第4章　フレームシステム

フレームは、1975年のM.ミンスキーの論文によって提案されたフレーム理論に基づいている。しかし、ミンスキーの論文では、具体的な実現方法まで言及されていないため、システムの設計者が様々に解釈して、フレームと名が付くシステムをコンピューター上に実現している。そのため、標準システムは、いまだ存在しないというのが実状である。

それにもかかわらず、フレームは現在のAIの研究開発の中で最も有力な知識表現手法の一つである。この章では、フレームを、宣言的および手続き的な知識を表現する手段として利用する手法を紹介する。その上でフレームの効用と限界について考察する。

4.1　はじめに

ミンスキーの提唱したフレーム理論は、人間が日常行なっているような推論や認識活動を説明するためのものであった。我々が日常の生活で「これは知識だ」ととらえることができる実体を説明するには、プロダクションルールのような表現方法では余りにも単位が小さ過ぎる。例えば「家」に関する知識は、家の機能、形、大きさといった属性や、家の中の部屋や屋根、庭、部屋の中のドアや窓、窓の中のガラスや錠といった様々のレベルの知識が集積したものである。こうした知識の自然な表現は、より階層化され、高度に構造化されていることが望ましい。また、家の設計や建築法について記述した知識(家に関する手続き的知識)があれば、それらは家の属性や家を構成する要素を記述した知識と結びついていることが望まれる。つまり、事実に関する知識と手続きに関する知識が結びついていることが求められる。

こうした主張を根幹として、視覚的な情報処理を例にとった知識表現について述べたのが、1975年の論文である。この論文は「コンピュータビジョンの心理」(P.H.ウィンストン編　白井、杉原共訳／産業図書)に収められている。ミンスキー自身のフレーム理論は「K結線理論」として、さらに包括的な体系として発展している。現時点でのミンスキーの理論は、AIの様々の領域

で用いられているフレームとは、独立したものとしてとらえるほうがいいであろう。

さて、ミンスキーの理論から離れたフレームは、「型にはまった場面を表現するためのデータ構造」として一人歩きをしている。一般に知識の一単位であるフレームが、特定の規則で結合した複数のフレームデータの集合のことを、

部屋A	
スロット1 ファセット 値	part-of value アパート1
スロット2 ファセット 値	住人 value N氏

窓1	
スロット1 ファセット 値	part-of value 部屋A
スロット2 ファセット 値	方角 value 南

鍵	
スロット1 ファセット 値	part-of value 窓1

図4-1 part-of 階層の例

フレームシステムという。その特定の規則として、大きく分けてpart-of階層とako階層の2つのものがある。

part-of階層は、「部屋」などの場面を表現するときに、「部屋」の部分として「窓」や「ドア」、「机」が存在するときに、上位の「部屋」と下位の「窓」などを結び付ける関係の名前である。part-of階層では、上位のフレームの持つ属性は、下位のフレームの持つ属性とは独立である。例えば、「部屋」の持つ「住人」や「用途」といった属性は「窓」にはないものである。**図4-1**にpart-of階層の例を挙げる。

ako階層はa-kind-of階層のことで、別名isa階層とも呼ばれる。これは、動物－ほ乳類－肉食獣－ライオンといった階層関係を表すために用いられる。part-of階層とは異なり、ako階層では上位のフレームの属性を、下位のフレームが継承する事がある。例えば「皮膚がある」「呼吸する」といった個々の動物の持つ属性を、ライオン、シマウマ、ペンギンの所にいちいち記述しておくのは記憶効率が悪い。そういった共通の属性は「動物」フレームのところに

