数理解析研究所講究録
第525巻 1984年 192-212

ＫＡＵＳによるグラフの知識表現と基本操作

東大工．境界研　　山内　平行　（Hiroyuki Yamauchi）
東大工．境界研　　大須賀節雄　（Setsuo Ohsuga）

１．はじめに

コンピュータの諸技術の発展とともに、各種問題解決のための一手法として、知識ベースシステムによるモデルの構築とその評価に関する研究が重要となっている。たとえば、機械および建築物などの３次元ＣＡＤのためのモデル、または人文、社会、経済、工学、および科学における複雑かつ大規模なネットワーク・モデルなど、様々な分野における対象のモデル化とそのモデルに基づく評価のしかたが重要となっている。

知識ベースシステムによるこのような対象（物）のモデル記述には、対象物の内部構造関係の表現と、個々の対象物の属性および対象物間の関係記述が含まれる。また、特定の条件を満たす対象物を生成するための規則も含まれる。一方、記述されたものを「知識」とし、これらによって対象モデルの解析、および生成・評価を一貫した方式で行なうための「推論系」または「変換系」が知識ベースシステムに要求される。

KAUSはこのような基本要求を満たすべく設計がなされており、構造化された世界とその上で定義された多層論理式によってモデルが記述される。筆者ら大須賀研究室では、現在、その第４版を作成中である。

ここで、対象のモデル化に際しては、グラフ理論的アプローチを用いることができる。すなわち、グラフ理論は多くの問題分野に対する抽象モデルと考えられ、その応用分野は非常に広いことが知られている。また、グラフは代表的な構造データの１つであり、グラフ理論におけるグラフに関する諸概念および諸定理が明確に体系づけられており、特定の専門分野に対する知識のコード化の問題の例として好適なものである。さらに、筆者らはKAUSによる「ソフトウェア開発におけるプログラムの自動合成」という問題を取扱う計画があり、本稿は知識ベース内にあるグラフ理論に関する諸知識を、グラフ・アルゴリズムの自動合成の問題にどのように利用するかという研究の一環としても位置づけることができる。

以下の章では、このような動機および目的のもとで、(1) グラフの構造表現、(2) グラフ理論の知識ベース化、(3) グラフの解析・合成のための基本操作について述べ、最後に、グラフと3 次元モデリング(多面体) との関係について簡単に述べる。

2．グラフの構造表現

KAUSでは、モデル記述に必要な語いおよびデータは、集合または集合の要素という形で構造化される。すなわち、概念名、個別名および識別子などは、集合あるいは集合要素の名前を表わすものとする。このとき、べき集合の概念が使われる。ある集合（これを基底集合と呼ぶ）のべき集合とは、基底集合のすべての部分集合を要素とする集合である。また、対象がどのようなものから組立てられているかということを表わすために、対象（物）の部分構造－全体構造という内的関係表現が別のデータ構造として表現される。この表現は構造要素間に空間的順序関係や結合関係が内在していることを明示する。多くの設計対象は、このような構造表現で与えることができる。

さらに、一様な多量のデータは、通常、データベース化して格納する。KAUSでは、一様な関係データは直積集合の要素、関係データ・ファイルは直積集合のべき集合の要素として定義する。

ここで、以上のような一般的規則に従ってKAUSによるグラフの表現を考えて見よう。

無向グラフは次のように定義することができる。

定義1. グラフGは、頂点の集合Xと辺の集合Eの組で定義される。

定義2. 辺は、頂点集合を基底集合とするべき集合の要素である。

定義3. 辺の集合は、頂点集合の2 階のべき集合の要素である。

定義4. グラフGは、２頂点とその間の辺の数の３組データの集合である。

定義5. 辺は、2 つの頂点の組データである。

定義6. 辺の集合は、2 つの頂点の組データの集合である。

定義7. 辺は、2 頂点を端点として持つ。

これらの定義において、最もプリミティブな対象は頂点である。また、ここで注意しておくことは、定義1,2,3,は、通常の１階グラフに対してのみでなく、高階グラフ(hypergraph)に

対しても適用できることである。高階グラフにおいては、辺は頂点の部分集合に一般化されている。通常のグラフの辺は、要素の数が2 である頂点集合であると言える。これが定義7である。定義4,5,6 は、グラフを直積集合の要素、すなわち、関係データベースで表現するばあいに適用できる。ただし、孤立点を含むグラフはこの形式では表現できない。2 点間の辺の数が問題となるときは有用である。

次に、有向グラフについては、辺(edge)の集合の代わりに弧(arc) の集合を対象とする。すなわち、辺の向きを考える。この辺の向きについては、次のように定義する。

定義8. 辺の向きは、データの格納順序で定義する。

KAUSの集合－要素表現における集合要素は、テーブル形式で表現される。したがって、弧の始点は、テーブルの最初の要素で、終点は２番目の要素となる。このときの集合－要素表現は、厳密には部分構造－全体構造表現であると見なされる。関係データベースに対しては、欄の順序によって決めることになる。

以上の定義は、グラフの最も基本的な定義である。これらの定義の中で、頂点(vertex)、辺(edge)およびグラフ(graph) という語いが使われている。これらの語いは定義に従って、図1 のように構造化する。図において、記号∈は集合－要素関係を、＊はべき集合関係を、そして▷は構造要素関係を表わす。□ で囲んだ語は基底集合名である。

