知識依存型概念設計システムの研究*

—数理構造の知識表現と操作—

岸 義 樹**

Studies on Concept-inferred Design System Associated with Knowledges
—Representation and Inference of Numerical Design Knowledges—

Yoshiki KISHI

In this paper, knowledge representations of numerical expressions and geometric structures in the mechanical design are discussed with an experimental frame system implemented on a Prolog coded in Lisp 1. 5. It is clarified that numerical expressions are translated into frame-typed knowledges by hierarchical network with the order of calculation preserving. Moreover it is possible to insert design restrictions into the frames at any places. The inference of calculations is done with dedicated predicates called as demon-predicates according to the order of slots in the frame. A value of slot is a design variable, and in slots it is possible to direct the unit. If a slot has a unit, the unit-demon converts the slot value from the values' own unit to the directed one. In case of the geometric structures, it is found that the structures are represented into the frame-typed knowledge with the B-reps data structure of them preserving. And it is ascertained that the moving process of vectors are represented in frame by extending the methods of numerical expressions. The inference of getting vector data of the structures is executed by the top-down searching. To simplify representing structures, some prototype structures such as the square and the cube are applicable.

Key words : concept-inferred design, knowledge engineering, frame system, Prolog, geometric knowledge, calculational knowledge, numerical inference

1. 序

概念設計の一つに、“あるもの”の目的、機能、及び構造を次第に明確にしていき、全体及び各部の必要性を確認しながら、設計の大綱を決めていくことという定義¹⁾がある。この設計過程が、設計者の経験と学習に基づいた知識、論理的あるいは個性的な知識操作に大きく依存していることは言うまでもないが、知識操作の手順自体の大部分も知識であることには注意する必要がある。

さて、知識あるいは知識操作には、記号を対象にするものと形状を対象にするものがあるが、操作手順そのものは記号で表現されるのが大部分である。この意味で、現在の形状モデラは形状の知識操作の基本要素を多数含んでいるといえよう。更に、記号、形状を問わず、知識あるいは知識操作は設計の諸々の数量をそ

れ自体で意味する数値表現と、特定の属性を持たせることのできる言語表現に分けられるが、究極は、属性として数値表現を必要とする言語表現すべてに特定の数値表現を決定することが設計の目的である。

このように考えるなら、概念設計は実質的には「あるもの」を存在させるための初期環境としての言語表現と数値表現を決定することにある。良い初期環境が設定できているなら、後段の設計過程ではミスチョイスが少なくなる。これが、概念設計においてこそ知識を主体とした強力な事前評価機能が必要と考えるゆえんである。

知識（人の判断、人の持つ情報を適当なデータ構造で表現した情報。本報は frame^{2)~4)} を利用しており、文脈上混乱しない時は単に frame と表記）の扱いは相当面倒で、統一的方法で表現するのは困難と考え、機械設計という分野で、model(事例情報)型知識、rule(規則情報)型知識、formula(数式情報)型知識、space(図形情報)型知識、generic(用語情報)型知識の5種類に分類して取り扱っている⁵⁾⁶⁾。

前報⁷⁾では、model 型知識を対象として、数値表現

* 原稿受付 昭和 60 年 10 月 8 日、昭和 59 年度精機学会春季大会学術講演会（昭和 59 年 3 月 27 日）にて発表

** 正 会 員 製品科学研究所（茨城県筑波郡谷田部町東1-4）

```

[ frame_name      frame_macro
  ident_name      ident_macro
  class_name      class_macro
  inherit_name    inherit_macro
  option_name     option_macro ]

frame_name ::= string      /* frame calling */
ident_name ::= HIERARCHY /* frame hierarchy,type,indexing */
class_name  ::= CLASS     /* detail class of frame type */
inherit_name ::= INHER    /* inheritance of frame */
option_name ::= OPTION    /* other data, not always usable */

C.F. The * part of *_name is addressed as 'macro name *'.
*_macro ::= (slot.slot_list) /* for frame_macro */
*_macro ::= slot              /* for other macros */
slot_list ::= (slot.slot_list) | ()
slot       ::= (name.filler.advices) | (name)
name       ::= printable strings without '#' at head
filler     ::= (name.filler) | (filler.filler) | (variable) | ()
advices    ::= (name.parameter.advices) | ()
parameter  ::= (variable.parameter) | (name.parameter) | ()
variable   ::= printable strings with '#' at head