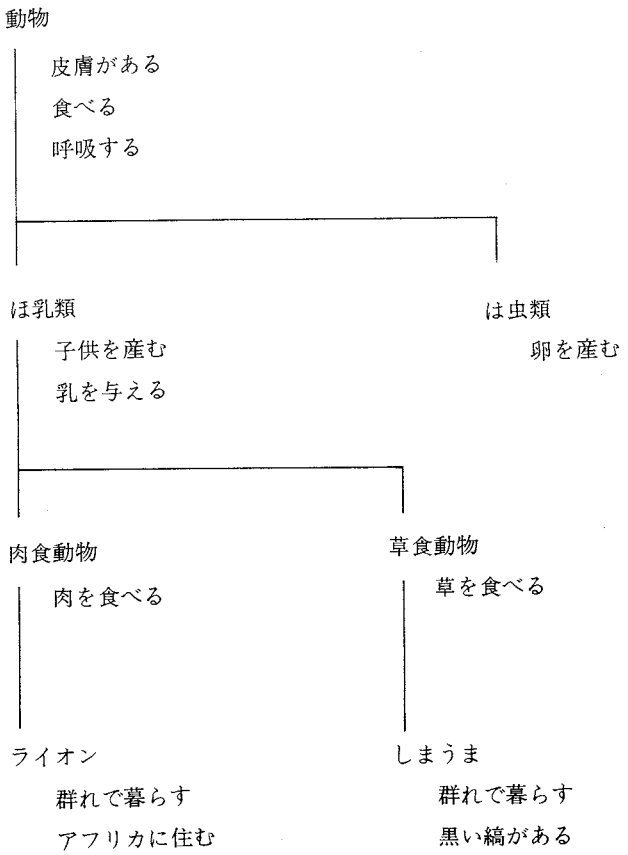

図4-2　動物に関する知識を表すフレームシステム

記述しておけば十分である。そういった場合、フレーム検索述語はako階層をたどって「ライオン」の「呼吸する」という属性を持つ事を見つけ出してくれる。

ako階層の例として、動物の世界を表す知識ベースをフレームで表現してみた。**図4-2**が対象モデルを図示したものである。

次の節ではフレームによる知識表現によって、どのような事ができるのかを考察することにする。

4.2　フレーム推論

データ表現の全ての分野と同様、フレームシステムにも、使用に適した分野とそうでない分野とがある。フレームによる表現に適した分野は、ある、決まりきった形の(状況が変わっても、考慮すべき属性値が変わらない)構造を持っていることが特徴になっている。

例えば、学校の時間割作成における科目と教室、教師、生徒の配分のようなスケジューリングを行う問題がある。例えば先生については、ある時間の仕事の有無や担当科目について考慮すればよい、といった具合いに考慮すべきスロット(属性)は固定している。

4.5節でインプリメントする、電気回路に関する推論システムにおいても、各配線について、どの端子につながっているのか、抵抗、電圧、電流の値はいくらかといった、固定した少数のスロットにのみ注目している。

また、フレームによる表現が適した問題の構造は、いくつかのはっきりした下位構造に分けられるようになっている。上位構造と下位構造を結び付けるのが、ako階層やpart-of階層である。

例えば、植物や昆虫の図鑑のような種類分けのきれいにできている、検索のためのシステムなどもフレームによる表現が効果を発揮する。キク科のタンポポの場合、タンポポの持つ属性の多くはキク科あるいは双子葉植物といった上位のフレームに記述しておくことができる。フレームのスロット値検索述語が、検索の際にako階層をたどって、上位フレームのスロット値を自動的に引き出してくれるからである。

4.4節で紹介する動物に関する属性の検索システムでは、そういったako階層による記憶消費の効率性が示されている。

次に、フレームシステムの利用の仕方について考える。利用の仕方には大きく分けて3つある。その内容を下に示す。

①　内容の分かっているスロット(属性)値から、内容の分からないスロット

値を導き出す。

例えば、スケジューリングの例でいえば、「月曜日の3時間目に1年2組の英語の授業を行なうには、どの先生をどの教室に配置したらよいのか」といった問題を推論させる。この場合、先生フレームの「曜日スロット」「時間スロット」「生徒のクラススロット」「科目スロット」の4つから、「名前スロット」「教室スロット」の2つの属性のとりうる値を推論するという利用法を採用したことになる。

② 一部のスロットの値が変わった時に、その影響を計算してシステム全体属性の一貫性を保つ。

例えば、電気回路の問題で、一部の回路がこわれたときに、各配線の電圧や電流がどう変わるかを計算しなおすといった機能を利用する。

③ フレーム名から、そのスロット値を引き出す。

例えば、植物名からはなびらの数や花の咲く時期といった属性を検索するといった利用法。

次にフレームシステムの構築法を考えよう。標準的なフレームシステムの構造は次ページの**図4-3**のようになっている。上半分がフレームシステムを構成する個々のフレームと推論に用いるスロットの関係を示している。

一番下にある「individual フレーム」はフレームの個々の具体的事例である。上部と「individual フレーム」との関係は、**図4-4**における「人間」フレームと「高野　真」フレームとの関係に相当する。

フレームシステムの利用の仕方として、既知のスロット値から未知のスロット値を導出するというものを挙げた。ここでの既知のスロット、未知のスロットは下の「individual フレーム」に属するものである。そして、既知のスロット値から未知のスロット値を導き出すための、スロット値の間の関係が上半分に記述してある。

スロット値の間の関係を用いて未知のスロットから既知のスロット値を導き出す手続きは、「デモン」と呼ばれている。デモンの主な種類としては「if_needed デモン」「if_added デモン」「if_removed デモン」の3種類がある。場合によっては「if_removed デモン」「if_added デモン」を一緒にしたような「if_changed デモン」も導入される。