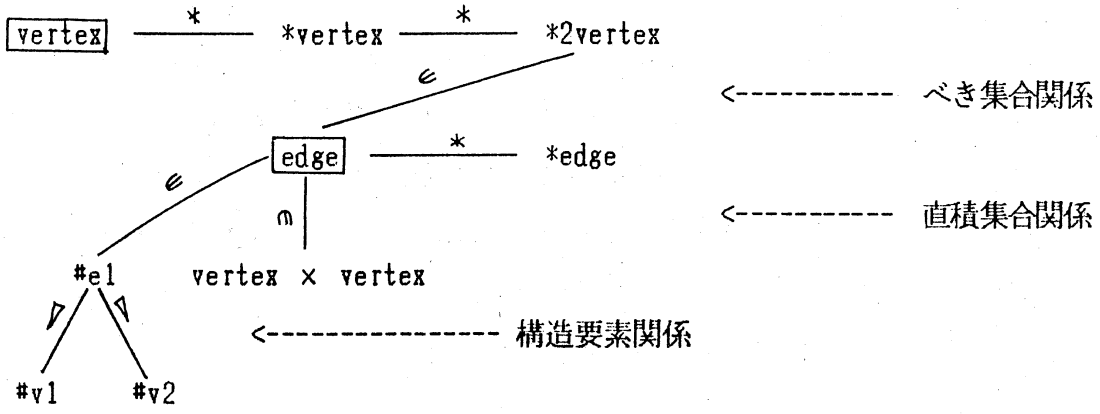


図１．グラフの構造表現(1)

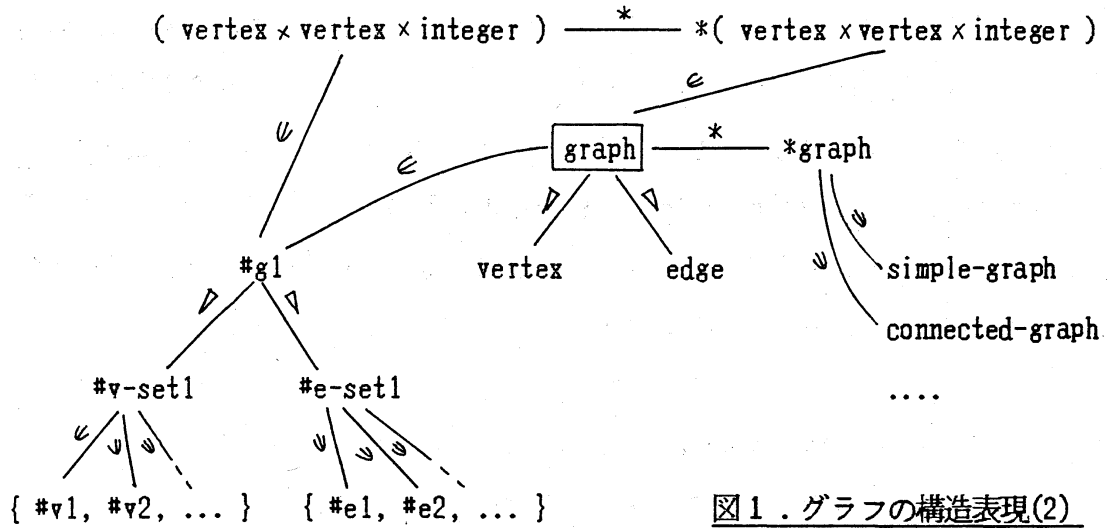


図１．グラフの構造表現(2)

グラフは、その頂点の結合状態によって、単純グラフ、連結グラフ、平面グラフ、および完全グラフなどの種々のグラフに分類できる。したがって、与えられたグラフが単純グラフであるばあいには、それを単純グラフの集合の要素として表現するのが適当である。また、グラフには、頂点集合を互いに素な２つの頂点集合に分割し、同一の頂点集合に属する頂点間には結合関係がないとする「２部グラフ」がある。したがって、２部グラフは、図２のように構造化する。一般に、ｋ部グラフはｋ個の頂点集合に分割し、同一の頂点集合の要素間には結合関係がないと定義されるので、ｋ部グラフの構造要素はＫ個の頂点集合と１つの辺の集合となる。ここで、「２つの頂点集合が互いに素で、同一の頂点集合に属する頂点間には辺が存在しない」という制限は、ｋ部グラフの制約条件として規則化し、知識ベースに組込んでおくことが必要である。

グラフに対する制約条件は、このほかにも各種考えられる。これらの制約条件は、特定のグラフ・データの入力の無矛盾性のチェックのために、および特定の条件を満たすグラフの生成のために利用される。

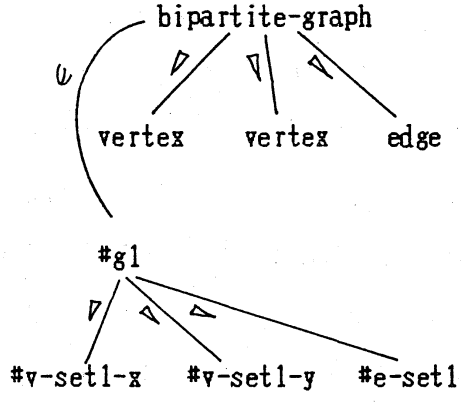


図２．２部グラフの表現

３．グラフ理論の知識ベース化

前章では、KAUSのデータ構造によりグラフの基本構造をその頂点集合と辺の集合によっ

て表現するしかたについて述べた。グラフ理論では、このグラフの頂点集合と辺(弧) の集合を基にして、ある特定の制約条件を満たすそれらの部分集合を考え、種々の基本的概念を定義している。ここで、グラフ理論に使われるすべての定義および定理について述べることはできないので、ここではグラフ理論で重要な位置を占める閉路（サイクル）に関する定義および定理を例に取って考えてみることにする。議論の都合上、ここでいうグラフは、辺の向きが与えられ連結した有向グラフと仮定する。

定義9. グラフG=(X,U) において、U の部分集合S のすべての要素の順序列$\mu=(u_1, u_2,\cdots,u_k)$ が次の条件を満たすとき、μをG のサイクルと呼ぶ。

