図5.26の例である．これはある大学の機能の一部をモデル化している．初めに事務室オブジェクトの教授会開催準備メソッドが起動さ

れ，その実行中に，第2会議室とOHPプロジェクタを予約するメッセージが，各々の予約管理オブジェクトに送られる．各教官オブジェクトには会議案内メッセージが送られ，出欠通知メッセージが返ってくる．備品予約管理オブジェクトは，OHPプロジェクタの予約メソッド実行に伴い，その貸し出し先へ備品返却要求メッセージを送る．

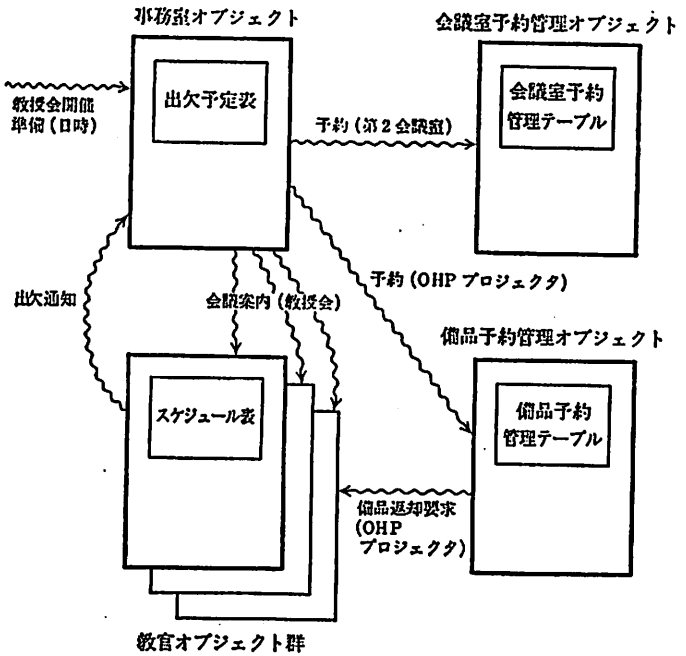


図 5.26　メッセージ送信によるオブジェクト指向プログラミング

このようなプログラミング方式では，オブジェクト内のデータの変更やメソッドの処理手順の変更が生じても，メソッドの機能が不変ならば，まわりのオブジェクトは変更しなくてすむため，プログラムの保守性が高くなる．

5.7.2　クラスとその階層

図5.27の会議室および備品の予約管理オブジェクトは，予約管理テーブルの内容が異なる以外は，図5.25のような同じ構造のオブジェクトと考えてよい．このようにメソッドが同じで，データの内容だけが異なるオブジェクトを効率良く作成するために，その型となるオブジェクトを用いて，そのインスタンスオブジェクトを複数個生成可能とする．この型となるものをクラスオブジェクトと呼ぶ．その様子を図5.27に示す．

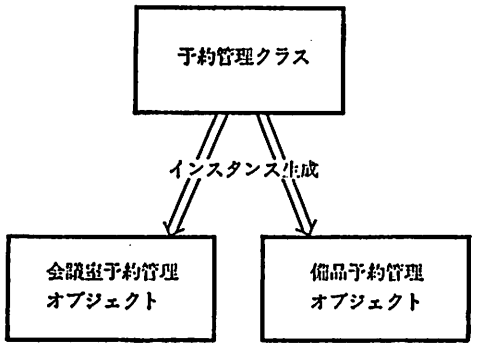


図 5.27　クラスオブジェクトからのインスタンスオブジェクトの生成

さらにオブジェクトの機能すなわちメソッドの集合が，少しずつ異なるものを効率良く作成するために，クラス間に階層構造を導入する．そして，下位のクラスは上位のクラスの機能，すなわちメソッドを継承できるものとする．図5.28に示す例において，優先予約管理クラスは親クラスから継承した予約，取消し，照会の各メソッドに加え，自分の内部で定義した優先予約メソッドをもつ．この時，予約管理テーブルも継承する．取消待ち予約管理クラスについても同様である．

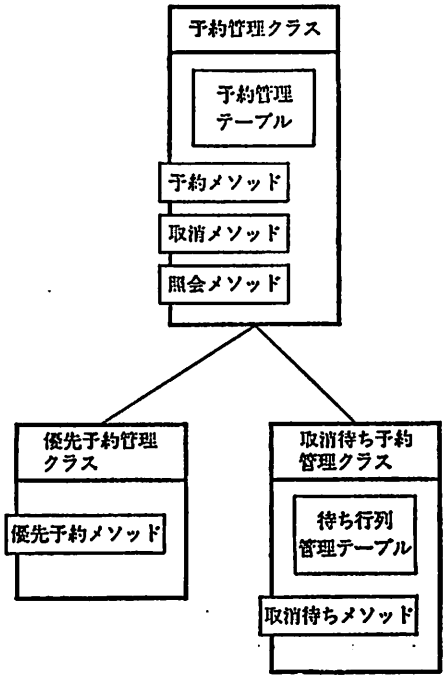


図 5.28　クラス階層

5.7.3　フレームとの比較

最初から知識表現方法として発展してきたフレームに比べ，オブジェクト指向言語はプログラミング技法として発展してきたものであるが，

結果的には類似点が多い．特にメソッドをフレームのスロットとみなせば，継承機能をもつ階層構造という基本的枠組みは同じになる．

一方，フレームがスロットへのアクセスに伴って，その付加手続きが起動される一種のデータ駆動型の計算モデルであるのに対し，オブジェクト指向プログラミングでは，メッセージ送信によって，メソッドが起動されるメッセージ駆動型の計算モデルである点が大きな違いといえる．

5.8 ハイブリッド型言語

人工知能システムで用いる知識は，実際には多種多様であり，本章でこれまで述べてきたような単一の知識表現形式のうち，一つだけですべてを記述するのは不適当な場合が多い．そこで，最近では，複数の表現形式を可能とする，ハイブリッド型の知識表現言語が用いられている．

5.8.1 マルチパラダイム

各々の知識表現形式は，問題解決のための知識を一つの型にあてはめてとらえる方法，すなわち，パラダイム（paradigm）を提供している．それらを記述言語としての特徴の面から分類すると次のようになる．

① プロダクションシステム：ノウハウ的知識の簡易な表現

② フレーム，オブジェクト指向言語：知識のモジュール化と階層化による構造的表現

③ LISP，Prolog：事実や規則などの宣言的知識と処理手順などの手続き的知識を統一的に扱う柔軟な表現

なお，意味ネットワークは表現形式がその適用対象に依存して決められることが多いので，単独で汎用化されるよりも，フレーム表現の拡張として実現されることが多い．

複数の知識表現形式，すなわち，マルチパラダイムを備えたハイブリッド型言語としては，上記①，②，③のうちの２種類を備えたものが多いが，すべてを含むものもある．中でも①と②を組み合わせたものが多く，その例としては，ルールとフレームを含んだART（Inference社）などがある．②と③を組み合わせたものとしては，オブジェクト指向概念を導入してPrologを拡張したESP（ICOT）や，LISPを基本としてオブジェクト指向プログラミングと，論理型プログラミングの機能を導入したTAO（NTT）がある．①，②，③のすべてを含むものとしては，LISPを基本としてルール，フレーム，オブジェクト指向の各機能を導入したKEE（IntelliCorp社），LOOPS（Xerox社）などがある．