if_needed デモンには、そのスロットの値が未知であるにも関わらず、他から参照された場合に値を見つけ出すための手続きが書き込まれている。

if_added デモンは、そのスロット値に値が入った時に自動的に起動され、フレームシステム全体の調整の指令を発する。

if_removed デモンも同様にスロット値が削除されたときに、他のフレー

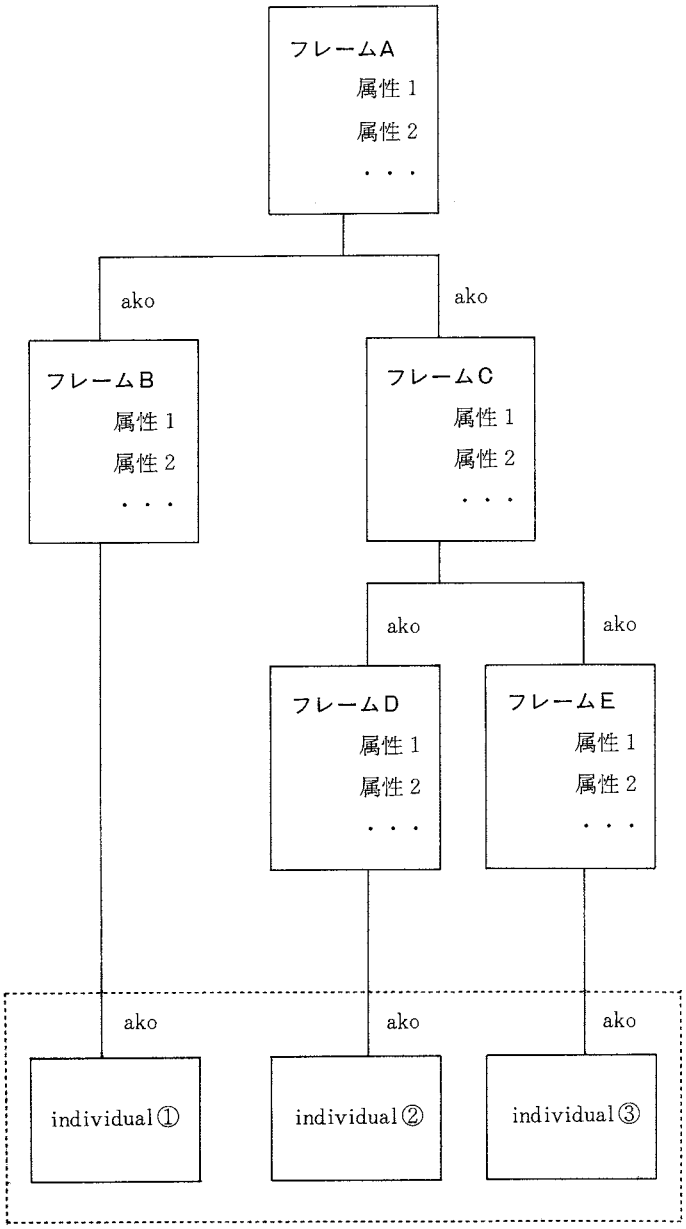

図4-3　フレームシステムの構造

ムに再計算の指令を伝える。

こういった、デモンの機能をメッセージ伝達に、ako関係をインヘリタンスと類似のものとしてとらえるフレームシステムは、オブジェクト指向言語に似ている。オブジェクト指向言語のメッセージ機能と同様に、デモンをいかにうまく用いるかが、フレームシステムの推論がきちんと働くかどうかの

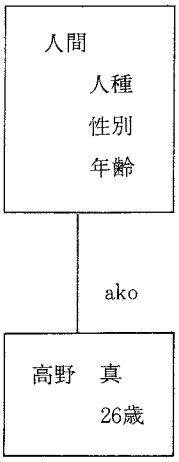

図4-4　individual フレームの例

鍵を握っている。フレームシステムの構築のための労力は大部分、このデモンの記述に費やされる事になる。(ただし、検索のためのフレームシステムは例外で、いかに記述の経済性を保つかが重要になる。)

4.3　フレームシステムのインプリメント

まず、フレーム検索プログラムを整備する。**図4-5**が今回用いたフレーム検索述語である。

fget 述語は引き数を 3 つ取る。第 1 引き数にフレーム名、第 2 引き数にスロット名を入れて実行すると、第 3 引き数にスロット値が返ってくる。最初の fget 述語はスロット値の継承を制限する restrict ファセットを処理する。2 番目の fget は最初に呼ばれた individual フレームに値が記述されている場合にその値を取ってくる。3 番目の fget は default ファセットの処理をする。4 番目の fget は if_needed デモンによって生成された値を返した上で individual フレームのスロット値としての登録を行なう。5 番目の fget は ako 階層をたどって上位フレームから値を引き出す機能を担っている。