(1). 各u_i $(1<i<k)$の両端点は、その前後の列要素u_{i-1} およびu_{i+1} の端点である。

(2). S の同じ要素がμ中に2 回以上現われない。

(3). μ の始点と終点が一致する。

定義10. 定義9 において、μが初等的なサイクル(elementary cycle)であるとは、さらに次の(4) を満足するときをいう。

(4). 同じ頂点を2 度以上通らない。

ここで上の定義において、μ上の弧をその方向に沿ってすべて辿ることができるときは、サイクルを閉路(circuit) と読み換えることができる。また、グラフが無向グラフのとき、上の定義のサイクルを閉路と読み換えることができる。

通常、このような非形式的な記述による知識を知識ベースに組込むには、これを厳密な形式的記述に変換（コード化）することが必要である。以下で定義9 および10のKAUSによるコード化を考えて見よう。

グラフ理論のコード化で、しばしば問題となるのは、グラフの辿り方をどのように表現するかである。また、一般の集合の要素の間には、普通、順序が定義されないことに注意しなければならない。したがって、定義9 は、S の要素の順列を考え、その中に定義を満たすものがあればよいが、実際的には次のように定義できる。

p-1. 集合S の要素がすべて異なったものか調べる。

p-2. 集合S のコピーであるT を作る。

p-3. T の最初の要素u1を取出し、u1の１端点v1を出発点とする。次に、u1を通ってu1のもう１つの端点v2に達する。そのあと、v2を出発点としてT を辿る。

p-4. すべてのT の要素を辿るためにp-3 を繰り返えす。

これにもとづいてコード化すると次のようになる。

```
(∀S,T/*edge)(∀U1,U2 edge)(∀V0,V1/vertex)
{ (cycle S) ← (all-different S) & (copy S T) &
              (get-e U1 T 1) & (get-c V0 U1 1) &
              (traverse V0 V0 U1 V1 T) }.          ----------------(1)
```

traverse述語は、次のように再帰的に定義できる。

```
(∀S/*edge)(∀U/S)(∀V0,V1,V2/vertex)
{ (traverse V0  V1 U V2 S)  ←  (cardinal S 1) & (find V1 U V2) } &
                               (eq V2 V0) }.          -----------------(2)

(∀S/*edge)(∀U1,U2/S)(∀V0,V1,V2,V3/vertex)(∀N/integer)
{ (traverse V0  V1 U1 V2 S) ←  (cardinal S N) & (gt N 1) &
                               (find V1 U1 V2) & (eliminate U1 S) &
                               (traverse V0  V2 U2  V3 S) }.   ----------(3)
```

ここで、(find V1 U V2)述語は、V1が与えられており、かつU の定義域がS(U/S)のとき、S の要素のうちでV1を端点とする最初の要素をU とし、U のもう一方の端点をV2とする手続き型のアトムである。また、(all-different S) はS の集合要素がすべて異なっていることを述べる述語で、これは次のように定義できる。

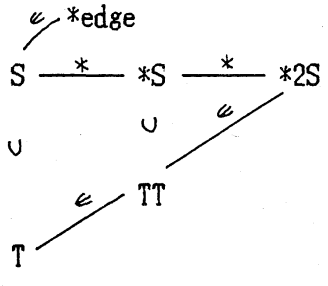
```
(∀S/*edge)(∀N/integer){ (all-different S) ← (cardinal S N) &
                                            (pair-diff S N) }.  --------(4)
(∀S/*edge)(pair-diff S 1).                                      --------(5)
(∀S/*edge)(∀X,Y/edge)(∀N,M/integer)
{ (pair-diff S N)   ← (gt N 1) & (sub M N 1) & (get-e X S N) &
                      (diff X S M)  & (pair-diff S M) }.        --------(6)

(∀S/*edge)(∀X/edge)(diff X S 0).                                --------(7)
(∀S/*edge)(∀M,N/integer)(∀X,Y/edge)
{ (diff X S N)  ← (gt N 0) & (get-e Y S N) & (ne X Y) &
                  (sub M N 1) & (diff S M) }.                   --------(8)
```

以上の(1) から(8) までにより、与えられた辺(弧) の集合S がサイクルを形成するかどうかが判定できる（稿末にKAUS-3による実際例を示す）。次に、与えられた集合S が初等的なサイクルを形成しているかを見るための定義10に対しては、次のようにコード化できる。

```
(∀S/*edge)(∀#U/S)(∀#V//U)
{ (elementary-cycle S)  ←  (reverse-cardinal V 2) & (cycle S) }    -------(9)
```

ここで、(∀#U/S)(∀#V//U) は、S の集合要素U の構造要素V で式の展開をすることを意味している。すなわち、(reverse-cardinal V 2)のAND 式に展開することを意味する。(reverse-cardinal V N)は、V を構造要素とする対象がN 個あることを示す手続き型アトムである。

これまでのcycle およびelementary-cycle述語によって、与えられた辺の集合が、それぞれ、サイクルおよび初等的なサイクルを形成しているかを解析できるが、与えられた辺の集合の部分集合のうちで、サイクルを形成するものをすべて求めるという生成の問題に対しても利用できる。すなわち、Sが与えられたとき、右図の関係から、(∃TT?/*2S)(∀T/TT)(cycle T)? なる質問を評価することにより、TTのすべての要素がサイクルを



成すようにTTに生成される。

ところで、定義9 において、S をグラフG の弧の集合U にとれば、定義9 はオイラー・サイクル(Eulerian Cycle)の定義となる。オイラー・サイクルについては、次の定理がある。

定理1 . 連結グラフがオイラー・サイクルをなす必要十分条件は、次数が奇数である頂点がないか、あっても二つだけであることである。

このコード化は次のように与えられる。

```
(∀S/*edge)(∃NN/*integer)(∀N/NN)(∀#U/S)(∀#V//U))
{ (eulerian-cycle S)  ←  (reverse-cardinal V N) & (mod 1 N2) &
                         ((cardinal NN 0) ∨ (cardinal NN 2)) }.   ----(10)
```

さらに、定義9 および10、それにグラフG の頂点集合X の要素がすべてS の端点集合と一致するという条件を満足する弧の集合は、ハミルトン・サイクル(Hamiltonian Cycle) を定義する。このハミルトン・サイクルについては、次に定理がある。

