知識工学と人工知能　30－4
（１９８３．６．２）

フレーム型知識表現言語ＦＭＳの構造について

伊藤秀昭　上野晴樹
東京電機大学　理工学部

1.　はじめに

知識型システムを構築しようとする際のテーマの一つに、知識表現（knowledge　representation）の問題がある。この問題は、知識工学におけるテーマである知識の利用、知識の獲得と並ぶものであり、知識型システム構築上のキイとなるものである。これには、プロダクション・システム、因果ネットワーク、セマンティック・ネットワーク、ブラックボード・モデル、およびM. Minsky により提案されたフレーム理論［ 1］に基づくフレーム・モデル等がある。そしてモデルが提案されると、それを取り扱い処理するためのツール、いわゆる知識表現言語もまた開発されている。この種の言語は、知識型システムを開発するためのプログラミング言語であり、従来の手続き中心のものではなく、宣言的に知識を定義し、利用するためのプログラミング言語である。

現在、研究、開発を進めているＦＭＳ（Ｆrame　Manipulation　System ）は、フレーム型知識表現言語の一つである。本システム開発の目的は次のものである。

－　複雑で高度な問題解決のための知識を表現するツールの開発。
－　人間（専門家）の意思決定のメカニズムの解明のための研究用ツール。

つまり、ＦＭＳは知識型システムを構築するための実験用ツールとなることを目指したシステムである。ＦＭＳが研究、開発用ツールとしてすぐれていると考えられるのは、標準的な階層型フレーム・モデルを構築することができるだけでなく、極端に言えば各種表現モデルを構築できるという可能性を含んでいるからである（後述）。また、知識型システム開発において問題となる、複雑で大量の知識の管理ということに関しても、ＦＭＳは、特に有効であると考えている。本稿においては、ＦＭＳの構造、知識表現、インプリメンテーション等について述べる。なお、ＦＭＳは、ＵＮＩＴＳ［ 5, 7］など一連のフレーム型知識表現言語を参考に設計され、ＵＴＩＬＩＳＰ［12］で記述されており、ＦＡＣＯＭ　Ｍ１６０　ＴＳＳ　環境下で稼動している。

2. 知識の表現とその利用

フレーム型システムが研究用汎用ツールとしてすぐれていると考えられるのは、種々の知識表現に関する要求事項［13, 20］をほぼ満たすと思われるからである。汎用フレーム・モデルが知識表現に対して強力なものであるのは、次のような構造をもつ知識ベースであるからである。概念対象（conceptual　objects ）をフレームと呼ばれる知識表現単位としてとらえる。Conceptual　objects の具体的な性質、事実の記述等は、フレームの構成要素であるスロットに記入される。概念には抽象的な階層性があるので、全体的な知識は一つの階層構造を持ったフレーム・システムとして構築される。これが知識ベースを構成する。また、特定のconceptual　objectに対する特有の推論手続きは、attached　proceduresとしてそのフレームのスロットの一つに記入される。したがって、そのフレームの評価を行うときにはきわめて有効な機能と成り得る。更にＦＭＳは、特定の推論機構を持たない。このことは、システムの利用者が目的に応じて自由に推論機構を設計できる。特にattached　proceduresは、局面に応じた推論メカニズムの記述を容易にする。しかしながらattached proceduresの設計、開発はユーザにとっては多少負担となるであろう。

ＦＭＳ設計においては、フレーム・モデルの基本的機能を実現するようにした。すなわち、

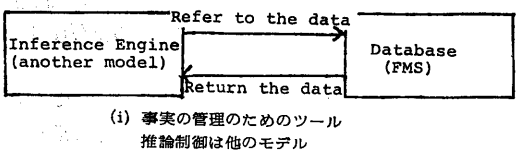
－　上位概念（上位フレーム）は、下位概念（下位フレーム）を包摂［ 8］しているので、それに対応できる。
－　上位フレーム、下位フレーム間に何らかの制約（下位フレームにおけるスロットの属性または値に対する）条件が存在するスロットに対しては、それが記入できなければならない。
－　フレーム・システムは階層構造であるが、フレーム間の参照関係が成立するようなネットワーク構造も表現できる。
－　抽象的概念と具体的な例示との区別。