```

Fig. 1 Basic structure of frame

を含まない言語表現⁸⁾による機械構造の知識表現と利用を考察し、Prolog と frame⁹⁾を基礎とした手法を確立した。

本報は、formula 及び space 型知識を対象に、属性として数値表現が必要な機械構造の言語表現、いうならば機械の数理構造の知識表現とその利用について、基礎手法の確立を試みる¹⁰⁾。更に、rule 型知識の表現も与える。本報で確立する手法は概念設計のみならず、基本設計、詳細設計及び生産設計にも適用できると考える。

2. 設計式の知識表現

設計式は相当に定形的知識であるから、frame を一つの計算パッケージとみなす方向もあるが、式構造が明らかでないため、構造操作に必要な意味理解は困難である。本章では式構造を明示的に扱う方法を考察する。

2.1 知識表現の基本

知識表現のデータ構造は、図1に再録する前報の model (設計範例) 型知識の frame 構造が若干修正される。まず、推論に適応させられた知識と情報を保持する data 型知識を新しく導入する。この知識型は frame-macro の slot-name として、その slot-filler が各々ベクトル並び、文字列、数値を意味する VECTOR, STRING, SCHOLAR を用意しているが、解釈をシステムの判断に任せる時は単に DATA と表すこともできる。

ここで、frame 操作の概要を説明する。まず、frame は frame-name で特定される。同名 frame があるなら、推論によって、与えられた問題への適合性を定義順にチェックされ、パスした frame が推論実行時の知

識として利用される。以降の議論では、frame-name, frame-macro 以外は必要な時を除いて、適当に省略している。

frame system は M 380 Lisp で試作した Prolog 上で実現されている。次章以降の logic で、# 付文字は Prolog の任意のリスト、ストリングとマッチングできる変数を示す。'[...]' は Prolog program の主張 '(ASSERT (...))' の略記であり、'>' は最も近くに出現した '<' に対応してリストを構成する。この Prolog は DECIDE (Decision Environment for Concept Inferred DDesign) system の記述言語である⁹⁾。

2.2 知識の階層的表現

Frame は知識の階層的表現が容易であると共に、内容を理解しやすい表現が可能である。一方、人は知識の整理に階層的方法を取り入れているように見受けられる。数式の表現も階層性を設定することで簡潔の表現を実現できるはずである。数式の場合には explicit に階層は設定できないが、慣習的に計算の優位順序が存在しているので、それを利用する。

計算の優位順序は、括弧式 (関数も含める) ー累乗 (底又は BASE と、指数又は EXP) ー乗除 (*, /) ー加減 (+, -) で、同順位は左から右へ評価する。これに注目して、数式をこの順序に従って階層化後、frame 表現に直すことを考える。数式の整理は、数式を

$F = f(\dots)$, F : 数式名, $f(\dots)$: 演算定義部

として、演算定義部が list notation で記述されている、つまり、リストが括弧式を表すなら次の手順に従う。

0) 与えられた数式の演算定義部から点対リスト (数式名, 演算定義部) を生成し, unresolved-problem として記憶する。

1) unresolved-problem がないのなら, 整理終了。

2) unresolved-problem からリストを1個取り出す。

3) 演算定義部が同じリストは最も古い数式名に統一後, unresolved-problem を書き直し, 2) から実行する。

4) 演算定義部中にリストがあれば, それは新しい定義部として unique な数式名を与え, unresolved-problem に記憶する。そして, 同じ構造のリストを新しい数式名ですべて置き換え, 2) から実行する。

5) リストの本体が同じ優位の計算なら, そのリス

δ : deflexion, W : load, I : 2nd moment of area,
 x : load position from open side, L : length,
 E : modulus of longitudinal elasticity

Fig. 2 Design formula for open side beam

トは整理されているので, resolved-problem として記憶しておき, 1) から実行する。

- 6) 点对リスト中に異なる優位の計算があれば, 連続する最優位の計算項, 演算子から順に, 新しい演算定義部として点对リストを書き直し, 1) から実行する。