定理 2 . (Ore[1961]) n個の頂点およびm個の辺を持つ単純グラフG が、次の関係を満たすならば、G はハミルトン・サイクルを持つ。

$$m \geqq (1/2)(n-1)(n-2)+2 .$$

このコード化は次のように与えられる。

```
(∀G/simple-graph)(∀X//G)(∀U//G)(∀N,M,N1,N2,N3,N4,N5/integer)
{ (hamiltonian-cycle G)
        ←  (cardinal-c G 2) & (get-c X G1) & (get-c U G 2) &
           (cardinal X N) && (cardinal U M) &
           (sub N1 N 1) & (sub N2 N 2) & (mlt N3 N1 N2) &
           (div N4 N3 2) & (add N5 N4 2) & (ge M N5) }.   ------(11)
```

以上の例の他に、グラフ理論には、切断集合、木グラフ、マッチング、グラフの被覆など数多くの重要な概念およびそれらに関する定理がある。これらに関する知識を知識ベース化するとき、たとえば、要素の数をカウントするなど、よく使われる基本的な手続きがある。KAUSにはシステム構成要素としてプログラム・ベースがあるので、このような基本手続きをプログラム・ベースに格納しておき、Cardinalのような手続き付加の述語（これを手続き型アトムPTA と呼ぶ）により、これを利用することができる。次の章で、グラフに関する基本的な手続き付加の述語について述べる。

4. グラフのための基本操作

グラフの基本構成要素は点と辺(弧) である。グラフはこの点と辺（弧）の集合によって定義される。これらに対する基本操作には(a) 要素のカウント（b)要素の取出し、(c) 要素の付加・削除、(d) ある性質に基づく要素の並べ換えがある。また、(e) 関係データベースによる表現と部分－全体構造表現による表現間の相互の表現形式への変換が考えられる。

(a).要素のカウント

頂点の次数とかグラフの位数を求めるのに使われる。これには、図3 のように、S の集合要素とS の構造要素の数をそれぞれ求める述語(PTA) と、要素が属している集合の数を求めるものがある。頂点の次数を計算するには後者が必要である。

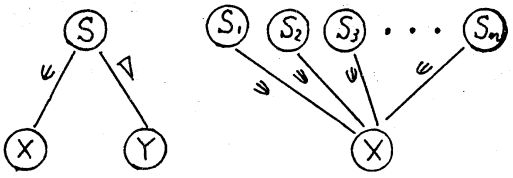


図３．集合と要素の関係

前例における(cardinal　S　N),(reverse-cardinal X N)がこの例である。

(b).要素の取出し

要素はテーブル形式で格納されている。要素には普通でいう集合要素とそれに構造要素がある。したがって、要素の取出しは、どのような集合の要素テーブルの何番目から、どのようなタイプの要素を取出すかの指定が必要である。

前例では、(get-e X S N), (ge-c X S N) が使われている。前者はS の要素テーブルのn番目の要素をX に取出すPTA で、後者は、同様にS の構造要素をX に取出すPTA である。

(c).要素の付加・削除

与えられたグラフを変形したり、ある条件を満たすグラフを生成したりするときに必要と

なるPTA である。要素の付加・削除の対象は頂点と辺(弧) がある。

頂点の削除　頂点を削除するばあいは、その頂点を端点とする辺も削除する必要がある。

辺の削除　辺の削除をするとき、その下位構造要素である頂点も削除する。ただし、頂点集合からは削除しないようにする。

頂点の付加　頂点集合の要素として単に付加するばあいと、特定の辺上に頂点を付加するばあいが考えられる。後者のばあいは、頂点集合への付加とともに、辺の分割ということが起るので、元の辺を辺の集合から削除し、分割により新たに生成される辺を辺の集合に付加することが必要である。もちろん、このばあい、辺の部分－全体構造表現も修正する必要がある。

辺の付加　辺の付加には３つのケースが考えられる。一つは、連結した一つの部分グラフとして付加するばあい、もう一つは、付加する辺の一端点を特定の頂点に一致させて行なうばあい、３番目は、両頂点を特定の頂点に一致させて行なうばあいである。たとえば、グラフ G に辺f を付加するとき、辺の集合にf を要素として新たに登録し、その辺の下位構造関係の表現を新たに行なうことが必要である。

(d).ある性質による要素の並べ変え

この基本操作は、一般的に言えば、ソーティングの問題である。グラフ理論においては、頂点の次数の大きさがしばしば問題となる。したがって、頂点集合の要素をその次数によってソートするとかすれば、定理の表現式が都合よく得られることがある。また、辺の集合をその接続関係によってソートしたりするのも有用である。この基本操作は、さらにもっとプリミティブな関数によって表現できようが、効率の問題とも絡んで手続きとして単独に定義しておけば有用であろう。

(e).データの表現形式の変換

2 章のグラフの構造表現のところで、具体的なグラフ・データは関係データベースのファイルとして表現したり、集合－要素および構造要素関係により表現したりできることを述べた。このような異種データ形式間相互の変換は、かなり大きな問題で、モデル・データの保守・管理および関係データベースの関係操作の適用を可能とするためにも重要である。

ここで、手続き付加の述語(PTA) の評価について触れておく。PTA はそれに付随する手続きの引数が定まっているときに、その手続きを実行し手続き固有の値を返す。また、すべて

のPTA はTrue/Falseの論理値を持つ。証明木にはいくつかのPTA がAND/OR結合されている。これらのPTA の評価順序は特に明示されていないので、各PTA の引数の状態を調べて、それに付随する手続きが実行可能なものから順次評価していく。ただし、AND 結合されているPTA のうちで1 つがFalse と評価されたときは当然その部分木(AND木) に対する評価は打切られるので、評価可能であっても評価されないばあいがある。その他、簡単なKAUS-3の紹介がAppendixにあるので参照されたい。