これらの機能を組み合せることにより、複雑で大量の知識を知識ベースとして構築でき、しかも極めて柔軟で強力な推論を実現できると考えられる。

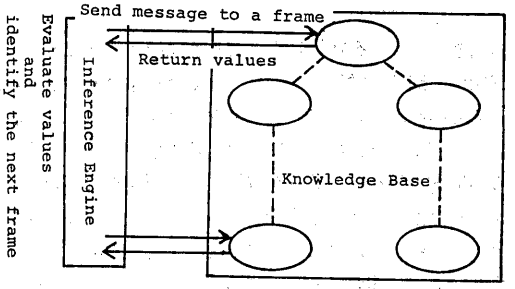
FMSを用いて、知識型システムを構築する際、FMSには次のような三種の利用形態ががあるのではないかと思われる。これらは、推論機構をどのように設計するかということと深くかかわっている。

最初のものは、FMSを事実の管理のためのツールとしてFMSを用いる方法である。フレーム・モデルには上述したような機能が具体的に備わっているので、フレーム・モデル自身が事実の管理ということに関しては強力である。よって、これを他のモデル、たとえば、プロダクション・システムの構成要素であるDatabase の部分に用いる［21］。つまり、推論機構の部分は、他のモデルを用いるが、事実の管理およびobjects の定義等にフレーム・モデルを用いる方法である。

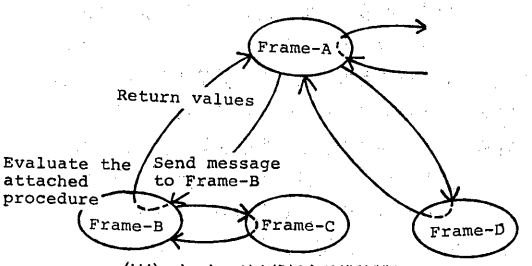
次に考えられるのが、推論機構とattached proceduresを併用する方法である。推論機構としてグローバルな処理手続きを用意し、そしてそのフレーム固有の処理をattached proceduresにより記述しようとするものである。ここで、attached procedure の役割は、基本的にそのフレーム内の評価を行うということである。つまり、FMS利用者が作成した知識型システム利用者に対して、推論機構および推論制御の一部に融通性を持たす必要がある知識型システムである場合に用いられると考えられる。ここで、attached proceduresはローカルな処理のために利用される。推論機構が特定のフレームにメッセージを送ると、attached procedure が起動されその結果を推論機構に返す。推論機構では、次にメッセージを送るフレームの決定および結果の評価等が行われる。この際、ローカルな推論のための情報と手続き、注視点の制御（focus of attention ）のための情報、終了条件のための情報や手続き等を知識ベースと推論機構にどのように持たせるかは、FMS利用者の考え方、システムの性格により大きく異なるであろう。



(i) 事実の管理のためのツール
推論制御は他のモデル



(ii) 推論機構とattached proceduresの併用



(iii) メッセージ交換による推論制御

図1 FMS の利用形態別分類

三番目の利用形態は、フレーム相互のメッセージ交換によつて推論を進めていくという形態である。すべての推論メカニズムを、attached proceduresにより実現する方法である。あるフレームの評価において他のフレームの評価結果が必要となった場合、フレームは、目的とするフレームに対して評価の依頼（attached proceduresの呼び出し）を行い、その結果を呼んだフレームに返す。このようなメッセージ交換を繰り返してgoalに近づいてゆく。この方法は、知識のモジュール化を高める意味においても有効な手段となり得る。先に報告したFMSを用いたCOMEXモデルの表現ではこの形態が応用されている［13］。このような推論制御を行う場合、知識ベースとなるフレーム・システム全体についてかなり明白な見通し（階層構造の設計）が必要となるであろう。ただし、フレームの評価方法および表現されているobjectの具体的な内容については問わない。ここで、もし階層構造およびattached proceduresがどのような処理を行うかに関してはっきりした見通しがないならば、知識ベースが混乱したものとなり、フレーム・システム自身がかなり能動的に動くので暴走を起す可能性があるように思われる。

以上、三種類に分類を施したが、厳密に区別されるものではない。知識型システム設計に際して、どのような推論制御を選択するにしろ、その問題の性質、システムの性格により大きく異なることはいうまでもない。図1は、それらの概略を示したものである。