5.8.2 ハイブリッド型知識表現

ハイブリッド型言語の設計においては，異なるパラダイムを融合して，単一言語としての統一的な言語仕様を設定することが重要であるが，多機能化の代償として言語仕様が複雑になりがちである．

この問題を回避するために，ハイブリッド型言語の一つであるS-LONLI[10]（日立）では，実世界の知識処理モデルの自然な表現に適するオブジェクト指向表現を，知識の全体構造の記述に用い，規則と事実の正確な表現に適する述語論理を，個別の知識の記述に用いることにより，両者の境界を明確にするような融合方式が採られている．さらに，個別の知識の記述には，述語論理のほかに，ノウハウ的知識を簡単に表現できるルール形式を導入することにより，上記①，②，③をすべて含んだハイブリッド型言語ES/X90[11]（日立）が開発されている．

その記述例を図5.29に示す．ここでは，団体というクラスの下に同好会というクラスが定義され，団体のスロットやメソッドを継承する．スロット定義では，リーダの名前が書き込まれた時に自動的に本部に変更届を提出するデモン（付加手続き）や，会員数の初期値，定員のデフォルト値が指定されている．

メソッドには述語メソッドとルールメソッドがあり，前者は，会員の入会や退会処理のような手続き記述に用いられている．send述語は，第１引数のオブジェクトへ第２引数のメソッドの実行依頼をする，メッセージ送信用の組込み述語である．入会の時は会員というクラスオブジェクトにインスタンスの生成

```
class 団体;
  inherit 組織;
  slots 団体名,
        リーダ (on-write (Lname, send (本部, 変更届 (Lname))),
        連絡先;
end.
```

```
class 囲碁会;
  inherit 団体;
  slots 会員数 (initial(0)),
        定員 (default(100));
  predicate-methods;
    入会 (Name):- send (会員, create(Name)),
                   X is #会員数 +1,
                   #会員数 :=X;
    退会 (Name):- send (Name, kill),
                  X is #会員数 -1,
                  #会員数 :=X;
  rule-methods;
    rules 会員数チェック;
      if #会員数=0 then write ('会員未登録です');
      if #会員数 >=#定員 then write ('満員です');
end.
```

図 5.29　ハイブリッド型知識表現の例 (ES/X 90)

を指示し，退会時にはそのインスタンスに消滅を指示するメッセージを送っている．ルールメソッドは，会員数チェックのように，特定の状態を認知して警告するための処理記述に用いられている．

参考文献

1) 上野晴樹，長谷川洋（編集）：特集：知識工学，情報処理，26，12，pp. 1456–1558 (昭 60-12)
2) 中所武司，増位庄一：知的プログラミング，計測と制御，25，4，pp. 368–373 (昭 61-4)
3) Steele Jr., G. L.: Common LISP, Digital Press, Mass. (1984)
   後藤英一（監訳）：Common LISP, bit 別冊，共立出版 (1985)
4) 角田博保（編集）：小特集：LISP の最近の動向，情報処理，26，7，pp. 710–749 (昭 60-7)
5) 正田郁雄，永田守男，山田眞市(編集)：特集：プログラミング言語 Prolog，情報処理，25，12，pp. 1312–1410 (昭 59-12)
6) Clocksin, W. F. and Mellish, C. S.: Programming in Prolog, Springer-Verlag, Berlin (1981)
   中村克彦（訳）：Prolog プログラミング，マイクロソフトウェア (1983)
7) 小林重信：知識工学の基礎と応用—知識の表現と利用，計測と制御，24，2-3，pp. 155–164，242–250 (昭 60-2, 3)
8) 斎藤正男，溝口文雄：知的情報処理の設計，コロナ社 (1982)
9) 鈴木則久（編集）：オブジェクト指向，共立出版 (1985)
10) Chusho, T. and Haga, H.: A multilingual modular programming system for describing knowledge information processing systems, IFIP 10th World Computer Congress, pp. 903–908 (1986)
11) 中所武司ほか：エキスパートシステム構築ツール ES/X 90，情報処理学会第 35 回全国大会論文集，pp. 1733–1750 (1987)

6 知識工学

6.1 知識工学の概要

人工知能の研究は，後にLISPを設計するJ. McCarthy，フレーム理論を提唱するM. Minsky，プロダクションシステムを開発するA. Newellらが1956年にダートマス大学に集まり，コンピュータを用いた知能の実現に関する研究を，人工知能と名づけた頃から活発化した．しばらくは人工知能実現のための一般的原理の研究が中心となり，多くの基礎技術が開発されたが，実用化には至らなかった．

この反省から，人間の問題解決能力の基本となっている経験的な知識に着目し，問題領域に固有の知識を利用する応用人工知能の研究が，1970年前後から開始された．この分野の先駆者で，1965年に化学物質の構造式を推定する最初のエキスパートシステムDENDRALの開発を始めた，スタンフォード大学のE. Feigenbaumは，1977年の第5回人工知能国際会議（5th IJCAI, Cambridge, 1977）の招待講演でこの分野の研究を知識工学（knowledge engineering）と名づけた．

知識工学の大きな特徴は，従来のコンピュータ応用技術では解決が難しい悪構造問題（ill-structured problem）をあつかえることである．すなわち，対象が数値データではなく記号データで表現され，解法のアルゴリズムが定式化できない問題をその分野の知識に基づいて解くことができる．このような知識工

学を応用したシステムを一般に知識ベースシステム(knowledge-based system)と呼ぶ．特に特定分野の専門家のもつ専門的知識を利用したシステムは，エキスパートシステム（expert system）と呼ばれる．

本章では，知識工学応用の代表であるエキスパートシステム，およびその構築ツールを中心に述べる．

6.2 エキスパートシステム

6.2.1 基本構成

エキスパートシステムは，対象とする問題領域に関して，その専門家がもっている専門知識を利用して推論を行い，専門家と同等の問題解決を行うシステムである．その基本構成を図6.1に示す．専門家は知識獲得インタフェースを通して，あらかじめ知識ベースに知識を入力しておく．利用者は，従来のように専門家と直接対話をする代わりに，知識利用インタフェースを通して問題の解決を依頼する．

エキスパートシステムの開発においては，主に次のような技術課題がある．

① 知識表現：知識を，どのような形式で知識ベースに保存しておくか．

② 知識利用：知識を問題解決にうまく利用するために，どのような推論機構を設けておくか．

③ 知識獲得：知識を専門家からどのように抽出して知識ベースに入力するか．

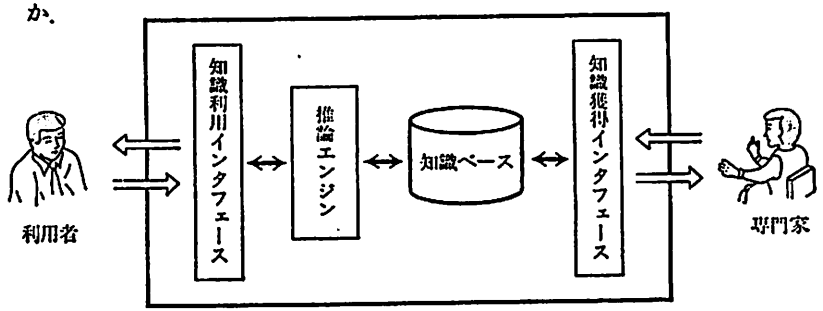


図 6.1 エキスパートシステムの基本構成

6.2.2 エキスパートシステムの例

ここでは先駆的システムで，それぞれ独自の特徴をもつ DENDRAL，MYCIN，HEARSAY-Ⅱについて説明する．

(1) DENDRAL

DENDRALは，知識工学の提唱者であるE. Feigenbaumらが開発したエキスパートシステムで，質量分析器から得られるスペクトルデータを解析して，有機化合物の分子構造を推定する．その具体的方法は，次のような計画，生成，検査の3段階で構成されている．