５．３次元モデリングとの関係

今まで、KAUSによるグラフの構造表現のしかたと、グラフにおける諸概念および諸定理のうちで、サイクルに関するものを例にとって、それらをコード化し知識ベースに組込むことについて述べた。これらのグラフに関する抽象的な知識は、実際的な問題に対して応用することができる。ここで、多面体による３次元モデリングについて考えて見よう。

実際の問題のグラフによるモデル化は、従来からよく行なわれているが、主に、対象の性質および関係の解析のために使われ、一般の設計問題のように特定の性質および関係を満たすモデルをコンピュータ支援によって合成または生成するという目的のためには、あまり使われていないようである。KAUSによるグラフに関する知識表現は、しかし、この対象の解析と生成の両目的のために利用することができる。したがって、KAUSを３次元モデルの設計問題に適用することができる。

３次元モデルの解析については特に問題はなさそうであるが、３次元モデルの生成をするというとき、生成とはどういうことをいうのか明確にしておく必要がある。生成と合成は厳密にはその意味が違い、３次元モデルの合成は、いわば、既知の部品を組合わせて新たなものを作るということである。それに対して、３次元モデルの生成とは、既存のオブジェクト（頂点とか辺）に未知のオブジェクトを新たに加えたり、既存のオブジェクトの一部を削除したりして、全く新しいものを作るこという意味合いが強い。また、モデルの一部を抽出するということも、広い意味での生成を意味するものと考えられる。さらに、３次元モデルでは、トポロジカルな性質のほかに、頂点の空間座標が問題となり、各頂点の空間座標を変えることにより形状の変化が起る。したがって、座標を変えることも一種の３次元モデルの生成と考えられる。

前置きが長くなったが、グラフの多面体との関係を図に示せば、図4 のようになる。この

ような多面体の構造表現および多層論理式によってモデルが定義される。

多層論理式による知識表現には(1) グラフ一般に関する知識、(2) 多面体一般に関する知識、および(3) 個々の多面体に関する個別知識がある。このうちで、3番目の知識は、モデル操作の過程で更新される性質を持っている。

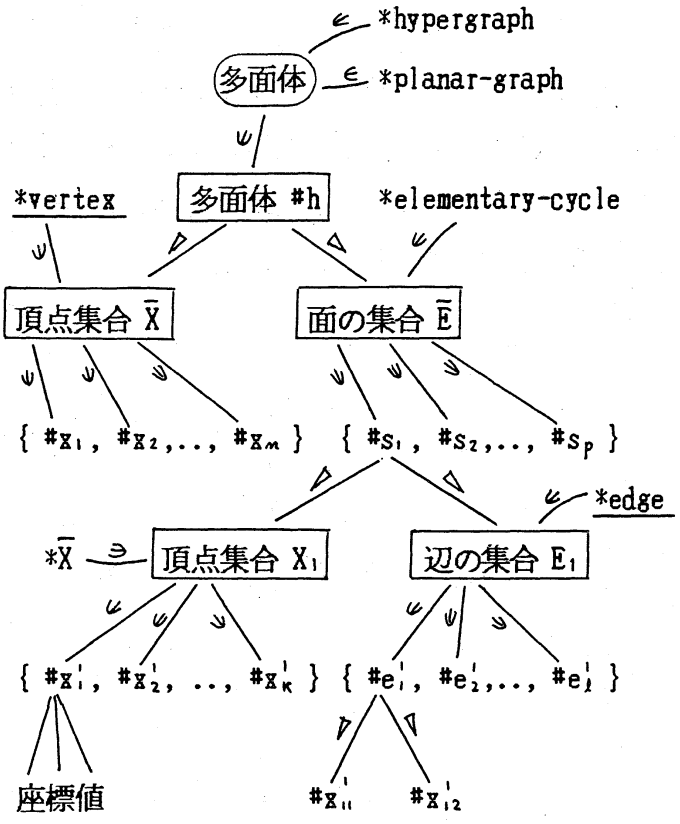


図4．多面体の構造表現

図について説明すると、多面体はその幾何学的形状を無視すれば高階グラフおよび平面グラフの一種であることを図は表わしている（平面グラフがすべて多面体として実現可能ではないことに注意しておく。高階グラフについても同様である）。抽象化して考えた多面体#hの辺の集合は、具体的には面の集合$\bar{E}$ となる。そして、面の境界線はグラフ的には初等的なサイクルをなしているものに限ることを$\bar{E} \in$ *elementary-cycle で示す。各面#s_i は、たとえば#s_1 は頂点集合X_1、および、辺の集合E_1によって定義され、X_1は$\bar{X}$ の部分集合であることを$X_1 \in$ *$\bar{X}$で示す。辺の集合要素#e'_l は、両端点として#x'_{12} および#x'_{11} を持つことを構造要素関係$\triangleright$で表わす。さらに、各頂点座標も構造要素表現を借用して、頂点の構造要素として表現する。

次に、モデルは図4 のように構造化されるとして、実際にはどのようにしてモデルを構築していくのであろうか。先に述べたモデルの生成ということからいえば、大体次のようなことになろう。まず、何もない状態から出発するとすれば、点集合を生成し、次にそのべき集合の部分集合（点の2組をその要素とする集合で辺の集合と見なされる）を生成する。これをL とし、集合L のべき集合を考え、その部分集合のうちで要素が初等的なサイクルのみからなる部分集合S を生成する。この部分集合S の要素は、一つの面（閉じた面）を形成して

いるものを一部に含む。そのうちの一つを選び、それをコピーして対応する頂点を線で結ぶことによって、一つの多面体を生成するということが考えられる。そして、生成された多面体の一部の座標を変えてモデルを変形する・・・（図5 参照）。