3. FMSの構成、構造

FMSシステムは、4つの主なモジュールから構成されている。それらは、フレーム・エディタ、ルール・チェッカ、ACTIVATOR、およびシステム関数群である。

フレーム・エディタは、知識ベースの定義や更新など、フレーム・システムの構築のために用いられる。推論制御機構もこれを用いて書かれる。これは知識ベースの管理において、ユーザ・インターフェイスとなる。詳細は文献［20］参照。ルール・チェッカは、フレーム・エディタを用いて構築された知識ベースの規約上のミスを検出するために用いられる。部分的なミスは、フレーム・エディタでも検出されるので、これは主に全体的な矛盾などの検出を行う。構築された知識ベースのconsistency をチェックするものが、ルール・チェッカである。現在、ルール・チェッカは次のようなチエックを行っている。１）データ型を基とに参照関係が成り立っているフレームが定義されているか、　２）同様に、attached procedure が定義されているか、３）フレームを構成しているスロットに値またはデフォルトが記入されているか、等である。これらのチェックは、フレーム・エディタにおいてもメッセージ、コメント等を示すが、ルール・チェッカは確認をユーザに対して求める。

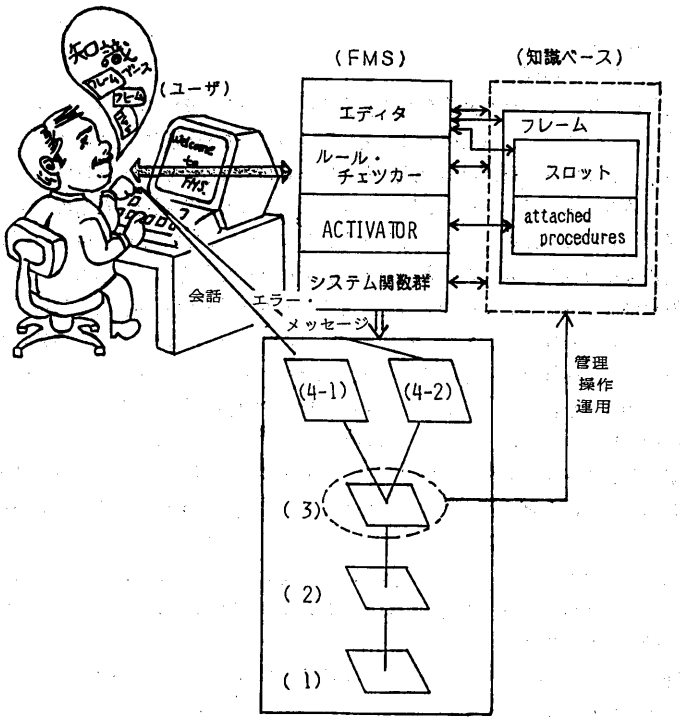


図２ FMS の概要

ＡＣＴＩＶＡＴＯＲは、推論制御を起動するためのものである。次の二つの機能を持っている。それぞれ、（ｉ）フレーム名、スロット名（後述）、およびargument－listをパラメタとして、（ii）関数名をパラメタとして引き渡し推論開始とする方法が機能として用意されている。また、前者の方法による起動は、そのフレームに取り付けられているattached procedure を評価することができるので、そのフレーム、サブ・フレーム・システムの部分的な評価を行うことができる。つまり、他のフレームとの相互作用を考えなくても推論の一部を実行することができる。また、推論の実行結果、評価結果がシステム利用者の思惑と異なった場合においても、どのフレームおよびサブ・フレーム・システムが思惑と異なった動作を行っているかを確めることができる。この方法は、実際に知識ベース・システムを構築する際（デバックを含めて）きわめて有効な機能であると考えられる。

システム関数群は、attached procedures等を書き易くするために備えられているＬＩＳＰ関数群である。但しこれらは共通関数であるので、システム自身も使っている。ＦＭＳを構築するために開発された関数群は、次のように階層化されている。（１）ＬＩＳＰシステム自身が持っている原始関数群、（２）それらを組み合せて高度なリスト処理を行う関数群、（３）知識ベースを意識しフレーム、スロットに対してなんらかの操作を行う関数群、（４－１）フレーム・エディタで用いられるユーザ・インタフェイスとなるための関数群（エディタ用関数群）でありＦＭＳで用いられる概念の説明等の出力、ある操作を行うためのパラメタ指定方法等の説明を組み込んだものである。（４－２）attached procedures等の記述を容易にするためにエラー処理等の機能を含むような関数群である。たとえば、定義されていないスロットが参照された時、もうすでに定義されているフレーム名と同じ名前により新しいフレームを生成しようとした時、等にエラー・メッセージを出力するような関数群も含んでいる。なお、（４－１）、（４－２）の関数群は、（３）の関数群を利用して作成されている。知識ベースを操作および処理するための関数群は次のものに機能的に大別することができる。１）フレーム操作用関数群。フレームの生成、削除、階層構造の変更、等、２）スロット操作用関数群。スロットの生成、削除、値の記入、等、３）attached proceduresの起動、４）種々のチェックを行う関数。値とデータ型、インヘリタンス・ロール、Ｆrame－type（後述）の相互関係が満足されているかを調べる関数。現在、システム関数群の基本的部分は完成されており、今後アプリケーションを通じて拡張する予定である。図２は、これらの概略を示したものである。