このような手順を繰り返すことによって数式は, 計算の優位順序を反映した階層化を達成できる。これを片持ちはりのたわみを表す図 2 の数式に適用すると, 図 3 のごとく, 演算子と演算項に階層化される。

frame でこの階層構造を表現するのに, frame-macro 記法として, slot-name に演算子名, slot-filler に演算子の直下の演算項名を採用するのが自然である。よって, ある演算項の子はすべてその演算項名を frame-name とする frame の slot となる。なお, 文字コードの都合により, *, /, +, - は各々 MULTI, DIV, PLUS, MINUS と書き換えている。

frame 表現の一部を図 4 a に掲げるが, #macro- i ($i=1, 2, \dots$) は, 図 4 b に示す各 frame に固有の情報で, 必要に応じ定義されているとする。PART-OF slot には, 必要なら, 階層構造上の親 frame-name と frame-id を

Fig. 3 Example of hierarchical structure of formula
 書き入れる。又, frame-macro の最後, END-OF-SLOT slot は, filler がその frame-name なら, frame の推論結果を data frame に変換する宣言である。

さて, slot には UNIT-RULE(*) という, slot-filler 中の単位を * に単位変換する専用述語 (便宜上, 単位変換 demon と呼んでおく。すべての slot-name は demon となっている。) が advice として記述できる。

```

a. Frame definition of formula
[deflexion
  ((MULTI (load) UNIT_RULE (N)
    (MULTI (POWER_1) UNIT_RULE (m^3)
      (MULTI (BRACKET_1)
        (DIV (3)
          (DIV (mole) UNIT_RULE (N/m^2)
            (DIV (sma) UNIT_RULE (m^4)
              (END_OF_SLOT (deflexion)
                UNIT_RULE (m) )
              )
            )
          )
        )
      )
    )
  )
  .#macro_221
]
[BRACKET_1
  ((PLUS (1)
    (MINUS (MULTI_1)
      (MINUS (MULTI_2)
        (END_OF_SLOT (BRACKET_1) )
      )
    )
  )
  .#macro_223
]

[load ((DATA (50 Kw). #macro_5)
  .....
  )
[POWER_1
  ((SUBSTITUTE (POWER)
    WITH ((BASE length) (EXP 3)
      (END_OF_SLOT (POWER_1)
        UNIT_RULE (m^3) )
      )
  )
  .#macro_222
]
[POWER
  ((BASE (#X)
    (EXP (#Y)
      (END_OF_SLOT (POWER)) )
    )
  .#macro_224
]

b. Detail of #macro-i
#macro_i ::= <HIERARCHY (FORMULA (date.frame_id))
/* frame_id::=(group self) | () */
PART_OF (X) /* X::=deflexion,MULTI_2, etc. */
CLASS (Y) /* Y::=MULTI/DIV,PLUS/MINUS, etc. */
INHER (Z) /* Z::=open_side_beam, etc. */
OPTION (...) /* other useful informations */
)

c. Condition frame
[COND_1
  ((MULTI (mole) UNIT_RULE (N/m^2)
    (DIV (sma) UNIT_RULE (m^4)
      (END_OF_SLOT
        (GREATER_THAN_OR_EQUAL (5 Kw))
        UNIT_RULE (N*m^2) )
      )
    )
  )
  .#macro_226
]

d. Frame for FORTRAN function
[COND_1
  ((COS ((PAR (#X)
    (FORTRAN (COS)
      (END_OF_SLOT (COS) )
    )
  )
  )
  .#macro_227
]

```

C.F. mole::=modulus_of_longitudinal_elasticity
 sma::=second_moment_of_area

Fig. 4 Structure of formula frame

この demon は MKS 単位を中間表現として、推論実行時に単位変換を行う。なお、「x/y」は「x の y 乗」を表す。

一方、POWER-1 frame 中の SUBSTITUTE (POWER) は WITH(name...) と対で、name slot の filler を... で順に置換することを意味している。その実行は推論時である。もちろん、置換を利用せず、数式をそのまま定義もできる。階層表現終端の非数値記号も frame 表現できるが、この種の frame は data frame か他の formula frame である。

slot-filler に条件を入れる時、例えば、図 4 a で荷重が 50 kgw 以上必要なら、

(load (GREATER-THEN-OR-EQUAL (50 kgw)))

と記述する。又、slot-filler 間の条件、例えば、

縦弾性係数×断面 2 次モーメント
 $\geq 400 \text{ N} \cdot \text{m}^2$,

などは、下記の COND slot を frame-macro に挿入する。

(COND (縦弾性係数 (COND-1)
 断面 2 次モーメント (COND-1)))

COND-1 は図 4 c に示す frame である。COND slot はどこに書き込んでも差つかえない。

更に、FORTRAN, LISP 関数 f の利用も、SUBSTITUTE を利用し、図 4 d に示す形式に対する置換

(** (#X) SUBSTITUTE (f) WITH
 (PAR X1 X2...))