このような過程の中で、生成の組合わせの数をなるべく減らすために、多くの知識を利用するということが考えられる。

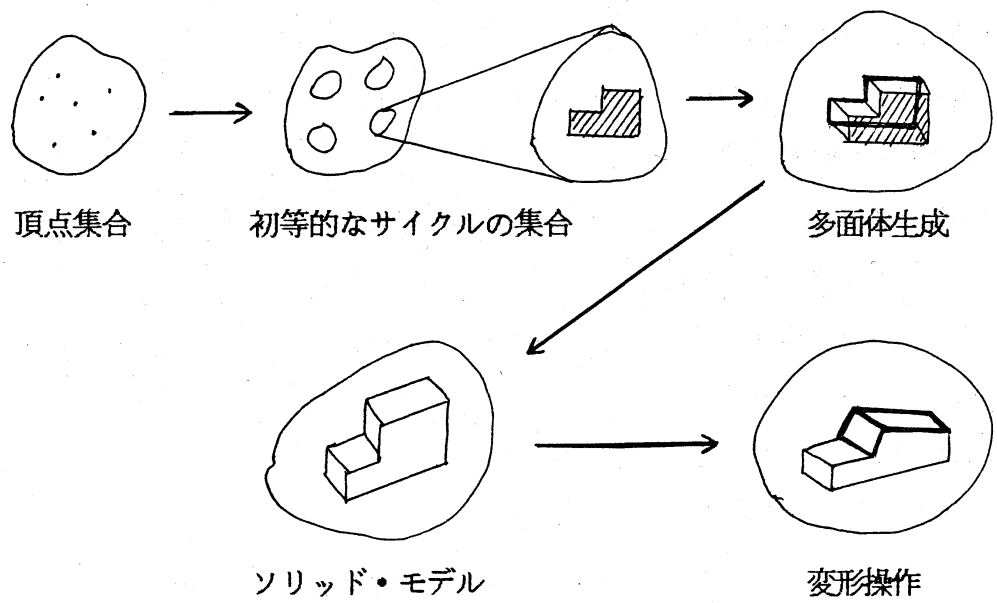


図５．３次元モデル生成

６．おわりに

KAUSによるグラフの構造表現とグラフ理論の知識ベース化について述べた。また、その一つの応用として、多面体による３次元モデルの生成について、その考え方を述べた。そのほかの各種の応用が考えられるが、本稿の目的ではないので割愛した。また、本稿は、最初にも述べたように、グラフ・アルゴリズムの合成の研究の一環としても有意義なものである。

なお、KAUS-3がCP/M86からUNIXに現在移植され、一部強化がなされて稼動中である。本稿で述べたグラフの構造表現はKAUS-3によって表現可能であるが、一部の多層論理式については構文の制約上できないものがある。KAUS-4は、KAUS-3の構文上の改善およびモデルの生成（合成）問題に対する全面的な強化を図ることにしている。

参考文献

[1]. Ohsuga, S. : A New Method of Model Description —— Use of Knowledge Base and Inference —— . in CAD system Framework (ed. Bo.K. and Lillehagen, F.M.), North Holland, pp. 285 - 312 (1983).

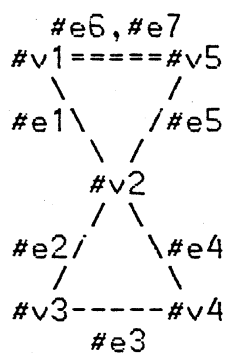
[2]. Ohsuga, S. : Predicate Logic Involving Data Structure as a Knowledge Representation Language, Proc. '83 Eighth International Joint Conf. on AI, pp. 391 - 394 (1983).

[3]. 山内, 大須賀 : 知識ベースシステムKAUSによる構造体表現と推論方式、電子通信学会、AL82 - 69, PRL82 - 57 (1982).