① 計画：プロダクションルールを用いた前向き推論により，スペクトルデータから，未知の化学物質がとり得る分子構造に関する制約条件を決定する．

② 生成：この制約条件を満たす分子構造をすべて生成する．

③ 検査：生成された分子構造をもつ化学物質が，実測された①のスペクトルデータと合致するか否かを検査する．

このようにDENDRALでは，計画段階において，ルールの前向き推論を探索空間の絞り込みに利用している．このシステムの成功をきっかけに，各種のエキスパートシステムの開発が活発化された．

(2) MYCIN

MYCINは，1973年頃からE. Shortliffeらによって開発された医療診断エキスパートシステムで，髄膜炎と菌血症の感染症の種類について診断し，投薬を指示するものである．このシステムの目的は，感染症の患者を担当している臨床医に対して，感染症の専門医と同レベルの助言をすることであった．

MYCINは，患者の症状や検査結果などについて，担当医に質問しながら診断を行う対話型システムであり，次のような特徴的機能がある．

① 後向き推論：診断結果の候補となる感染症の種類を順々にゴールとして設定し，診断用のプロダクションルールを用いて，そのゴールが満足されるか否かを後向き推論で確かめる．

② 確信度：不確かな知識を扱うために，事実やルールに対して1から−1の間の確信度をつける．1は完全な肯定，−1は完全な否定を示す．得られた推論結果の確信度は，これらの事実やルールの確信度を用いて計算される．

③ 説明機能：担当医はシステムからの質問に対し，"why" を入力することにより，その質問の答が現在の推論にどのように利用されるかを知ることができる．またシステムが出した結論に対し，"how" を入力するとその結論に至った過程を説明してくれる．

これらの機能は，以後のエキスパートシステムに大きな影響を与えたが，MYCIN自身は処理時間等の問題により実用にはならなかった．

(3) HEARSAY-Ⅱ

HEARSAY-Ⅱは，1960年代後半から1975年にかけてカーネギーメロン大学で開発された音声理解システムである．ここでは発声音の認識に加えて，文脈に関する規則や文法的な規則を用いた単語認識や文章認識を行うために，大規模かつ多様な知識集合が，お互いに協力しながら問題解決を行う黒板モデル（ブラックボードモデル，blackboard model）が導入された．その特徴は次のようなものである．

① 複数の知識源：音声理解のために必要な部分的処理に用いられる知識の集合を，知識源としてまとめる．たとえば，音節の候補から単語を生成するものや，単語列の候補を解析して句を生成するものなどである．

② 黒板：複数の知識源の間での情報の受け渡しは，この黒板にメッセージを書き込むことによって行う．この黒板は，音声の解析単位である入力信号，音節，単語，単語の並び，句，文などに対応してレベル分けされている．

③ 黒板モニタ：これは黒板上のメッセージを監視し，適用可能と思われる知識源の待ち行列に登録する．

④ スケジューラ：これは待ち行列の中からその時点で最適と思われるものを選択し，その知識源を起動する．

このような黒板モデルは，大規模で複雑な問題に対する協調型問題解決手法と

して，以後のエキスパートシステムにも用いられている．

6.3 エキスパートシステム構築ツール

最初のエキスパートシステムの開発には，およそ15人年の工数がかかっているが，これは種々の試行錯誤の中から，最適な推論機構や知識表現形式を見出すまでに要した労力である．その後の多くのエキスパートシステムの研究開発の経験を通して，現在ではエキスパートシステム構築のための汎用ツールが，数多く商品化されるまでに至っている．

エキスパートシステム構築ツールとは，図6.1のエキスパートシステムの基本構成において，問題領域に固有の知識ベース以外を共通化したものである．したがってエキスパートシステム開発者は，対象とする専門分野の知識ベースだけを構築すればよいので，開発工数が少なくてすむことになる．

6.3.1 ツールの基本機能

エキスパートシステム構築ツールは，既存のエキスパートシステムから発展したものが多い．たとえば，MYCINからEMYCINというツールが作られた．このようなツールは，その母体となったエキスパートシステムの影響が強く，適用対象が限定される．EMYCINの場合はMYCINと同様の診断システムの開発に適している．

一方，最初から汎用ツールとして開発されたものは，広範囲への適用をねらっているため，知識表現や推論機能の豊富なものが多い．知識表現では5.8で述べたハイブリッド型言語が多い．推論機能も前向き推論や後向き推論に加えて，両者を組み合わせた双方向推論や，Fuzzy集合の考え方を用いたあいまい推論が可能なものもある．

6.3.2 ツールの選択

ツールの種類は多いが，単に機能が豊富なほど良いというものではない．まず開発するエキスパートシステムに関して以下の項目を考慮する．

① 利用者：エンドユーザ，専門家自身など．
② 利用目的：診断，計画，設計など．
③ 利用環境：パソコン，ミニコン，大型計算機など．
④ 規模：ルール数

次に開発方法について以下の考慮をする．

① 開発者：専門家，ナレッジエンジニア，プログラマなどの有無と人数．
② 開発環境：利用環境と同じか否か．
③ 保守体制：開発後の知識の変更の有無とその方法．

以上のような観点から，各々のツールが提供している機能の必要性を考慮して選択すればよい．

6.3.3 ハイブリッド型ツール

ここでは，5.8で述べたハイブリッド型知識表現の中で，最も典型的なフレーム表現とルール表現を組み合わせたツールについて，その1例であるエキスパートシステム構築ツールES/KERNEL[4]（日立）を用いて説明する．ただし，詳細な機能には触れず，フレームおよびルールを用いて前向き推論が行われる様子を，患者への投薬診断の例で説明する．

（1） フレーム

推論の対象となる患者に関する知識は，フレームとして表現する．図6.2は，大和太郎というフレームが，患者というクラスフレームのインスタンスフレームであることを示している．大和太郎のフレームは，患者のフレームがもつ体温，湿疹などのスロットを継承する．

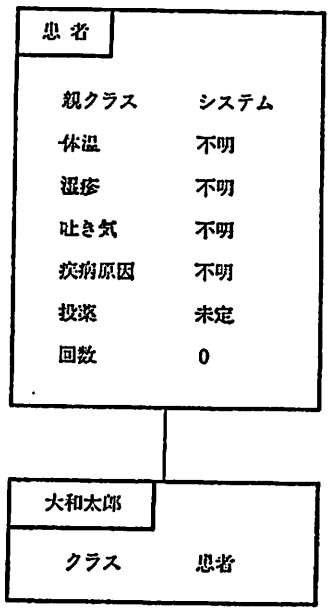
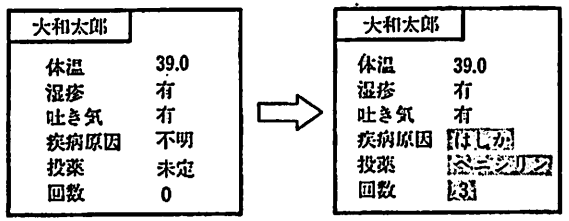