4. 知識表現

4.1 フレームの構造

図3にフレームの基本構造を示す。各フレームは、そのフレーム・システムにおいてユニークなフレーム名をもち、複数のスロットから構成されている。その中の9つは、全てのフレームがもつ共通のスロットである。：Frame−typeスロット。すでに述べたように、フレームはconceptual objects を記述したものである。このスロットには、知識ベース中に形成されたworld においてそのフレームが抽象概念（類またはprototype と考えてもよい。）を記述したものであるのか、または具体的な例示（instance）を記述したものであるかを示す情報が記入される。FMSでは、それぞれsubclass、instanceとなっている。ユーザは、この二種のFrame−typeのうちどちらかを指定しなければならない。階層構造中において、subclassとなっているフレーム（subclass node frame ）は子孫フレーム（descendant frames）を持てるがinstance node frame は持てない。A−kind−ofスロットは親フレームへのポインタ、D. Descendentsスロットは子フレームへのポインタ・リスト、Description−information スロットは、そのフレームに関する備考などのテキスト値、Create スロットおよびModify スロットは、そのフレームがいつ誰により生成、修正されたかを示す。上記以外のスロットは、目的に応じて自由に定義されるものであり、数の制限はない。

Frame-name	Frame-type
A-kind-of slot	
D.Descendant slot	
Description-information slot	
Create slot	
Modify slot	
Slot 1	
⋮	
Slot 2	

Frame-type: instance
subclass

図3 フレームの構造

4.2 スロットの構造

各スロットは、現在のところ、7つの要素から構成されている（図4）。すなわち、スロット名、インヘリタンス・ロール、FROM部、データ型、データ値、デフォルトおよびオプションから成る。

Slot-name	Inheritance-role	From	Data-type	Value	Default	option

Inheritance-role;S,Same value
U,Unique value
R,Restriction value
M,Member value
O,Option value
I,Independent value

Datatype ;ATOM TEXT TABLE BOOLEAN LISP
FRAME LIST RANGE INTEGER
NUMBER NUMERIC EXPR FSYSTEM
FLIST STRING

図4 スロットの構造

スロット名： そのフレーム内においてユニークなスロット識別名である。

インヘリタンス・ロール： 現在、インヘリタンス・ロールは、6種を持っている。インヘリタンス・ロールの概念および具体的機能は後述する。

FROM部： そのスロットがフレームをノードとするような階層構造の中で、どのフレームにおいて最初に定義されたかを示している。´＊TOP＊´と記入されているスロットは、そのフレームにおいて定義されたものである。ただし、この要素はシステムが管理しているので、特にユーザは意識する必要はない。

データ型： 各スロットの値部がとる値の属性を指定するものである。データ型ＬＩＳＰはattached proceduresに、FRAMEはフレーム間のリンクのために用いられる。詳細は文献［13］参照。

デフォルト部： そのスロットにおいて値が決定されていない場合にデフォルト値を記入することができる。しかし値とデフォルトを同時に記入することはできない。

なお、オプション部は、システム利用者が自由に用いることができるように融通性をもたしたものである。システムは何のチェックもしないので十分注意しなければならない。短期記憶（ＳＴＭ）の代用や、そのスロットに対するメッセージ、コメント等の格納、等のために利用できる。

5. インプリメンテーション

5.1 インヘリタンス・ロールの実現

フレーム・モデルにおいてインヘリタンスの問題は、特に重要である。なぜなら、あるフレームにおけるスロットの値が、どのようにしてなぜ他のフレームから決定されたかを示すからである。このインヘリタンスの仕方を表わしたものが、インヘリタンス・ロールである。FMSにおいては、インヘリタンス・ロールはスロットの属性となっている。インヘリタンス・ロールは、多くの汎用フレーム・システムで採用されており、その実用性、有効性が確認されている。この理由は、インヘリタンス・ロールの指定により、親フレームのスロットから子フレームのスロットの値に対して制約条件が生