[4]. 河上 : 知識ベースシステムの３次元CAD への応用、東京大学修士論文 (1983)

```
%Yama27 kaus

W E L C O M E !!          [ KAUS-3 Unix Version  1983.10 ]

:: $kbg1
(sk_p *pred *3vertex *edge *graph).
(sk_e *2vertex edge).
(sk_e *3vertex *edge).
(sk_e *vertex v_set1 v_set2).
(sk_e *edge e_set1 e_set2).
(sk_e vertex #v1 #v2 #v3 #v4 #v5 #v6 #v7).
(sk_e edge #e1 #e2 #e3 #e4 #e5 #e6 #e7).

(sk_e graph g1).
(sk_c g1 v_set1 e_set1).
(sk_e v_set1 #v1 #v2 #v3 #v4 #v5).
(sk_e e_set1 #e1 #e2 #e3 #e4 #e5 #e6 #e7).
(sk_c #e1 #v1 #v2).
(sk_c #e2 #v3 #v2).
(sk_c #e3 #v3 #v4).
(sk_c #e4 #v4 #v2).
(sk_c #e5 #v2 #v5).
(sk_c #e6 #v1 #v5).
(sk_c #e7 #v5 #v1).


(sk_e *pred clear).
[AX/#e_set1](| (clear) ~(del_e X e_set1)).


(sk_e *pred cycle traverse all_different pair_diff diff).

[AT/*edge][AU1/edge][AV0/vertex][AV1/vertex]
(| (cycle T)
        ~(all_different T)
        ~(get_e U1 T 1) ~(get_c V0 U1 1)
        ~(traverse V0 V0 U1 V1 T)
).

[AS/*edge][AU/e_set1]
[AV0/vertex][AV1/vertex][AV2/vertex]
(| (traverse V0 V1 U V2 S)
        ~(count_e 1 S) ~(find V1 U V2)
        ~(eq V2 V0)
).

[AS/*edge][AU1/e_set1][AU2/e_set1][AV0/vertex]
[AV1/vertex][AV2/vertex][AV3/vertex][AN/num]
(| (traverse V0 V1 U1 V2 S)
        ~(count_e N S) ~(gt N 1)
        ~(find V1 U1 V2) ~(del_e U1 S)
        ~(traverse V0 V2 U2 V3 S)
).
```

```
[AS/*edge][AN/num]
(| (all_different S)
        ~(count_e N S) ~(pair_diff S N)
).

[AS/*edge](pair_diff S 1).
[AS/*edge][AN/num][AM/num][AX/edge][AY/edge]
(| (pair_diff S N)
        ~(gt N 1) ~(sub M N 1)
        ~(get_e X S N)
        ~(diff X S M)
        ~(pair_diff S M)
).

[AS/*edge][AX/edge](diff X S 0).
[AS/*edge][AX/edge][AN/num][AY/edge][AM/num]
(| (diff X S N)
        ~(gt N 0) ~(get_e Y S N) ~(ne X Y)
        ~(sub M N 1)
        ~(diff X S M)
).

\*  A sample graph g1 is shown
   to the right figure.          g1:
    G1 is defined by

       g1 = ( v_set1, e_set1).

    With the following test sequence,
   it can be checked whether a subset
   of e_set1 constructs a cycle.
```




```
TEST SEQUENCE:
(cycle e_set1)?
(clear)? (sk_e e_set1 #e1 #e2 #e3 #e4 #e5 #e6). (cycle e_set1)?
(clear)? (sk_e e_set1 #e3 #e4 $e2 $e1).          (cycle e_set1)?
(clear)? (sk_e e_set1 #e6 #e7).                  (cycle e_set1)?
(clear)? (sk_e e_set1 #e1 #e4 #e5).              (cycle e_set1)?
(clear)? (sk_e e_set1 #e1 #e5 #e6).              (cycle e_set1)?
*\


--EOF kbg1
::
```

```
:: (print e_set1)?

<-e_set1 :
  (-e1 :          v1   v2   )
  (-e2 :          v3   v2   )
  (-e3 :          v3   v4   )
  (-e4 :          v4   v2   )
  (-e5 :          v2   v5   )
  (-e6 :          v1   v5   )
  (-e7 :          v5   v1   )
>
YES.

:: (cycle e_set1)?
NO.

:: (clear)?
YES.

:: (sk_e e_set1 #e1 #e2 #e3 #e4 #e5 #e6).
:: (print e_set1)?

<-e_set1 :
  (-e1 :          v1   v2   )
  (-e2 :          v3   v2   )
  (-e3 :          v3   v4   )
  (-e4 :          v4   v2   )
  (-e5 :          v2   v5   )
  (-e6 :          v1   v5   )
>
YES.

:: (cycle e_set1)?
YES.

:: ((clear)?

    ^ Error : a skelton-word or a variable expected.
:: (clear)?
YES.

:: (sk_e e_set1 #e3 #e4 #e2 #e1).
:: (print e_set1)?

<-e_set1 :
  (-e3 :          v3   v4   )
  (-e4 :          v4   v2   )
  (-e2 :          v3   v2   )
  (-e1 :          v1   v2   )
>
YES.

:: (cycle e_set1)?
NO.
```

A-1. Outline of KAUS

The knowledge based system KAUS has been being revised and enlarged for availability of the structure model building and evaluation during this few years. The fundamentals of KAUS are the set-theoretic formalisms of representing knowledge and inference mechanism based on multi-layered logic. This system is now partly running under the CP/M-86 machine environment with 512 KB memories.

KAUS uses similar ways to resolution principle to infer problems. However, algorithms differ in details from resolution principle, by taking into account of the following way of representing knowledge by KAUS:

1). Knowledge base of KAUS comprises concepts structure of vocabulary, the part-whole data structure, and multi-layered predicate formulas and interrelationships among them.
2). The internal representations of formulas are not conjunctive normal forms but AND/OR trees.
3). Predicates (functions) called procedural atoms(PTAs) are used for representing knowledge, each of which is evaluated by the special procedure attached to it.

We give now some explanations about inference, retrieval and control systems illustrated in the figure.

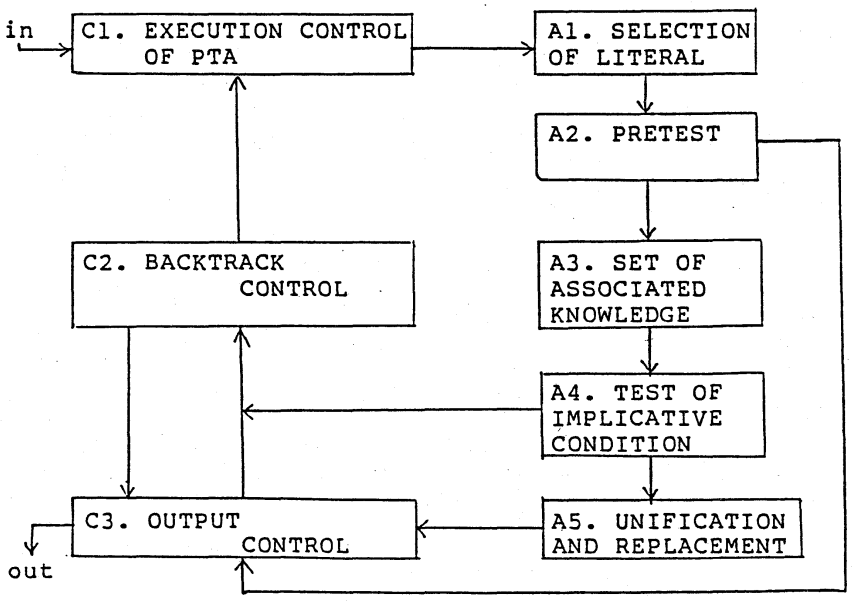


Fig.A1- Inference diagram

C1). Execution control against procedural atoms
There are two kinds of procedural atoms: built-in functions and user defined functions. When they reside as the terminal nodes in the query tree, the procedure attached to them are evoked from the program base. If all the arguments of functions necessary to execute have been instantiated, they are ready to execute and are actually executed.