図 6.2 推論対象のフレーム表現



（a） 推論前のフレーム　　（b） 推論後のフレーム

```
(ルール1
 もし (?患者 の @疾病原因 が 不明 であり
             @体温 が 38度 以上であり
             @吐き気 が 有 であり
             @湿疹 が 有 である)
 ならば (send ?患者 assign (疾病原因, はしか))
)
(ルール2
 もし (?患者 の @投薬 が 未定 であり
             @疾病原因 が はしか である)
 ならば (send ?患者 assign (投薬, ペニシリン)
             assign (回数, 3))
)
```

（c） 推論に用いたルール

図 6.3　ルールとフレームを用いた前向き推論

（2） 前向き推論

今，患者の症状からはしかの投薬をするために，次のような二つのルールがあるとする．

① ルール1：もし，患者の疾病原因がわからず，体温が38度以上であり，吐き気があり，さらに湿疹があれば，患者ははしかである．

② ルール2：もし，患者の投薬が未定で，疾病原因がはしかであれば，ペニシリンを1日3回投薬する．

これらのルールはES/KERNELでは図6.3（c）のように記述する．?で始まる単語は変数，@で始まる単語はスロット名を表す．

図6.2の患者フレームのスロットの値のままでは，これらのルールの条件部は成立しないが，大和太郎に対する問診により，体温，湿疹，吐き気に関し，図6.3（a）に示すようなスロット値が彼のフレームに記入されたとする．この時，ルール1の条件部の変数「?患者」に大和太郎というフレーム名が代入され，そのフレームの四つのスロットの値が照合され，条件が成立することがわかる．そこで，そのルール1の実行部が実行され，大和太郎のフレームにメッセージが送られ，疾病原因スロットにはしかが代入される．

その結果，次にルール2の条件部が満足され，同様に投薬スロットにペニシリン，回数スロットに3が代入される．

以上に述べたように，フレームとルールを用いて前向き推論を行う診断システムを簡単に作成できる．

6.4 応 用 例

6.4.1 適 用 分 野

エキスパートシステムの適用分野は広範囲に及ぶ．当初は医学をはじめとする大学関係の研究分野が多かったが，最近では産業分野，ビジネス分野への適用事例が急増している．これらのエキスパートシステムを機能面から分類すると，およそ次のようになる．

① データ解釈・診断

② 監視・制御

③ 計画・設計

各々の特徴を以下に述べる．

（1） データ解釈・診断

データの解釈とは，測定器やセンサによる測定データを解析し，システムの状態を推定するものである．この事例としては，6.2で述べたスペクトルを解析するDENDRALや音声信号を解析するHEARSAY-Ⅱのほか，金属鉱床を

探査するPROSPECTOR (SRI)，油田を探査するDipmeter Advisor (Shulumberger社) などがある．

診断は，システムの異常現象が発生した時に観測されたデータから，因果関係の知識を用いてその原因を推定するものである．事例としては6.2で述べた感染症診断のMYCINなどの医療診断のほかに，機械，コンピュータ，電気回路などの故障診断，プラントや電話網などの異常診断のシステムがある．

（2） 監視・制御

これはシステムの状態を常に監視しながら，適切な状態を保つようにシステムを制御するものである．鉄鋼，電力のプラント制御，原子炉運転制御，列車運行管理，大型コンピュータ運用管理などのシステムがある．

（3） 計画・設計

これは与えられた目標を満たす無数の解の中から，各種の制約条件を満たす，より良い解を求めるものである．このような合成問題は人間の知的創造活動に対応するため，本質的な難しさがある．事例としては，コンピュータシステムの機器構成設計を行うR1のほか，LSIの回路設計や各種レイアウトを行うシステムがある．特にカーネギーメロン大学のR1はXCONという名前で，すでにDEC社で実用化されており，数千のプロダクションルールをもつ，代表的な大規模実用エキスパートシステムである．

6.4.2 実用システム

エキスパートシステムの実用例として，計算機室機器レイアウトエキスパートシステム[6]（日立）を紹介する．

このシステムは，従来はシステムエンジニアが行っていた業務を肩代りするもので，計算機室の形状と設置すべき機器が与えられた時，以下のような制約条件を満足するレイアウト図を作成する．

[制約条件の例]

① 機器はお互いに重ならない．

② 機器のまわりに保守エリア（保守のために機器の扉を開けて作業する領域）を確保する．

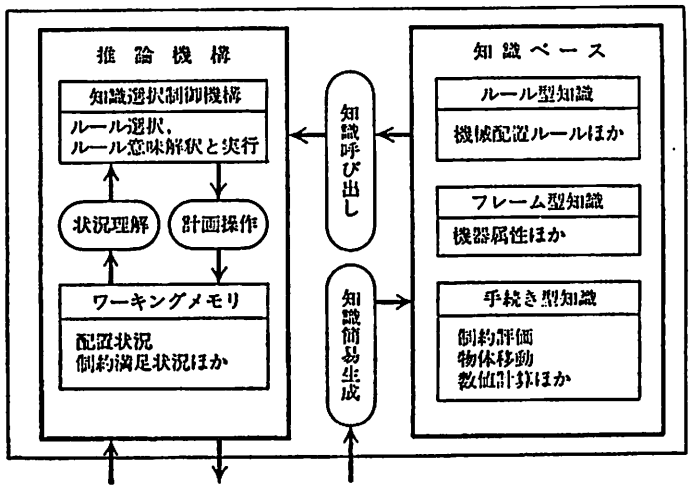


図 6.4 計算機室機器レイアウトエキスパートシステムの構成

③ 機器の前面がコンソールディスプレイのところから見える．

④ 入出力機器は出入口の近くに配置する．

⑤ 磁気ディスク装置は部屋の奥に配置する．

⑥ 横並びの機器の前面をそろえる．

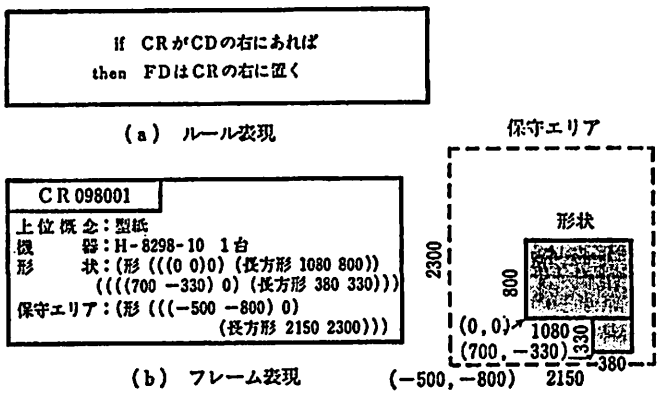
全体のシステム構成は，図6.4に示すような知識ベースと推論機構からなる．知識ベースには，次のような3種類の知識表現形式の知識を格納している．各各の例を図6.5に示す．

① ルール型知識：配置候補地に関する定石的知識や作業手順．図6.5（a）の「CR（カード読取機）がCD（コンソールディスプレイ）の右にあれば，FD（フロッピーディスク装置）はCRの右に置く」という規則は，CDを操作するオペレータにとってCRやFDを使いやすくするための定石的知識である．

② フレーム型知識：配置状況や機器の属性に関する知識．図6.5（b）のCR098001という名称のフレームは，親フレームが型紙であり，H-8298-10という名称のCRの型紙を意味し，形状および保守エリアは図に示したも

のになっていることを表現している.