じるからである。これは、知識の定義、値の設定および更新や利用を矛盾のないものとするとともに、知識の定義の経済化にも役立っている。つまり、インヘリタンス・ロールの役割は、子フレームのスロットの値に対して何らかの制約条件を生じさせることである。

FMSを用いて、スロットに知識を設定しようとした場合には、常にトップ・ダウン的に値を設定しなければならない。これはあるスロットの値の設定に際しては、親フレームでのスロットの値が定められて初めて子フレームでの値が設定できるという、知識の定義順序を意味している。これは、ある概念対象を常に、その上位概念から明らかにしていくという立場に立ったものである。

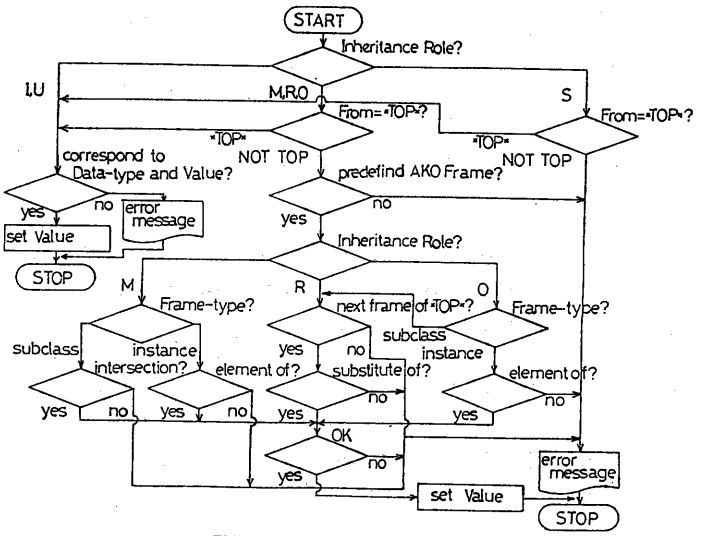


図5　チエックの流がれ

FMSにおいては、6種のインヘリタンス・ロール（S、M、R、O、U、およびI）が準備されている。インヘリタンス・ロールの指定に際しては、十分にそのスロットの値の特性を考えなければならない。このことは以下に述べることにより明らかである。以下、各インヘリタンス・ロールの役割およびそのスロットに記入しようとしている値に対してどのようなチェックが行われるかを述べる。尚、親フレームから子フレームのスロットへ必ず受け継がれる属性は、スロット名、データ型およびインヘリタンス・ロールである。

```
<<<< HIERARCHY >>>>
 ROOT
      LISPPROC(*)
      TABLE
      ANIMAL
           DOG
                POCH(*)
                TARO(*)
                SHIRO(*)
```

FRAME-NAME:DOG			FRAME-TYPE:SUBCLASS	
A-KIND-OF	(U)	ROOT	FRAME	ANIMAL
DDESCENDENT	(U)	ROOT	FLIST	(POCH TARO SHIRO)
DESINF	(U)	ROOT	TEXT	"DOG"
CREATE	(U)	ROOT	STRING	("G00126" "8305180958")
MODIFY	(U)	ROOT	STRING	("G00126" "8305181012")
IF-NEEDED	(U)	ROOT	LIST	NIL
IF-ADDED	(U)	ROOT	LIST	NIL
IF-REMOVED	(U)	ROOT	LIST	NIL
REQUIRMENT	(S)	ANIMAL	ATOM	02
COLOR	(M)	*TOP*	LIST	(BLACK WHITE TOWNY-COLOR)

FRAME-NAME:POCH			FRAME-TYPE:INSTANCE	
A-KIND-OF	(U)	ROOT	FRAME	DOG
DDESCENDENT	(U)	ROOT	FLIST	NIL
DESINF	(U)	ROOT	TEXT	"ANIMAL IS POCH"
CREATE	(U)	ROOT	STRING	("G00126" "8305180959")
MODIFY	(U)	ROOT	STRING	("G00126" "8305181013")
IF-NEEDED	(U)	ROOT	LIST	NIL
IF-ADDED	(U)	ROOT	LIST	NIL
IF-REMOVED	(U)	ROOT	LIST	NIL
REQUIRMENT	(S)	ANIMAL	ATOM	02
COLOR	(M)	DOG	LIST	WHITE