fput 述語はフレーム名、スロット名、スロット値を与えられた時にそのフレームを登録する機能を持っている。if_added デモンの呼び出しも行なう。

fremove 述語はフレーム名、スロット名、スロット値で指定されるフレームを削除して if_removed デモンを起動する。

各デモンは、ako 階層をたどって上位のフレームから手続きを引き出すことを試みる。

```
%********************************%
%                                %
%       フレーム制御述語         %
%                                %
%********************************%
:-op(100,xfy,':').
%************************%
%       f g e t          %
%************************%
fget(Frame,Slot,not(Value)) :-
        frame(Frame,Slot,restrict,negrect:Value),
        asserta(restrict(Slot,negrect:Value)).
fget(Frame,Slot,Value) :-
        frame(Frame,Slot,value,Value),
        ¥+restrict(Slot,negrect:Value).
fget(Frame,Slot,Value) :-
        ¥+frame(FrameName,Slot,value,Value),
        frame(Frame,Slot,default,Value).
fget(Frame,Slot,Value) :-
        frame(Frame,Slot,if_needed,Value),
        ¥+restrict(Slot,negrect:Value),
        Frame =.. [_,Individual],
        ¥+frame(Individual,Slot,value,Value),
        fput(Individual,Slot,Value).
fget(Frame,Slot,Value) :-
        frame(Frame,ako,value,SuperFrame),
        fget(SuperFrame,Slot,Value).
%************************%
%       f p u t          %
%************************%
fput(Frame,Slot,Value) :-
        frame(Frame,Slot,value,Value),
        !.
fput(Frame,Slot,Value) :-
        retract(frame(Frame,Slot,value,_)),
        asserta(frame(Frame,Slot,value,Value)),
        if_added_demon(Frame,Slot,Value),
        !.
fput(Frame,Slot,Value) :-
        asserta(frame(Frame,Slot,value,Value)),
        if_added_demon(Frame,Slot,Value),
        !.
%************************%
%       f r e m o v e    %
%************************%

fremove(Frame,Slot,Value) :-
        retract(frame(Frame,Slot,value,Value)),
        if_removed_demon(Frame,Slot,Value),
        !.
```

```
%********************************%
%       if_addedデモン　%
%********************************%
if_added_demon(Frame,Slot,Value) :-
        frame(Frame,Slot,if_added,Value),
        !.
if_added_demon(Frame,Slot,Value) :-
        frame(Frame,ako,value,SuperFrame),
        if_added_demon(SuperFrame,Slot,Value).
if_added_demon(_,_,_).
%**************************************%
%       if_removedデモン      %
%**************************************%
if_removed_demon(Frame,Slot,Value) :-
         frame(Frame,Slot,if_removed,Value),
         !.
if_removed_demon(Frame,Slot,Value) :-
         frame(Frame,ako,value,SuperFrame),
         if_removed_demon(SuperFrame,Slot,Value).
if_removed_demon(Frame,Slot,Value) :-
```

図4-5　フレーム検索述語

4.4　動物に関する属性検索フレームシステム

動物に関するフレームシステムを例に、述語の機能を追ってみよう。動物に関するフレームを図4-6に示す。それを分かりやすい表記に改めたものを図4-7に示す。

```
%************************%
%       個体フレーム　%
%************************%
frame(しま子,ako,value,しまうま(しま子)).
frame(しし丸,ako,value,ライオン(しし丸)).
frame(虎丸,ako,value,とら(虎丸)).
frame(ペン君,ako,value,ペンギン(ペン君)).
frame(ピー子,ako,value,ひばり(ピー子)).
%************************%
%       所与のフレーム　%
%************************%
frame(動物(Ind),Slot,if_added,Value) :-
        write('フレーム：'),
        write(Ind),
        write('　の'),
        write(Slot),
        write('の値として'),
        write(Value),
        write('を追加しました').
```

```
frame(動物(Ind),Slot,if_removed,Value) :-
        write('フレーム：'),
        write(Ind),
        write('　の'),
        write(Slot),
        write('の値'),
        write(Value),
        write('を削除しました').
frame(動物(Ind),特徴,if_needed,皮膚がある).
frame(動物(Ind),行動,if_needed,食べる).
frame(動物(Ind),行動,if_needed,呼吸する).
frame(肉食動物(Ind),ako,value,ほ乳類(Ind)).
frame(草食動物(Ind),ako,value,ほ乳類(Ind)).
frame(ほ乳類(Ind),ako,value,動物(Ind)).
frame(鳥類(Ind),ako,value,動物(Ind)).
frame(ほ乳類(Ind),特徴,if_needed,毛がある).
frame(ほ乳類(Ind),生殖,if_needed,子供を産む).
frame(ほ乳類(Ind),育児,if_needed,乳を与える).
frame(は虫類(Ind),ako,value,動物(Ind)).
frame(は虫類(Ind),生殖,if_needed,卵を産む).
frame(鳥類(Ind),特徴,if_needed,羽を持つ).
frame(鳥類(Ind),能力,if_needed,飛ぶ).
frame(鳥類(Ind),生殖,if_needed,卵を産む).
frame(肉食動物(Ind),食生,if_needed,肉を食べる).
frame(肉食動物(Ind),特徴,if_needed,牙を持つ).
frame(肉食動物(Ind),特徴,if_needed,爪を持つ).

frame(肉食動物(Ind),特徴,if_needed,目が前方に付いている).
frame(草食動物(Ind),食生,if_needed,草を食べる).
frame(草食動物(Ind),特徴,if_needed,蹄を持つ).
frame(草食動物(Ind),特徴,if_needed,反すうする).
frame(しまうま(Ind),ako,value,草食動物(Ind)).
frame(ライオン(Ind),ako,value,肉食動物(Ind)).
frame(とら(Ind),ako,value,肉食動物(Ind)).
frame(ペンギン(Ind),ako,value,鳥類(Ind)).
frame(ひばり(Ind),ako,value,鳥類(Ind)).

frame(ライオン(Ind),体色,if_needed,黄褐色).
frame(ライオン(Ind),特徴,if_needed,おすにたてがみがある).
frame(ライオン(Ind),分布,if_needed,アフリカに住む).
frame(ライオン(Ind),暮らし,if_needed,群れで暮らす).
frame(とら(Ind),体色,if_needed,黄褐色).
frame(とら(Ind),特徴,if_needed,黒い縞がある).
frame(キリン(Ind),体色,if_needed,黄褐色).
frame(キリン(Ind),特徴,if_needed,黒い斑点がある).
frame(キリン(Ind),特徴,if_needed,長い).
frame(キリン(Ind),特徴,if_needed,長い).
frame(しまうま(Ind),特徴,if_needed,黒い縞がある).
frame(しまうま(Ind),暮らし,if_needed,群れで暮らす).
frame(ペンギン(Ind),能力,restrict,negrect:飛ぶ).
frame(ペンギン(Ind),能力,if_needed,泳ぐ).
frame(ひばり(Ind),鳴き声,if_needed,ピーチクパーチク).
```