③ 手続き型知識：重なりなどの制約条件のチェックや，物体移動および数値計算を実行する手続き．図6.5(c)の例では，二つの長方形AとBの重なった部分Cを計算している.



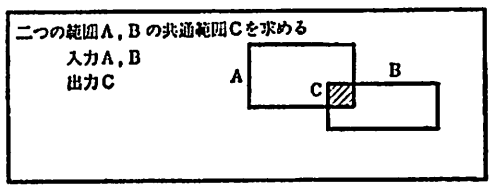


図 6.5　3種類の知識表現

一方，推論機構は，機器配置状況や制約満足状況を記憶するワーキングメモリと，ルールの選択・意味解釈・実行を行う知識選択制御機構からなり，次のような問題解決の手順がとられる.

① 現時点での配置状況を理解して，次に実行すべきルールを選択する.

② 選択したルールの意味を解釈し，配置候補地の範囲を決定する.

③ その範囲内で制約を満足する配置位置と，機器の方向を決める.

④ 配置不可能な場合は，配置済みの機能をずらして，再び③を試みる.

このようにして作成したレイアウト図の例を図6.6に示す.

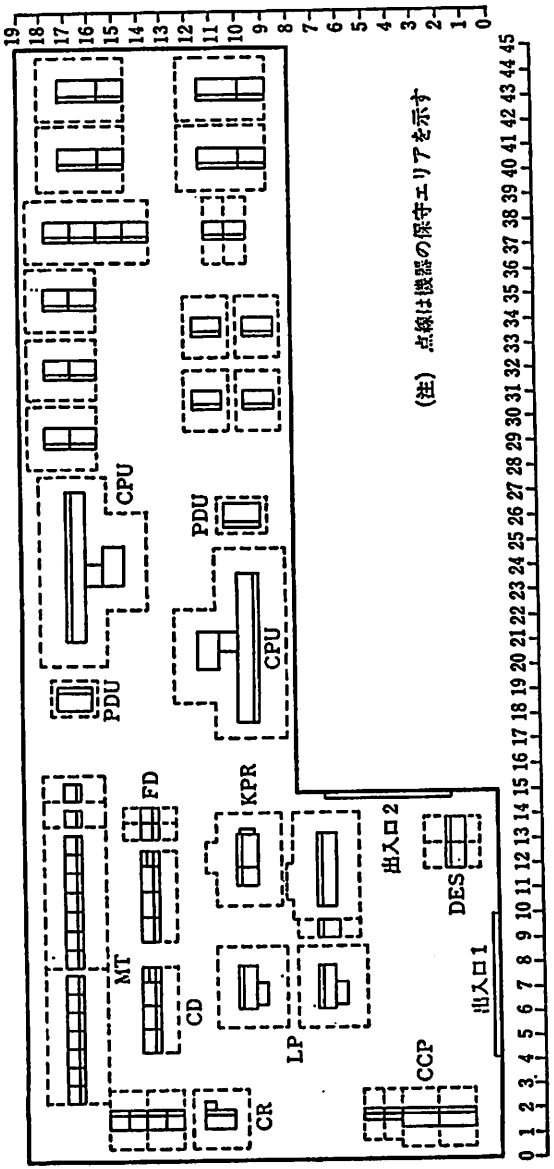


図 6.6　計算機室機器レイアウトの出力例

本システムの規模は，ルール数400，フレーム数は固定的なものが100，実行途中で生成されるものが300，手続き数1 000程度のものであり，従来システムエンジニアが5～8時間かけていた作業を，30分程度へと短縮している．

参考文献

1) 上野晴樹：特集：エキスパート・システム，情報処理，28，2，pp. 146-236 (昭62-2)
2) Harmon, P. and King, D.: Expert systems, John Wiley & Sons (1985) 阪訪基（監訳）：エキスパートシステムズ，サイエンス社 (1986)
3) 小林重信：知識工学の基礎と応用-エキスパートシステム，計測と制御，24，8-9，pp. 730-738，833-840 (昭60-8，9)
4) 金森喜正ほか：ES/KERNELでの知識表現方法と高速推論方式，日立評論，69，3，pp. 219-224 (昭62-3)
5) 佐々木浩二，鳩田晋：知識工学の産業への応用，人工知能学会誌，1，1，pp. 64-71 (昭61-9)
6) 渡辺俊典ほか：知識工学の計算機室機器レイアウトCADへの応用，日立評論，67，12，pp. 967-970 (昭60-12)

7 自然言語理解と機械翻訳

7.1 基本方式

人間が日常用いている言語は，計算機言語のような人工言語に対して自然言語と呼ばれている．自然言語を計算機に理解させようとする自然言語理解の研究は，とくに複数個の自然言語間で相互変換を行うための基本技術として重要である．言語の意味を理解しなければ，正しい変換は困難であるからである．この自然言語間の相互変換技術が機械翻訳と呼ばれている．本章では，この自然言語理解を含む機械翻訳の問題について述べる．

7.1.1 機械翻訳の歴史

機械翻訳の考えは，最初のコンピュータが作られた1945年よりも前にすでにあったが，実際の研究が始まったのは，その翌年の1946年からである．その後，1952年には米国で第1回機械翻訳会議，1956年には国際会議が開かれるなど，活発な研究が行われたが，期待された成果が出ないまま，1966年には近い将来に実用システムはできないという報告書が出された．

しかし，その後の計算言語学の分野での着実な技術の蓄積と最近の知識処理技術の進歩により，対象分野を限定すれば実用化は可能であると考えられるようになった．

一方，経済や科学技術の分野の国際化とともに，機械翻訳システムへの期待

が急速に高まってきた．特に日本では，日本語が国際的に通用しないことから，翻訳を必要とする文書が多く，機械翻訳システムとそれに関連した自然言語理解の研究開発が活発である．

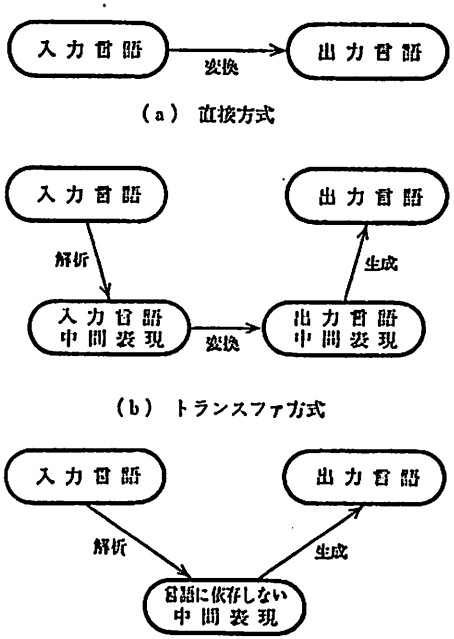


図 7.1　機械翻訳の基本方式

7.1.2　機械翻訳の基本方式

機械翻訳とは，入力言語の文を出力言語の文に変換することであり，図7.1に示すような方式がある．このうち変換過程で中間表現を介しない直接方式は，効率は良いが複雑な文章の変換には適さないため，適用範囲が限られる．