により、#X に結果を統一化することで可能になっている。LISP で計算する時は FORTRAN の位置に LISP が入る。

3. 設計式の推論

設計式の推論には、slot-filler に演算結果、数値を代入して値を求める演算操作に関する推論と、演算操作に加え、知識、すなわち frame の構造を slot の置換、削除、挿入で思考実験する組換操作に関する推論が考えられるが、本章は演算操作のみについて考察する。

3.1 数式の推論原理

formula 型知識は一連の推論ルールに従って操作を

```
[ ProblemSolver (#in #state #out)           /* Declare a rule frame. */
  RULE1 /* Define the 1st rule. This rule always fails. */
  ((Assert (ENVIRONMENT #in))                /* Set ENVIRONMENT. */
   (Question #frame)                          /* Find a question. */
   (ENVIRONMENT #e)                          /* Get ENVIRONMENT. */
   (Refute                                     /* call another world of DECIDE.*/
    ((FormulaSolver #e (#frame #v) #x)) ) /* Solve a question. */
   (OR ((Equal #state S)                      /* Solve step by step ? */
        (Get #out DataMemory #d)             /* Get data_frames. */
        (Refute                               /* Call ECIDE system. */
         ((AnswerPrint (#frame #v) #d)        /* Print answers. */
          (Delete (ENVIRONMENT #any)          /* Delete ENVIRONMENT. */
           (Assert (ENVIRONMENT #in) )        /* Set ENVIRONMENT. */
          ((Equal #state C)                   /* Solve sequentially ? */
           (Refute                             /* call another world of DECIDE.*/
            ((Delete (ENVIRONMENT #any)        /* Delete ENVIRONMENT. */
             (Assert (ENVIRONMENT #x) ) )      /* Set ENVIRONMENT. */
            (false) ) ) /* Fail for backtrack. */
        ) ) ) ) ) /* Define the 2nd rule. This rule is always refuted.*/
  ((ENVIRONMENT #out)                        /* Get ENVIRONMENT. */
   (Get #out DataMemory #d)                  /* Get data_frames. */
   (Get #out Frame #frame)                   /* Get the last question.*/
   (Get #out LocalValue #v)                  /* Get the value of question.*/
   (Refute                                   /* call another world of DECIDE.*/
    ((AnswerPrint (#frame #v) #d) )          /* Print answer. */
    #macro_311 ) ) ) )
C.F. (Refute (...)) refutes (...) in another forked DECIDE world.
```

Fig. 5 Example of rule frame definition

```
(ASSERT (ProblemSolver #in #state #out)
  .((Assert (ENVIRONMENT #in))
    ....
    (false) )
)
(ASSERT (ProblemSolver #in #state #out)
  .((ENVIRONMENT #out)
    ....
    (Refute ((AnswerPrint (#frame #v) #d) )
  )
)
} same as rule_macro of RULE1 in Fig. 5
} same as rule_macro of RULE2 in Fig. 5
```

Fig. 6 Directly refutable definition of rule

施され、問題解決に利用される。ルールの大部分は frame 表現に従う rule 型知識による方法と、Prolog の述語定義による方法で、知識として定義される。

rule 型知識とは、Prolog の述語定義の体部を frame の macro (macro 名は RULE *, * は 76 字以内の任意の文字列) に持つ frame であり、一つの frame に複数の rule-macro が定義できる。その frame-name は述語名そのもので、述語定義の頭部の変項はパラメータとして frame-macro に保持される。