C2). Backtrack control
When the proof has been failed, instantiated variables and replaced literals are recovered. Data for recovering have been put into stacks. The stacks are used for remembering instantiated variables, literals to be recovered, proved literals and knowledge units to be used to prove the literal selected from the query tree.

C3). Control to output answer
The output type is determined according to the query type. The query types are classified into YES-NO, WHAT, WHY and HOW types; among them, the last two query types are not supported in the system level, but could be requested to the system and processed in the knowledge representation level. For the query of YES-No type, the answer 'NO' could be rigidly distinguished from the answer 'I DON'T KNOW' by setting up the query mode.

A1). Selection of literal to be proved
The strategies to select a literal to be proved from the terminals of the query tree are as follows:
a). to search a literal to be proved in the depth first order,
b). to select a non-procedural atom,
c). to have a preference for a negative literal, and
d). to select a literal which has at least one constant or existentially quantified variable as the argument.
Though the selection strategies adopted will affect the efficiency of inference and the amount of memories necessary, only simple ones are used in KAUS.

A2). Pretest
The literal selected by A1 is examined whether it is provable by the literal in the query tree instead of using knowledge in the knowledge base. (this module is not yet currently implemented).

A3). Set of associated knowledge
The set of associated knowledge (formulas) is selected from the knowledge base to prove the literal selected by A1. The multi-layered predicate formulas can be accessed through their constant or existentially quantified arguments as the indices. The selected set consists of pieces of multi-layered predicate formulas which are the elements of the set of intersection of formulas associated with each of the upper concept nodes (vocabularies) to the indexing nodes and have the opposite sign to the literal to be proved.

A4). Test for implicative condition

Whether the selected knowledge (formula) implies the literal to be proved is examined according to the implicative condition shown in Tab.A1.

More precisely, it is examined whether each of the combination of the quantifiers of the corresponding variables and the set relation between the domains of the corresponding variables will satisfy the condition given in Tab.A1. The domains of variables are classified into three types, i.e., sets, individuals, and neutrals. Among them, sets(concept nodes) and individuals(instance nodes) have been predefined in the knowledge base, and set relation between given two nodes can be determined by the check diagram shown in Fig.A2. If the neutral variables are used in the formula, the formula expansion reflecting the multi-layered structure of the variables is usually necessary. We will detail this problem later in the sequel.

Q_P	Q_Q	COND.	Q_R	D_R
$\forall$	$\forall$	$D_P \supset D_Q$	$\forall$	D_Q
$\forall$	$\exists$	$D_P \cap D_Q \neq \emptyset$	$\exists$	$D_P \cap D_Q$
$\exists$	$\exists$	$D_P \subset D_Q$	$\forall$	D_P
$\forall$	—	$C_Q \in D_P$	—	—
—	$\exists$	$C_P \in D_Q$	—	—
—	—	$C_P = C_Q$	—	—

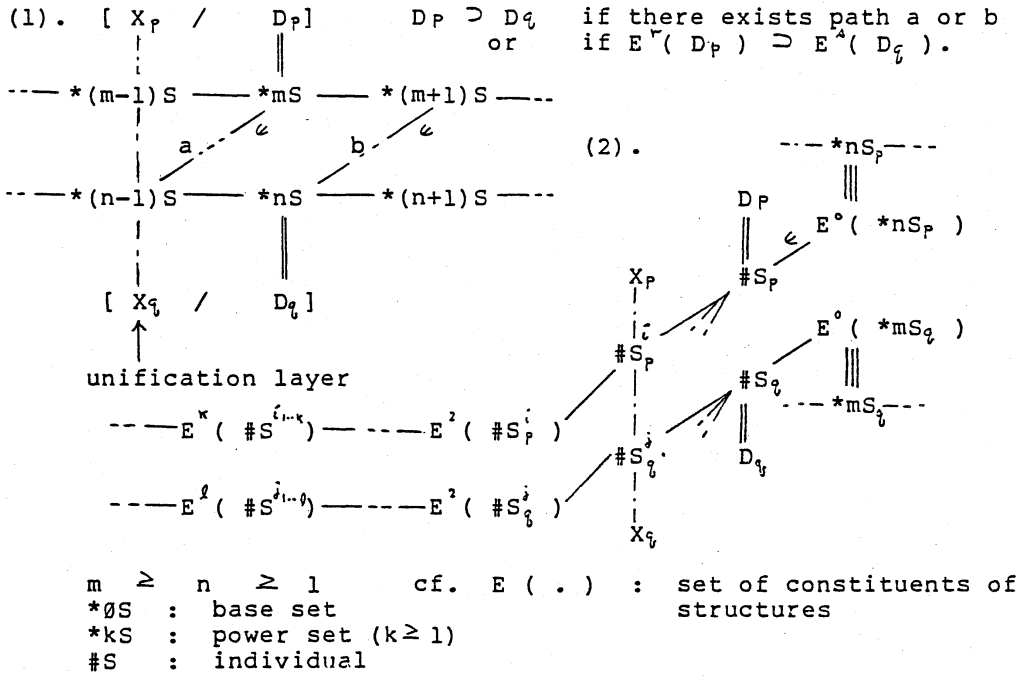
P: $[Q_P X_P / D_P] F(X_P)$ assertion
Q: $[Q_Q X_Q / D_Q] F(X_Q)$ query
R: $[Q_R X_R / D_R] F(X_R)$ replacement

Tab.A1- Implicative condition for P ---> Q

A5). Unification and replacement

The deduction tree (that becomes new query tree) is generated through instantiating variables and replacing the proved literal by the formula that has proved it. The instantiated variables and the replaced literal are kept in the stack for backtracking control.

In regard to memory management, areas of knowledge base and works for generating the deduction tree altogether use memory cells of fixed length (32 bytes) and automatic garbage collection is performed during the inference process.



m ≥ n ≥ 1 cf. E (.) : set of constituents of structures
*ØS : base set
*kS : power set (k≥1)
#S : individual

Fig.A2- Check diagram for $D_p \supset D_q$