一般の機械翻訳では，文の構造や意味を形式的に表現する中間表現を介して変換を行う．この方式には，中間表現の設定の仕方により，トランスファ方式とピボット方式がある．トランスファ方式は図7.1(b)に示すように，入力言語の中間表現と出力言語の中間表現を別々に設定し，入力言語を解析した後，これらの中間表現間で変換を行い，最後に出力言語の生成を行う．

一方，ピボット方式は図7.1(c)に示すように，入力言語と出力言語に共通の中間表現を設定するもので，変換過程がない．どの言語にも依存しない中間表現を設定することは難しいため，厳密な意味でのピボット方式の実現は難しいが，入力言語と出力言語の両方に依存する変換の部分を，できるだけ小さくするという考えは重要である．

7.2　文　解　析

入力文から適切な訳文を生成するためには，その入力文の構文，意味，さらには文脈の解析処理が重要である．このように入力文を解析し，意味を理解する過程は，とくに自然言語理解と呼ばれており，機械翻訳や人間が機械に指令を与えるための，知的インタフェースの基本技術として重要である．

この自然言語理解のための要素技術のうち，構文については言語学的にもかなり解明されてきているが，意味や文脈については，計算機で形式的に扱う技術は未だ不十分である．本節では構文解析について述べ，次節で意味処理について述べる．

7.2.1　形態素解析

入力文はまず初めに形態素に分割する．形態素 (morpheme) とは，単語や接辞などのように意味のある最小単位の構文構成要素である．たとえば，「計算機科学と言語学の発展により機械翻訳への道がひらけた．」という入力文は形態素解析 (morphological analysis) により，次のような形態素の列になる．

計算機／科学／と／言語学／の／発展／に／より／機械／翻訳／へ／の／道／が／ひらけ／た／．

特に日本語の場合は次のような特徴があり，形態素解析は英語の場合よりも一段と難しい．

① 単語と単語の間に空白がない．

② 送りがな，句読点の用法に自由度が大きい．

③ 接頭辞，接尾辞が多いうえに，他の語との結合規則が明確でない．

したがって日本語の形態素解析では，単語の境界を認識するのに字種情報を利用したり，品詞の接続可否を判断しながら辞書との対応をとっていく．

7.2.2 構文解析

構文解析（parsing, syntactic analysis）では，形態素解析で得られた品詞情報などを用いて，文全体の構文構造を決定する．この時に特定の文法理論と結びついた構文アルゴリズムが用いられるが，実用的なシステムでは，処理速度が効率的な文脈自由文法や格文法が用いられる．

（1） 文脈自由文法

文脈自由文法（context free grammar）はN. Chomskyが提案した生成文法理論に合うもので，主語，述語，目的語などの語順の制約が厳しい英語などの解析に適している．その例を図7.2に示す．同図(a)は文法規則を表す．$\alpha \longrightarrow \beta$は，左辺$\alpha$は右辺$\beta$で置き換えてよいことを意味し，「|」は「または」を意味する．文，名詞などは構文規則の表現のために，便宜的に導入されたものであり，非終端記号（nonterminal symbol）と呼ばれる．John, runsなどは実際の文に用いられるものであり，終端記号（terminal symbol）と呼ばれる．文脈自由文法とは，左辺αが一つの非終端記号に限られた文法のことである．

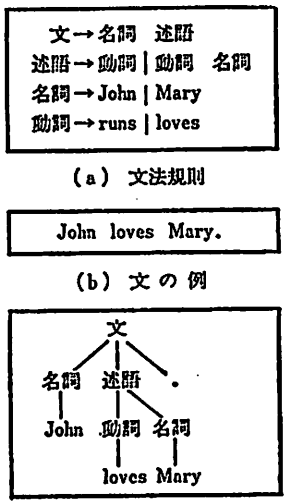


図 7.2　文脈自由文法による構文解析

非終端記号の文から始めて，これらの文法規則を用いて左辺を右辺で置き換えることを繰返し，最終的に非終端記号を含まない終端記号だけの列を得ることができるが，このようにして得られた文字列だけが，この文法に合った正しい文である．この例では図7.2(b)の文を含め，12種類の文が生成される．

したがって文脈自由文法の構文解析とは，与えられた入力文が文法規則から生成可能であることを示すことである．その過程で文法規則の適用順が決定され，最終的には図7.2(c)に示すような構文解析木が得られる．

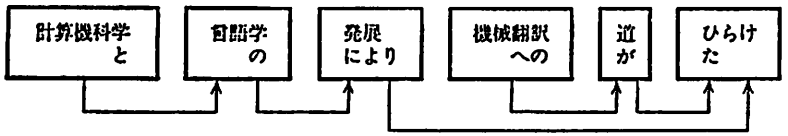


図 7.3　文節の係り受け構造

なお，この方式は自然言語を対象とした時に文法規則が膨大になるとか，意味規則が扱えないなどの問題があるため，実際の適用時には，この欠点を補うための拡張が行われている．

（2） 係り受け規則

日本語は英語と異なり，文末が用言で終る以外は主語，目的語などの語順が自由であるという大きな特徴があり，文脈自由文法は適さない．そこで日本語の構文規則としては，文節間の修飾関係に注目した係り受け規則が用いられる．文節とは名詞，動詞，形容詞などの内容語とそれに続く助詞，助動詞などの機能語をひとまとめにしたものである．先に形態素解析に用いた文例での文節の係り受け構造の例を図7.3に示す．

このような文節間の係り受け構造の解析には，次のような規則を用いる．

① 文末以外の文節は必ず後方の文節に係る．

② 係り文節から受け文節に矢印をひいた時，矢印が交差することはない．

③ 一つの文節が，同じ意味的役割の文節を複数受けることはない．

これら3条件は簡単ではあるが，係り受け解析の有力な手段となっている．この方法で係り受け構造が幾種類か考えられる場合は，意味的な制約や統計に基づいた発見的（ヒューリスティック，heuristic）な規則により，正解の可能性の高いものを選定する．

7.3 意味表現

前節で述べた構文解析では，文を構成する各々の語がもつ意味を扱ってはいないので，出力言語への変換は未だできない．たとえば，次の二つの文は，構文は同じだが太字の部分の意味が異なる．

「太郎が　クレヨンで　絵を　かいた.」

「太郎が　　教室で　恥を　かいた.」

最初の「で」は手段を表し，後の「で」は場所を表すため，英語に翻訳する時には異なる前置詞を用いなければならない．また，「かいた」については「書いた」，「画いた」「掻いた」「欠いた」などのいずれにあたるかを決定しなければならない．

このような意味解析（semantic analysis）を行った結果として得られる意味構造の表現形式は，いろいろ考えられているが，この表現形式は，基本的には知識表現形式とみることができる．ここでは，5章で述べた知識表現形式と関連の深い意味表現形式を中心に説明する．

7.3.1　意味ネットワーク表現

5.6で述べた意味ネットワークを用いた意味表現の例を図7.4に示す．ノードは太郎，クレヨンなどの単語のほかに，意味表現のために解析処理の過程で導入された文ノード記号や，過去形のような情報を表す．ノード間のリンクは，その文の中で各々のノードがもっている意味的役割を表している．