変項は推論前の環境、データ、推論後の環境が 3 項でリストになっている。環境とは推論実行状況を示すリストで、それは name と parameter の対が複数個で構成されている。推論前と後の環境では、通常、parameter の一部が推論によって書き換えられている。図 5 に rule 型知識の構造を掲げる。

rule 型知識自体は frame で定義されるが、計算機的主記憶にロードされる時には、frame で登録されると共に、DECIDE により frame-name と frame-macro の点対リスト、(frame-name, frame-macro) を頭部、

rule-macroを各々体部とする Prolog 述語定義にも変換され、実行形式で登録される。図5の frame 表現を実行形式に変換した結果を図6に示す。

そのため、知識を frame そのものとして引用する場合は、frame-name を先頭にした4個以上の変項を持つリストで表すが、rule 型知識を実行形式で引用する（これは推論プロセスの起動を意味する）場合には、変項は3個以下となる。一方、推論ルールが Prolog の述語定義そのものである場合も同様に変項は3個以下である。

推論ルールは、slot-name, advice の pattern で駆動される frame 構造依存の demon 型の interpreter-rule と、slot-filler で駆動される構造独立の expert-rule を区別している。interpreter-rule は demon 型であり、slot の解釈、実行が目的である。本報は interpreter-rule に限定して考察する。

formula 型知識は、推論ルールから見れば一つのデータベースである。もちろん、推論ルールもシステムから見るとデータベースであることに変わりはない。このデータベースは推論実行途中では一切変更しない、いわゆる長期記憶¹¹⁾である。その代わり、推論中に前述の推論環境（いわゆる短期記憶）を構成して情報の受渡しを確保する。この環境は Prolog の back-track 制御により consistency を保証されている。

3.2 推論の実行

議論を簡単にするため、図2の片持ち梁のたわみで考える。値を求めたい変数名（同時に frame-name でもある）をシステムに Prolog 述語で、

(ASSERT (Question deflexion))

と定義しておく。そして、たわみを求めるのに必要な知識、初期データを frame 表現し、長期記憶に登録する。

その後、ProblemSolver という述語、

(ProblemSolver (# in # state # out))

をキー入力して、推論を起動する（端末から起動するには上の表現を更に括弧でくくる）。# in は推論前環境、# out は推論後環境、# state が C なら、(Question ***) の主張が尽きるまで短期記憶を置換しながら実行する。S なら、(Question ***) が1個終わるご

Numbers in HIERARCHY_macro mean 198ymdd such as 19840319. Initial data frames are defined on 19840319, and inferred on 19840528.	:	[MULTI_1 < (DATA ((2.499997E-12))) > HIERARCHY < (DATA(40528)) >]
[smoa < (DATA ((720000 mm^4))) > HIERARCHY < (DATA(40319)) >]	:	[BRACKET_1 < (DATA ((1.000000E+00))) > HIERARCHY < (DATA(40528)) >]
[mole < (DATA ((21000 kgf/mm^2))) > HIERARCHY < (DATA(40319)) >]	:	[length < (DATA ((60 cm))) > HIERARCHY < (DATA(40319)) >]
[POWER_2 < (DATA ((1.000027E-36 (m ^ 3)))) > HIERARCHY < (DATA(40523)) >]	:	[POWER_1 < (DATA ((2.160003E-01 (m ^ 3)))) > HIERARCHY < (DATA(40528)) >]
[MULTI_2 < (DATA ((2.314875E-36))) > HIERARCHY < (DATA(40523)) >]	:	[load < (DATA ((500 kgf))) > HIERARCHY < (DATA(40319)) >]
[position < (DATA ((9.999999E-11 cm))) > HIERARCHY < (DATA(40319)) >]	:	[deflexion < (DATA ((2.380928E-03 m))) > HIERARCHY < (DATA(40528)) >]

Fig. 7 Result of inference caused by "deflexion" frame

とに結果を出力し、推論環境をクリアすることを意味している。推論ルールは図5である。FormulaSolver において frame の各 slot を順序に従って評価するわけであるが、これは前にも述べたように slot-name を述語名とする演算用 demon により実行される。

演算用 demon は Prolog の述語定義形式で、第一変項が推論前の環境、第二変項が推論後の環境である。環境は属性子と属性値のリストであり、属性子と属性値は各々、Frame と推論中の frame 名、Name と推論中の demon 名、Filler と slot-filler 値、LocalValue と被演算値、Date と日付、Hierarchy と hierarchy-macro, DataMemory と data frame 群、Scratch-Memory と substitute で生成された frame 群で、属性値はすべて list notation である。推論後環境は次の demon の推論前環境となる。