図4-6　動物に関するフレーム

鳥類	
ako	動物
特徴	羽を持つ
能力	飛ぶ
生殖	卵を産む

ペンギン	
ako	鳥類
能力	negrect(飛ぶ)
能力	泳ぐ

ひばり	
ako	鳥類
鳴き声	ピーチクパーチク

図4-7　動物に関するフレームの一部を分かりやすい表記に直したもの

(1) fget 述語の機能

「ひばり」の個体として「ピー子」がいるとする。図4-6では、

ピー子
ako　　ひばり

となっているが、これは「ピー子」が「ひばり」の具体例であることを示している。前節で説明した「individual フレーム」がこの「ピー子」のフレームに該当する。この「ピー子」の能力について知りたい場合は

?-fget (ピー子, 能力, X).

と入力する。「ピー子」の上位の「ひばり」フレームには「能力」に関する記述はないが、そのさらに上の「鳥類」フレームには「飛ぶ」という能力について記述されている。このようにfget述語は、ある概念にスロット値が記述されていない時には、自動的に上位概念からスロット値を引き出すように働くのである。「ひばり」や「鳥類」フレームのファセットの所には、if_neededと名付けられたものがある。これが、if_neededデモンと呼ばれるもので、あるスロットを満たす属性値が記述されていない時には、デモンが起動して値を引き出し、その値を新たなフレームとして登録するのである。if_neededデモンは、このように必ず、フレーム登録という処理を伴っているのである。fgetで「ピー子」の能力を問い合わせた後の、「ひばり」のフレームは

ピー子
ako　　：ひばり
能力　　：飛ぶ

というように「能力」スロットが追加されている。

動物などの知識を表現するとき、フレームのako関係を利用すれば記述を経済的にできるという利点があるが、例外的な属性値にぶつかったときに困った問題が生じる。例えばほとんどの鳥類は飛ぶが、ペンギンは飛べない。従って、まともに解決しようとすると、鳥の全ての種類に関し飛べるか飛べないかを記述しなければならない。これでは記述が冗長になってしまう。こういった問題を解決するために、多くのフレームシステムでは属性の継承のしかたを細かく指定できるようにしている。今回のシステムでも、上位の概念から属性を制限する「restrict」ファセットを用意した。今回の場合は

スロット名,restrict,negrect: スロット値

という構文で上位概念からの継承を行なわないように指定することができる、といったごく単純な継承制限しかできないが、拡張することも可能である。

今回は、ペンギンは「飛べない」ということを記述するのに、restrictファセットを用いた。ペンギンの個体である「ペン君」に関し、

fget (ペン君,能力,X).

とたずねると、まず

not (飛ぶ)

を返す。他の解をさらにたずねると

泳ぐ

を返し、それ以上の解は見つからない。この場合は、鳥類の「能力」－「飛ぶ」という属性はペンギンには継承されなかったのである。

(2) **if_added デモン**

フレームの属性が追加された時に自動的に実行されるデモンである。今回の例では、属性が追加されたときにメッセージを出力することに用いている。

「ライオン」に「おすがたてがみをもつ」という特徴属性を付加してみよう。**図4-8**は、フレームシステムのスロット値を付加するための fput 述語と属性付加の結果を示している。fput 述語の中では、まだそのスロット値が ako 階層の中に登録されていないことを確認している。通常 if_added デモンは、一貫性を確認、保持するために用いられる。特に ako 階層の上位のスロットに値を付加するときは、下位のスロットにどのように属性が継承するかを指定しなければならない。

```
?-fput(しし丸、特徴、おすがたてがみをもつ).

しし丸　の　特徴　として　おすがたてがみをもつ　を追加しました
```

図4-8　fput 述語の使用例と if_added デモン

(3) **if_removed デモン**

フレームの属性が削除されたときに起動するデモンである。if_added デモンと同様、今回の例では属性が削除されたときにメッセージを出力することに用いている。このデモンも fput と同様な問題を含んでいる。ako 階層の上位のスロット値の削除が、下位のフレームに影響を与えてしまうからである。上位フレームのスロット属性が記述されている場合は、上位フレームにさかのぼって、その属性を消してしまうというわけにはいかない。その上位のフレームには、複数の下位フレームがぶら下がっているときには、ユーザーの予期しない影響が現れてしまうからである。本来こういった場合は、属性の継承を制限することで対処する。**図4-9**は、「ライオン」フレームの「分布」というスロットの値を fremove 述語により取り除き、別の値に変える例である。

```
frame(しし丸, 分布, vale).

というフレームがある場合、

?-fremove(しし丸の場合, 分布, value),
  fput(しし丸, 分布, [アフリカ, インド]).

しし丸　の分布の値アフリカを削除しました。
しし丸　の分布として[アフリカ, インド]を追加しました。
```