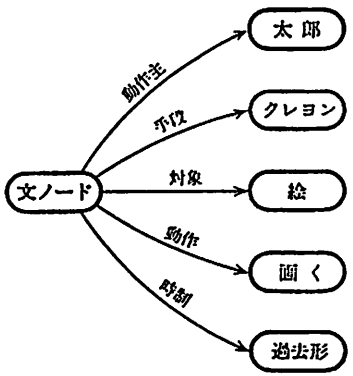


図 7.4　意味ネットワークによる意味表現

たとえば，「クレヨンで」という表現は意味解析の結果，クレヨンというノードへ手段リンクを結んだ形になっている．これが「教室で」の場合は教室というノードへ場所リンクを結ぶことにより，「で」の意味を正確に表現する．

7.3.2　フレーム表現

同じ例を5.5のフレーム形式で表現した場合は，図7.5のようになる．すでに述べたように，フレームも意味ネットワークも（対象，属性，値）の三つ組

表現が基本となっているため，図7.4と図7.5は単純な対応関係になっている．すなわち，意味ネットワーク表現における文ノードをフレームとし，文ノードからの各リンクの属性である，動作主，手段，対象，動作，時制などをそのフレームのスロットに対応づける．各属性リンクの先のノードがフレームにおける対応するスロットの値となっている．

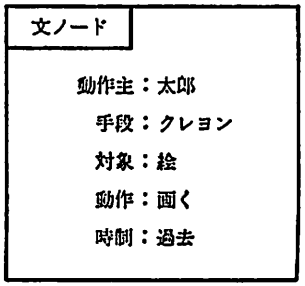


図 7.5　フレームによる意味表現

7.3.3　格フレーム表現

フレームと似た意味表現形式として，図7.6に示すような格フレーム（case frame）表現がある．これは，動詞を中心に意味を解析していくものであり，格文法（case grammar）で利用されている．

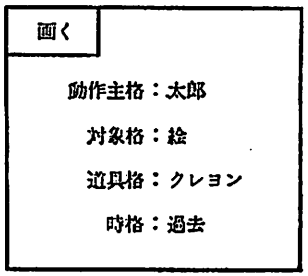


図 7.6　格フレームによる意味表現

この基本的な考えは，文の意味理解においては，動詞が重要な役割を果たしており，動詞を中心に，その文の意味構造が決まるというものである．たとえば「画く」という動詞を与えられると，我々は「誰が」「なにを用いて」「なにを」画くという概略の意味構造を想像することができる．

ここでの格とは主格，目的格などの構文的な表層格（surface case）ではなく，文の中での意味的な役割を果たす深層格（deep case）のことであり，動作主格，対象格，目標格，源泉格，理由格，道具格，場所格，時格などがある．このほかにも多数の格が考えられる．格には各々の動詞によって，必ず文の中になくてはならない必須格と省略されていてもよい任意格がある．

このような意味表現形式には，次のような特徴がある．

① 構文が同じで意味の異なるものが適確に表現できる．たとえば「クレヨンで」と「教室で」は道具格と場所格に区別される．

② 構文は異なるが，意味が同じものを同じ表現形式にできる．たとえば，

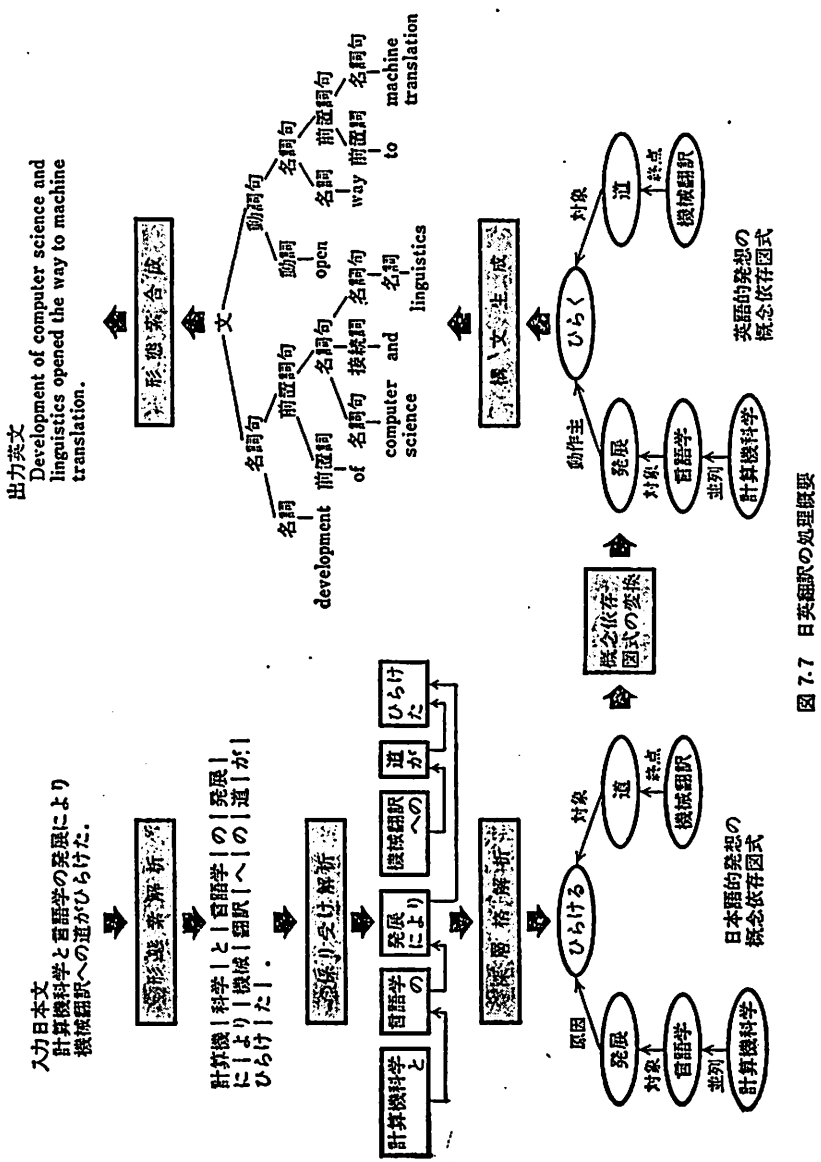


図 7.7　日英翻訳の処理概要

「絵は太郎によってクレヨンでかかれた.」という受身の文章や「クレヨンで太郎が絵をかいた.」という語順の異なる文章もすべて同一表現になる.

7.4　機械翻訳システム

本章では，これまで技術が比較的進んでいる解析処理を中心に，自然言語理解について述べてきた．しかし，文の変換，生成処理については，まだ主要な技術が確立しているとはいえない状況にある．そこで，変換，生成技術の一般論の説明の代わりに，具体的な機械翻訳システムの例を述べる．

7.4.1　日 英 翻 訳

日英機械翻訳システムの例として，HICATS/JE[9]（日立）を紹介する．以下，図7.7の翻訳例に沿って説明する．

（1）　日本文の解析処理

まず入力日本文に対して，7.2で述べた形態素解析と係り受け解析を行う．さらに7.3で述べた深層格解析を行い，概念依存図式と呼ぶ意味表現形式で中間表現をする．この概念依存図式は，格構造表現を拡張したものである．

（2）　概念依存図式の変換

入力日本文を解析して得られた概念依存図式を，英語的発想の概念依存図式に変換する．日本語の発想と英語の発想が同じ場合は，変換処理は不要になる．この例では受動表現を能動表現に変換している．

（3）　英文の生成

次に英語的発想の概念依存図式の各ノードに訳語を割り当てながら，英文の句構造を生成する．句構造（phrase structure）は単語から句，さらに句から文を構成し，文の構成要素のまとまりを階層的に表現するものである．文形や訳語の品詞の決定は，英語の文法に合致する範囲内で文の簡潔さを考慮しながら行う．

最後に句構造の終端ノードの単語を一列に並べるとともに，ノードの属性と

入　力　日　本　文	出　力　英　文
6.2.4　データベース作成の準備 必要なデータを蓄積し，データベースを作成する前に，ファイルを初期設定したり，物理ファイルや論理ファイルを定義したりしなければならない．	6.2.4　Preparation for data base generation Before accumulating necessary data and generating a data base, files must be initialized, and physical files and logical files must be defined.