図6　フレームの例

1）S（Same）：親フレームと同じ値を持つスロットであることを意味する。設定に際しては、常にFROM部が“*TOP*”でなければならない。子孫フレームへの設定は自動的に行われる。値は、データ型との一致を見なければならない。

2）M（Member）：“Subset of”の関係にある値のために用いられるロールである。subclass node frameには親フレームとの値との関係が常にSubset ofの条件を満たさなければならない。当然データ型とのチェックが行われる。また、instance node frameでは親フレームでの値の一つの要素となっていなければならない。このロールとの組合せが可能なのは、リスト形式を値に持たなければならないデータ型である。

3）R（Restriction）：“Substute of”の関係にある値のために用いられるロールである。このロールに対しては、subclass node frameにおいては一意にその値を決定できる。値は、親フレームでの値の範囲内になければならない。

4）O（Option）：Rと同じようなものである。しかしながら、instance node frameにおいては一意にその値が決定される。もちろん親フレームでの値の範囲内になければならない。

5）U（Unique）：値は、その親フレームでの値の制限が子フレームの値に対して何ら制限を与えるものではない。

スロットの値は、知識ベースの階層構造の中でそれぞれのレベルについてユニークなものである。値には、データ型との一致が求められる。

6）I（Independent）：その概念対象の中だけに、存在すればよい知識のための指定であり、下位フレームへの影響はない。チェックは、データ型との一致だけである。

上記のように、インヘリタンス・ロールがどのようなものであろうと最低限のチェックとしてデータ型と値の対応関係は調べなければならない。但し、例外的に`M´と`O´については、データ型との不一致が生じる。なぜなら、instance node frame においては、値を一意に決定することができるからである。しかしその値に対しては、それぞれのデータ型がとるべき値の各要素についている属性は守っていなければならない。たとえば、データ型がＦＬＩＳＴ（値はフレーム名のリストである。）であるスロットの場合、値部にはフレーム名が記入される。図5は、スロットに値を設定しようとした場合の基本的なチェックの流れを表している。

4つのインヘリタンス・ロール（S、M、R、O）は、`S´を除いてとり得る値が常に親フレームの値よりも小さな範囲を示したり、制限をより厳しくするようなものである。我々がこのようにしたのは、上位概念が下位概念を包んでいるとしうる定説によるものである。ユーザは、常にこのことを意識して知識モデルを作成しなければならない。よって親フレームの値よりも子フレームの値の範囲を拡大したり、より制限を緩めたりするようなインヘリタンス・ロールは必要でないと考える。図6は値の記述を行った例である。

5.2　システム関数の例

Attached procedureの評価を行うための関数がいくつか用意されているが、それらの一つに関数ＭＥＳＳＡＧＥ＃がある。これは、フレーム相互のメッセージ交換のために用いられる。しかし、一般的なメッセージ交換ということではなく、現在のＦＭＳではかなり制限の強いものとなっている。つまり、ただ単純にattached proceduresの評価を行うというのみである。この関数のフォームは次のものである。

ＭＥＳＳＡＧＥ＃（スロット名、フレーム名、argument－list）

フレーム名、スロット名より求められるスロットの値部には、そのフレームにおけるattached procedure となる手続き名（関数名）が記入されている。ＭＥＳＳＡＧＥ＃は、その値部に記入された関数にargument－listを適用し、その評価を行い、値として返す。この関数が実行されるためには、次のような要件が満たされなくてはならない。フレーム、スロットおよび値部が定義されていなければならない。スロットのデータ型は、ＬＩＳＰであること。値部に記入されているsymbolは、関数として定義されていなければならない、等である。これらの要件に反する場合、ＭＥＳＳＡＧＥ＃は、その旨を明記したエラー・メッセージを出力し、その実行を保留する。

Attached proceduresを起動するためには、フレーム、スロット、値部、そして値というパスを通じてその評価を行うという、間接的な手続きの呼び出しが可能という形式となっている。このような定義の方法は、メッセージを送った側のフレームは受け取り側の詳細な処理手続きまでをも考えなくてもよく、ただ返ってくる値にのみについて考慮し継続処理を行えばよい。こうすることにより、さらに知識のモジュール化が高まる。この方法の利点は、SmalltalkのようなObject －orientedプログラミング言語と同様のものであると考えられる。フレーム型システムを、この種の言語の一つと考えることができる。また、インヘリタンスの機能をこれに組み合わせれば、上位フレームを下位フレームに対するprototype としてみることもでき、処理手続のインヘリタンスも容易に行うことができる。