図7に、解決すべき frame 名 deflexion を与えた時に推論された data frame を掲げる。現在、数値計算は主に Lisp 関数を利用し、一部特殊関数、COS, SIN 等は fortran library によっている。

4. 図形への応用

図形は表現と共に、移動という数値的操作も考える必要がある。図形の幾何学的知識表現は完全に階層的記述が可能であるが、図形の移動には matrix 演算が必要とするので、formula 型知識を拡張しなければな

らない。

4.1 図形の幾何学的知識表現

図形の幾何学的知識表現 (space 型知識の一部を構成する) を議論する時, SOLID MODELER とのリンケージを考慮しなければならないが, SOLID MODELER と知識ベースを融合した考察を行うには荷が重すぎるので, SOLID MODELER と知識ベースには適当なインタフェースが構成できるという仮定を基に考察を進めていく。図形の階層構造は B-reps 表現がそのまま利用できる。その階層は立体 (SOLID) ~面 (FACE) ~辺 (EDGE) ~点 (POINT) である。今のところ, 立体, 面は多面形のみを考えている。

frame の構造は formula frame とほぼ同じであるが, 図 4 b の FORMULA が SPACE に, CLASS macro の内容が SOLID, FACE などに, 更に frame-macro の slot-name が SOLID, FACE, EDGE, POINT になり, 各々立体, 面, 辺, 点であることを示している。その際, frame-macro のすべての slot-name は同じ必要がある。frame-macro slot は図形の構成要素を示し, slot 順序は SOLID frame では何の意味もないが, FACE frame では上から順に面を表側から見た時の右回り順を示し, EDGE frame では上 slot が始点, 下 slot が終点である。

POINT slot の slot-filler は, frame-macro に VECTOR slot を持つ frame-name か, ベクトル並び (X Y Z 1) 自体をその値に持つ。X, Y, Z は各々 X, Y, Z 座標値を表し, 1 は matrix 演算用タグであるが, 必要でない時は Z, 1 の省略ができる。

記述の簡便さを考え, プロトタイプの frame を用意している。そのいくつかを図 8 に掲げる。SUBSTITUTE は公式表現の場合と全く同じである。

次に, frame を使った matrix 演算を導入する。点を

```

      — a. Point definition —
[POINT_DEF
  ((VECTOR ((#x #y #z 1)))
   (COND (DNUMBER (#x #y #z)))
   .#macro_411 ]

      — b. Edge definition —
[EDGE_DEF
  ((POINT (START_POINT))
   (POINT (END_POINT))
   .#macro_412 ]

[START_POINT
  ((SUBSTITUTE (POINT_DEF)
   WITH (VECTOR ((#x #y #z 1)
   .#macro_413 ]

[END_POINT
  ((SUBSTITUTE (POINT_DEF)
   WITH (VECTOR ((#x #y #z 1)
   .#macro_414 ]

      — c. Square definition —
[LSQUARE_DEF
  ((EDGE (TO_SQUARE))
   ..
   (EDGE (LE_SQUARE))
   (COND
    (LENGTH (EQ #x TO .. LE)
     ANGLE (EQ (90 deg) TO .. LE)
     BIAS (EQ (0 0 0) TO .. LE))
   .#macro_415 ]

[TO_SQUARE
  ((SUBSTITUTE (EDGE_DEF)
   WITH (POINT TO_LE_SQUARE
   TO_RI_SQUARE))
   .#macro_416 ]

[LE_SQUARE
  ((SUBSTITUTE (EDGE_DEF)
   WITH (POINT BO_LE_SQUARE
   TO_LE_SQUARE))
   .#macro_419 ]

      — d. Cube definition —
[CUBE_DEF
  ((FACE (TO_CUBE))
   ..
   (FACE (BO_CUBE))
   (COND
    (ANGLE (EQ (90 deg)
    TO .. BO)
     BIAS (EQ (0 0 0) TO .. BO)
     .#macro_4110 ]

[TO_CUBE
  ((SUBSTITUTE (SQUARE_DEF)
   WITH (EDGE BA_TO_CUBE
   ..
   LE_TO_CUBE))
   .#macro_4111 ]

[BO_CUBE
  ((SUBSTITUTE (SQUARE_DEF)
   WITH (EDGE BA_BO_CUBE
   ..
   BO_LE_CUBE))
   .#macro_4116 ]

[BA_TO_CUBE
  ((SUBSTITUTE (EDGE_DEF)
   WITH (POINT BA_LE_TO_CUBE
   BA_RI_TO_CUBE))
   .#macro_4117 ]

[LE_TO_CUBE
  ((SUBSTITUTE (EDGE_DEF)
   WITH (POINT FR_LE_TO_CUBE
   BA_LE_TO_CUBE))
   .#macro_4128 ]

[BA_BO_LE_CUBE
  ((VECTOR ((0 0 0 1))
   .#macro_4129 ]

[FR_RI_TO_CUBE
  ((VECTOR ((1 1 1 1))
   .#macro_4136 ]

      BA::=back,BO::=bottom,FR::=front,LE::=left,RI::=right,TO::=top.