図 7.8　日英機械翻訳システムによる計算機マニュアルの翻訳例

して表現されている時制や，数に応じた語形変化を施すことにより，訳文が得られる．図7.8に，本システムを用いてデータベース管理システムのマニュアルを翻訳した例を示す．

7.4.2　英 日 翻 訳

英日機械翻訳システムの例として，HICATS/EJ[4]（日立）を紹介する．以下，図7.9の翻訳例に沿って説明する．

（1）　英文の解析処理

まず入力英文を単語に分割し，辞書を検索する．この際，動詞，名詞，形容詞などの語形変化のある単語は，その形態素を解析し，標準形を選択する．次に品詞や属性の並びから，多品詞語の品詞を決める．たとえば，funは名詞と動詞の多品詞語であるが，例文ではisとの関係から名詞を選択する．

次に構文解析は，句解析，述部構造の解析，修飾関係の解析の順に行う．句解析では名詞句や前置詞句などを認識する．述部構造解析では動詞の文形を判定し，主語，目的語，補語などの役割をする構文要素を認識する．この時にIt is～to～のような構文において to 以下の不定詞が isの真の主語であることなども認識する．そのほか，動名詞や関係節のように，述部が述部を含む再帰的構造の認識も行う．

修飾関係の解析では，副詞や前置詞句などの任意格的要素の修飾先を認識する．この修飾関係は図7.9に示すように，句構造のリンク（実線）とは異なるリンク（破線）で表現している．この方式を疑似的句構造と呼ぶ．従来の英語の解析では句構造が用いられてきたが，前置詞がいくつも使用された文の場合

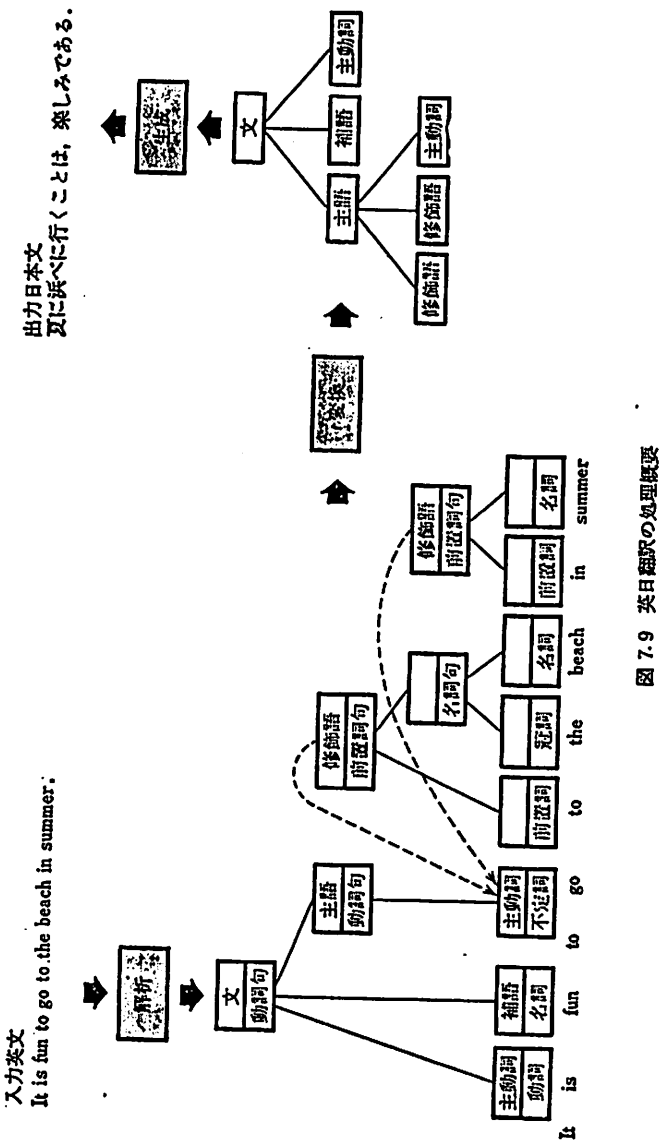


図 7.9　英日翻訳の処理概要

入力英文	出力日本文
(1) Personal income rose 0.5% in December, while consumer spending increased 1.2%.	(1) 個人所得は，12月に0.5%上がった．一方，消費者支出は1.2%増加した．
(2) The President's Council of Economic Advisors recommended some technical changes in Fed procedures for somewhat greater money supply growth.	(2) 大統領の経済諮問委員会は，幾分より大きなマネー・サプライの成長のために連邦準備制度理事会の手順のいくつかの技術的変更を勧告した．

図 7.10　英日機械翻訳システムによる経済記事の翻訳例

に，その修飾関係を句構造の生成順に決める必要があり，処理が複雑になっていた．それに対し，動詞を中心とした主語，目的語などの必須格を従来通りの句構造で表現し，副詞，前置詞句などの任意格を別のリンクで表現する疑似的句構造により，任意格の解析順序に関する制約がなくなる．たとえば，例文のin summerという前置詞句がgoを修飾することが認識できれば，もう一つの前置詞句のto the beachの認識が終っていなくてもgoへのリンクをつけられる．この方式は構文解析木の段数が少なくなるという利点もある．

（2）　中間言語の変換

疑似的句構造で表現された中間言語を，日本語向きの格的句構造に変換する．この格的句構造は，述語を中心とする格構造を基本単位とし，格構造の要素に格構造を埋め込んだ再帰的構造である．格構造の要素には，その深層格的意味やニュアンスを表す属性をつける．

この変換処理は，英語と日本語の表現の違いを解決するのが主目的であり，格構造の変形処理が中心となる．語順の変更，能動構文から受動構文への変換，無生物主語の道具格への対応づけなどの処理が行われる．

（3）　日本文の生成

最後に，格的句構造の中間言語から日本文を生成する．主語，目的語などの構文的役割に合った助詞を付加したり，動詞，形容詞などの活用語尾を決める形態素処理と，語発やニュアンスに適した訳語の選択処理が行われる．図7.10に，本システムを用いて経済記事を翻訳した例を示す．この例からわかるように実際の文書の翻訳には，専門語辞書の充実が重要である．

参考文献

1) 長尾真（監修）：日本語情報処理，電子通信学会編（1984）
2) 溝口文雄（編集）：大特集：機械翻訳，情報処理，26，10，pp. 1139-1236（昭60-10）
3) 新田義彦（編集）：特集：計算言語学，情報処理，27，8，pp. 854-954（昭61-8）
4) 梶博行，岡島惇，吉村紀久雄：意味処理に基づいた機械翻訳，日立評論，69，3，pp. 239-248（昭62-3）