5.3　エラー処理

ＦＭＳは、attached proceduresを評価実行中に知識ベースに対して異常な操作が行われようとした場合、エラーを発生する（例えば、知識ベース中に存在しないスロットが参照された場合、等）。エラーが発生すると、その原因を示すコメントが出力され、ＢＲＥＡＫループに入る。システム利用者は、ＢＲＥＡＫループ中で通常のＬＩＳＰシステムと同様な方法でエラーに対処する。ループ中では知識ベースの構造は意識されていないので、内装エディタ等の使用には十分注意しなければならない。

ＦＭＳでは、エラー処理ということに関してerror recoveryということを考慮していない。これは次の理由による。Attached proceduresは、そのフレームの評価の仕方等を記述したものである、したがってその知識ベースに対する処理手続きは全て明白に定義されていなければならないと考えられる。これによりattached proceduresが具体的にどのような処理を記述したものであるかを明白にすることができる。もし、error recoveryが、次のように考慮されていると

する。例えば、システム関数ＭＥＳＳＡＧＥ＃を評価しようとした場合、スロット名、フレーム名が入れ替っているとrecovery機能が解釈した場合、もし知識ベースがそのように解釈すれば先述した要件が満たされ、それが実行されるとシステム利用者の思惑とはまるで異なった動作を起しかねない原因となる。また、スロットの値を求めるために、不用意にそのスロットのデフォルトおよびインヘリタンスされた値を利用しようとするならば、そのprocedure 中ではまるで役に立ない値が入っていて、そのまま実行される場合等がある。Recovery 機能が働いてシステム利用者が考える結果と異なった際の原因追及は、非常に困難なものであると考える。よってattached procedures中において用いられる関数は、その動作の仕方が明白に定義され、そして定義された知識ベースに対してのみ作用しなければならない。こうすることにより、デバックを含めて知識の管理が容易になると思われる。ただし関数は融通性に欠けたものとなるであろう。なお、attached procedures中では前述した全てのレベルの関数が用いられるが、エラー処理を含まないレベルの関数の使用に関しては知識ベースに対する異常動作の原因をつかみ難いのでそれらの使用には注意しなければならない。

エラー処理は、次のように実現されている。例えば、関数ＧＥＴＶＡＬ＃（スロット名、フレーム名）（スロットの値部を返す関数。）は、スロットが未定義、フレームが未定義の場合にエラーを発生する。この関数の定義を示す（図７）。エラーと認識されれば、´ＦＭＳ：´という接頭辞を持つシンボルに関数ＦＵＮＣＡＬＬが適用される（この場合、ＦＭＳ：ＦＲＡＭＥ－ＮＯＴ－ＤＥＦＩＮＥＤ、ＦＭＳ：ＳＬＯＴ－ＮＯＴ－ＤＥＦＩＮＥＤ－ＩＮ－ＦＲＡＭＥ）。これらのシンボルは、大域変数として宣言されており、値にはエラー・メッセージを出力しＢＲＥＡＫループに入るという定義を持つＬＡＭＢＤＡ式が格納されている。ＬＡＭＢＤＡ式中では、状況を説明するために、スロット名、フレーム名、そしてエラーが発生した関数名を引き数として渡され、それらが出力される。このように、シンボルにＬＡＭＢＤＡ式を格納し評価するという間接的な呼び出しを行うというのは、ユーザ指定によりエラー処理を行える、ということを可能にしている。Ｅｒｒｏｒ recoveryも、これにより実現可能である。この実現方法は、ＵＴＩＬＩＳＰと統一を図るものである。この種のシンボルは、現在２９個用意されている。エラー処理のための関数は、接尾辞´＃´を持っており、現在、約６０個ある。

```
(DEFUN GETVAL# (SNAME FNAME)
   (PROG (ANS)
      (SETQ ANS (SUB-GETVAL SNAME FNAME))
      (COND ((MEMBER ANS FMS-ERRORLIST) (GO ERR)))
      (RETURN ANS)
ERR   (COND ((EQUAL ANS 'FRAME-NOT-DEFINE)
             (FUNCALL FMS:FRAME-NOT-DEFINED FNAME 'GETVAL#))
            ((EQUAL ANS 'SLOT-NOT-DEFINE)
             (FUNCALL FMS:SLOT-NOT-DEFINED-IN-FRAME SNAME FNAME 'GETVAL#)))))
```