図4-9　if-removed デモンの使用例

4.5　電気回路に関するフレームシステム

次に、電気回路についてフレームで記述し、いくつかの推論を行なわせることを考えてみよう。フレームシステムの利用の仕方としては、電気回路に関し、いくつかのスロット(属性)値が分かっているときに未知の属性を推論するということに、絞って考えていくことにする。

4.5.1　基本的機能

フレームシステムで表現できるのは、オームの法則だけで計算できる簡単な回路である。**図4-10**にあるような、ブリッジ回路などは扱うことはできない。また、回路構成はあらかじめ part-of 階層のフレームで与えられていなければならない。

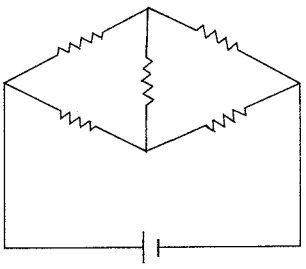

図4-10　ブリッジ回路の例
(この章のシステムでは扱えない)

このプログラムはオームの法則が適用できる回路であれば、どんなに複雑な回路でも対応することができる。その場合、推論プログラムを変える必要は全くない。この点は、フレーム推論の優れた点である。

図4-11が、例として扱う回路である。図4-11を表現したフレームを**図4-12**に示す。ここでは、各配線が直列回路に含まれるのか、あるいは並列回路に含まれるのかを記述したフレームは、既に与えられているものとする。配線がどんな回路に含まれるかは、part-of 階層および ako 階層を組み合わせて表現してある。

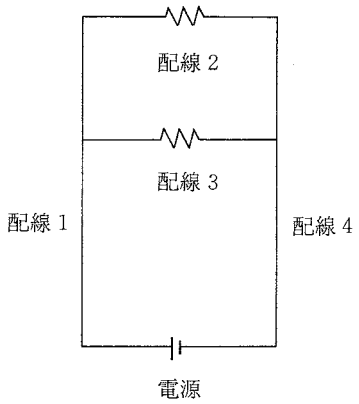

図4-11　電気回路の例

```
frame(配線(1),part_of,value,回路1).
frame(配線(2),part_of,value,回路(g2)).
frame(配線(3),part_of,value,回路(g2)).
frame(配線(4),part_of,value,回路1).
frame(回路1,ako,value,直列回路(回路1)).
frame(回路1,入力端子,value,－極1).
frame(回路1,出力端子,value,＋極1).
frame(回路(g2),ako,value,並列回路(回路(g2))).
frame(回路(g2),入力端子,value,pin1).
frame(回路(g2),出力端子,value,pin2).
frame(回路(g2),part_of,value,回路1).
```

図4-12　回路構成のフレームシステム

電気回路の配線は、**図4-13**に挙げるような ako 階層を形成している。

また、起電力などの最初に与えられている individual フレームを**図4-14**に示す。4本の配線の端子および抵抗値、電池の起電力は既に分かっているものとする。

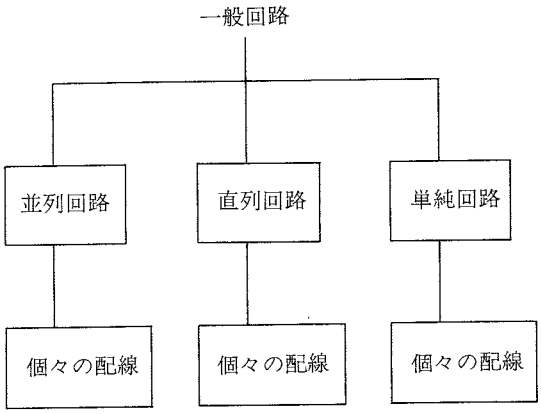

図4-13　図4-12の ako 階層を図示したもの

```
%****************%
%                各配線  %
%****************%
frame (配線(1),ako,value,単純回路(配線(1))).
frame (配線(1),抵抗,value,1).
frame (配線(1),入力端子,value,－極１).
frame (配線(1),出力端子,value,pin1).
frame (配線(2),ako,value,単純回路(配線(2))).
frame (配線(2),抵抗,value,2).
frame (配線(2),入力端子,value,pin1).
frame (配線(2),出力端子,value,pin2).
frame (配線(3),ako,value,単純回路(配線(3))).
frame (配線(3),抵抗,value,2).
frame (配線(3),入力端子,value,pin1).
frame (配線(3),出力端子,value,pin2).
frame (配線(4),ako,value,単純回路(配線(4))).
frame (配線(4),抵抗,value,3).
frame (配線(4),入力端子,value,pin2).
frame (配線(4),出力端子,value,＋極１).
%****************%
%                各端子  %
%****************%
frame (pin1,ako,value,端子(pin1)).
frame (pin2,ako,value,端子(pin2)).
frame (－極１,ako,value,電池－(－極１)).
frame (＋極１,ako,value,電池＋(＋極１)).
%****************%
%                全回路  %
%****************%
frame (回路１,ako,value,全回路(回路１)).
frame (回路１,起電力,value,10).
```

図4-14　最初に与えられている individual フレーム
(図4-12に含まれるものは別)

記述の仕方には若干の制限がある。直列回路を構成する配線や回路はいくつあってもよいが、並列回路は2つの配線もしくは回路で構成されているものとして記述しなければならない。3つ以上の配線からなる並列回路は、並列回路が別の並列回路に含まれるという記述で分けて表現する必要がある。

4.5.2　推論の手順

まず、全回路の合成抵抗を求め総電流を求める事を目標とする。全回路のスロット値が分かった後は、各回路を流れる電流や、端子電圧などを次々に求めていくようにする。最初に分かっているスロット値から、未知のスロット値を導き出せるようなデモンを、いかにうまく組み込むかがシステムの機能の優劣の分かれ目となる。

図4-15に電気回路に関する推論を行なうためのフレームシステムを示す。このフレームシステムと図4-5のフレーム検索述語、図4-12の回路構成フレーム、図4-14のindividualフレームを併せてコンサルトすると、推論を行なわせることができる。

```
%*********************************%
%                                 %
%        フレームシステム         %
%                                 %
%*********************************%
%************************%
%        全回路フレーム  %
%************************%
frame(全回路(Cir),電流,if_needed,I) :-
        fget(Cir,抵抗,R),
        fget(Cir,起電力,E),
        I is E / R.          /*I は電流値を表す変数*/
%************************%
%        端子フレーム    %
%************************%
frame(電池－(Pin),電圧,if_needed,V) :-
        fget(Cir,入力端子,Pin),
        fget(Cir,起電力,V).
frame(電池＋(Pin),電圧,if_needed,0).
frame(端子(Pin),電圧,if_needed,V) :-
        fget(Cir,出力端子,Pin),
        fget(Cir,出力電圧,V).
%************************%
%        回路フレーム    %
%************************%
frame(一般回路(Cir1),電流,if_needed,I1) :-
        fget(Cir1,part_of,Cir),
        fget(Cir,ako,並列回路(Cir)),
        fget(Cir2,part_of,Cir),
```

```
        Cir1 ¥== Cir2,
        fget(Cir,電流,Iall),
        fget(Cir1,抵抗,R1),
        fget(Cir2,抵抗,R2),
        I1 is (R2 * Iall) / (R1 + R2).
frame(一般回路(Cir1),電流,if_needed,I) :-
        fget(Cir1,part_of,Cir),
        fget(Cir,ako,直列回路(Cir)),
        fget(Cir,電流,I).
frame(一般回路(Cir),電圧降下,if_needed,V) :-
        fget(Cir,抵抗,R),
        fget(Cir,電流,I),
        V is R * I.
frame(一般回路(Cir),入力電圧,if_needed,IV) :-
        fget(Cir,入力端子,P),
        fget(P,電圧,IV).
frame(一般回路(Cir),出力電圧,if_needed,OV) :-
        fget(Cir,入力電圧,IV),
        fget(Cir,電圧降下,V),
        OV is IV - V.
%********************************%
%       並列回路フレーム         %
%********************************%
frame(並列回路(Cir),ako,value,一般回路(Cir)).
frame(並列回路(Cir),抵抗,if_needed,R) :-
        fget(Cir1,part_of,Cir),
        fget(Cir2,part_of,Cir),
        Cir1 ¥== Cir2,
        !,
        fget(Cir1,抵抗,R1),
        fget(Cir2,抵抗,R2),
        R is (R1+R2) / (R1*R2).
%********************************%
%       直列回路フレーム         %
%********************************%
frame(直列回路(Cir),ako,value,一般回路(Cir)).
frame(直列回路(Cir),抵抗,if_needed,R) :-
        kai_list(R,(fget(C,part_of,Cir),fget(C,抵抗,R)),R_List)
        sum(R_List,R).
%********************************%
%       単純回路フレーム         %
%********************************%
frame(単純回路(Cir),ako,value,一般回路(Cir)).
%********************************%
%       ユーティリティー         %
%********************************%
append([],X,X).
append([A|X],Y,[A|Z]) :-
        append(X,Y,Z).
for_all(P,Q):-
        P,
        (Q->
```

```
                fail;
                !,fail).
for_all(P,Q).
kai_list(X,Pr,Return):-
        asserta(kai([])),
        for_all(Pr,
                (retract(kai(Y)),
                 append([X],Y,Z),
                 asserta(kai(Z)))),
        retract(kai(Return)).
sum([],0).
sum([X|Y],N) :-
        sum(Y,N1),
        N is X+N1.
/*
ex1 :-
        fget(回路1,抵抗,R),
        write(R).
ex2 :-
        fget(回路1,電流,I),
        write(I).
ex3 :-
        fget(回路(g2),電流,I),
        write(I).
ex4 :-
        fget(配線(1),電流,I),
        write(I).
ex5 :-
        fget(配線(2),電流,I),
        write(I).
ex6 :-
        fget(配線(2),電圧降下,V),
        write(V).
ex7 :-
        fget(pin1,電圧,V),
        write(V).
ex8 :-
        fget(pin2,電圧,V),
        write(V).
ex9 :-
        fget(＋極1,電圧,V),
        write(V).
*/
```