図７ GETVAL# の定義

6. まとめ

フレーム型知識表現言語ＦＭＳの、構造について述べた。現在、ＦＭＳは開発の第一段階が終了した。今後、アプリケーションを通じてシステムの拡張を図る予定である。たとえば、Ｓmalltalkのようなプログラミング言語システムにおいて導入されているmultiple inheritance 、およびＮＥＴＬ［19］のようなtangled hierarchy 等の考え方、ＦＭＳが持っているデータ型とインヘリタンス・ロールの種類と効果の評価等である。また、ＦＭＳは人間の意思決定のメカニズムを解明するツールとしても開発されているので、人間の知識構造の理解、表現のモデルを記述することは不可欠なものである。このような方向で開発を進めることにより、システム関数群、フレームの構造等に要求される機能、メタ知識等が浮かぶものと考えられる。

また、メタ知識の一種であるＦrame－typeの問題がある。現在、フレーム自身の属性を示すものとしてsubclass、instanceが用意されている。例えば、抽象的な概念を表現するためにＦrame－typeをsubclassとして登録する。このフレームにより表現されたconceptual objectが不完全なものであるのか、完全なものであるのか、またフレーム・システム中において定数、変数のような用いられ方をするのか等を示すようなメタ知識は導入されていない。このような問題に対しても具体的なアプリケーションが必要であろう。なお、アプリケーションを通じて得られた結果に対しても考察は必要である。なぜなら、システムはコンパクトなものとしてまとまっているほうがよいと考えるからである。

最後に、本システムは約３０００行のプログラムである。ただし、各種説明等のためのテキスト、およびコメントは含まない。

謝辞

システムのインプリメンテーションを行なってくれた小沢健司、高島伸彦の両君、並びにＵＴＩＬＩＳＰの使用を許し

ていただいた東京大学　和田研究室に感謝する。

参考文献

1) M. Minsky ; A Framework for Representing Knowledge, in P. Winston (ed.) The Psichology of Computer Vision, Mc Graw-Hill (1975)
2) 上野晴樹；フレーム理論に基づく知識型システム、IPSJ人工知能と対話技法 21-3 (1981)
3) H. Ueno, D. Lindberg et al. ; Design of a Criteria-Based Rhematology Consulting System for a Microcomputer, Proc. MEDINFO 80 (1980)
4) I. Goldman; NUDGE: A Frame-Based Scheduling System, Proc. 5th IJCAI (1977)
5) M. Stefik ; An. Examination of a Frame Structured Representation System, 6th IJCAI (1979)
6) D. G. Bobrow and T. Winograd ; An Overview of KRL, a Knowledge Representation Language, Cognitive Science, Vol. 1, No. 1 (1977)
7) R, G, Smith and P. Friedland; UNIT Package User's Guide, Stanford Heuristic Programming Project Memo HPP-80-28 (1980)
8) 井上昌計；論理学、成文堂
9) 上野晴樹；COMEX：汎用知識型エキスパート・システム開発言語、情報処理学会知識工学と人工知能研究会資料、1982
10) 上野晴樹、COMEXマニュアル、東京電機大学経営工学科、1982
11) 上野晴樹、データベース技術と知識ベース・システム・シンポジューム資料、1982
12) Chikayama, T; UTILISP MANUAL, 1981、東京大学
13) 上野晴樹、伊藤秀昭； FMS： フレーム理論に基く汎用知識表現言語、AL-82-64、電子通信学会オートマトンと言語研究会資料、1982
14) D. Lindberg, G. Sharp, H. Ueno, et al. ; Computer-Based Reheumatology Consultant, Proc. MEDINFO 80 (1980)
15) S. J. Greenspan, J. Mylopoulos ; Capturing More World Knowledge in the Requirement Specification, Proc. 6th ICSE (1982)
16) Mark S. Fox; On Inheritance in Knowledge Representation, Proc. 6th IJCAI (1979)
17) P. H. Winston, B. K. P. Horn ; LISP, Addison-Wesley (1981)
18) A. H. Borning; Multiple Inheritance in Smalltalk-80, Proc AAAI-82
19) S. E. Fahlman; Representing and Using Real-world Knowledge; in Artificial Intelligence, An MIT Press (1979)
20) 上野晴樹、伊藤秀昭；フレーム型知識表現言語の設計およびインプリメンテーションについて、東京電機大学理工学部紀要、Vol. 5 (1983)
21) N. Aiello, C. Bock et al. ; AGE Reference manual, Stanford University, 1981