図4-15　電気回路に関するフレームシステム

この節の例は、Prolog-KABAでも動作するように、全ての値は整数で求まるようになっている。しかし、他の例にも適用する場合は実数をサポートする必要がある。その場合は、RUN/PROLOGのように実数をサポートする

Prolog 処理系に移植することが望ましい。**図4-16**に RUN/PROLOG に移植する場合の変更点を示す。

```
%********************************%
%        ユーティリティー変更点  %
%********************************%
for_all(P,Q):-
        P,
        ifthenelse(Q,fail,(!,fail)).
for_all(P,Q).
```

図4-16　RUN/PROLOG に移植する場合の変更点

4.6　電気回路に関する推論の実行例

図4-17がこのフレームシステムを用いて各配線を流れる電流や端子電圧、各回路の合成抵抗値を推論させた経過である。

```
| ?-fget(回路１,抵抗,R).
R       = 5
yes
| ?-fget(回路１,電流,I).
I       = 2
yes
| ?-fget(回路(g2),電流,I).
I       = 2
yes
| ?-fget(配線(1),電流,I).
I       = 2
yes
| ?-fget(配線(2),電流,I).
I       = 1
yes
| ?-fget(配線(2),電圧降下,V).
V       = 2
yes
| ?-fget(pin1,電圧,V).
V       = 8
yes
| ?-fget(pin2,電圧,V).
V       = 6
yes
| ?-fget(＋極１,電圧,V).
V       = 0
yes
```

図4-17　実行例

最初に回路について分かっている値は、各配線の抵抗値だけである。システムは、各直列回路、並列回路の合成抵抗、および各配線を流れる電流値、端子電圧を if_needed デモンを活用して求める。例えば、システムに対し「端子－pin1」の電圧を

```
?-fget (pin1,電圧,X).
```

とたずねると、システムは「端子－pin1」を出力端子とする配線の入力端子の電圧、およびその配線における電圧降下をもとに、出力端子における電圧を計算する。その時、配線における電圧降下値が value ファセットとして登録されてなければ、さらに別の if_needed デモンを起動して、配線の抵抗や電流から電圧降下を計算して、新たなフレームとして登録する。その時また、電流が求まっていなければ全回路を流れる電流から、直列回路や並列回路における分電流を求める計算式を利用して推論する。

このように、推論は if_needed デモンの系列的な呼び出しによってなされるのである。呼び出しの停止点は、すでに value ファセットが登録されていて、if_needed デモンを起動する必要のないフレームということになる。このフレームシステムを作る時は、デモンの呼び出しが無限ループに陥らないようにするという点に最も気を使う。たとえば、電流、電圧、抵抗値は

```
電圧　＝　抵抗　×　電流
```

となっている。電圧を求めるのに電流と抵抗を用いるデモンが、電流を求めるデモンを起動し、それが元の電圧を求めるデモンを呼び出してしまう、といった無限ループを避けなくてはならないのである。

そのためには、ある程度値を求める順番を決めた上でフレームシステムを構築しなければならない。このように、フレームシステムは、純粋に宣言的な記述システムではなく、手続き的な側面も持っていることが分かる。

4.7　システムの評価

フレーム検索の部分のインプリメントは、非常に簡単である。難しいのはフレームシステムの構築である。その労力の大部分はデモンを用いた推論の手順の構成の部分に当てられた。

フレーム推論で用いている回路は配線4本の簡単な回路だが、もっと複雑な回路も無論扱える。ただし、パソコン上の Prolog では、計算量はかなり大きくなる。実行時間の点からいってパソコン上のシステムとしては少々荷が重い。

フレームシステムの利点は、データとなるindividualフレームと推論を行なうフレームシステムとが完全に切り離されている点であろう。この電気回路に関する推論システムは、対象とする電気回路に依存しない汎用のものになっている。対象とする電気回路のフレーム記述さえ与えてやれば、推論システムは共通に使うことができる。

また、フレームシステムはスロット値の参照、追加、削除という基本的な3種のアクセスにより種々の推論を行うことができるというのが一つの魅力である。ちなみに、プロダクションシステムでも、作業記憶に対する参照、追加、削除といった3つのアクセスメソッドで推論を行なうようになっている。

しかしながら、フレームシステムの構築は、プロダクションシステムと比較すると、たいへん困難なものとなっている。それは、フレームのスロットがどのフレームから呼ばれどのフレームに値を知らせるか、というアルゴリズムの流れを意識して定められていることに原因している。つまり、フレームシステムは完全な宣言型言語のパラダイムに乗っておらず、手続き型言語の側面を多分に持っているのである。知識表現が手続き的であるということは、問題領域の変化にあわせて、フレームやスロットの構成を変えてやらなければならないということになる。

最近の知識表現では、このようなフレームシステムの欠点を修正しようといった試みがみられる。例えば、表現をより宣言的にして、問題領域や問題の前提が変わっても知識を書き換えなくてもすむようなものが出現している。