```

Fig. 8 Some standards of space frames

表す frame EX-1 を回転, 平行移動して得られる frame EX-2 を記述してみる。まず, frame EX-1 は図 9 a のごとく記述できる。これを原点に関し T° 回転後, (0, 0, 0) 平行移動させる space frame (見方を変えれば, 推論ルールを demon を利用して記述している) は, 図 9 a に掲げる frame EX-2 となる。APPLY demon は APPLY 属性値 EX-1 から advice 部の TO 属性値 MATRIX-1 への, 演算 MATRIX (これも demon である) の実行を意味する。

一方, matrix を意味する frame は, VECTOR slot が 4 個連続した右から作用する matrix として表現する。この matrix は推論時に, 要素を row ごとに「:」で区切った list となっている。演算結果は data frame として表現される。APPLY は SUBSTITUTE と同様, 推論時にしか起動されないことに注意する必要がある。

```

----- a. Given frames -----
[EX_1
  (( VECTOR ( (1 1 1 1))
    HIERARCHY (SPACE(40317))
    CLASS (POINT)
  )
  :
  [MATRIX_1
    (( VECTOR ( (COS T) (SIN T) 0 0)
      ( VECTOR ((- SIN T) (COS T) 0 0)
        ( VECTOR ( 0 0 1 0)
          ( VECTOR ( 0 0 0 1)
            ( END_OF_SLOT (MATRIX_1) )
          )
        )
      )
    HIERARCHY (DATA(40317 0 3))
    CLASS (MATRIX)
  )
  :
  [T ((DATA (45 deg) UNIT_RULE (rad))
    HIERARCHY (DATA(40317 0 4))
    CLASS (DATA)
  )
]

----- b. After EX_2 is inferred,
       the next frame exists in ScratchMemory. -----
(EX_2 ((VECTOR (((1.859664E-05 1.414213E+00 1 0 :))))
  HIERARCHY (SPACE (40323 0 1)) INHER (EX_1) CLASS (POINT))
C.F. ':' shows a internal discriminator of rows of matrix.

```

Fig. 9 Matrix operation to space frame

```

----- a. face definition -----
(ASSERT
  ( RECTANGLE
    ( (EDGE ((TO_RECTANGLE)
      (EDGE ((RI_RECTANGLE)
        (EDGE ((BO_RECTANGLE)
          (EDGE ((LE_RECTANGLE)
            HIERARCHY ( SPACE(31120 1))
            CLASS (FACE)
          )
        )
      )
    )
  )

----- b. Edge definition -----
(ASSERT
  ( TO_RECTANGLE
    ( (POINT (LE_TO_RECTANGLE))
      (POINT (RI_TO_RECTANGLE)) )
    HIERARCHY ( SPACE(31120 2))
    CLASS (EDGE)
  )
  :
  ( RI_RECTANGLE
    ( (POINT (LE_TO_RECTANGLE))
      (POINT (RI_TO_RECTANGLE)) )
    HIERARCHY ( SPACE(31120 3))
    CLASS (EDGE)
  )
  :
  ( BO_RECTANGLE
    ( (POINT (RI_TO_RECTANGLE))
      (POINT (BO_RI_RECTANGLE)) )
    HIERARCHY ( SPACE(31120 4))
    CLASS (EDGE)
  )
  :
  ( LE_RECTANGLE
    ( (POINT (BO_RI_RECTANGLE))
      (POINT (BO_LE_RECTANGLE)) )
    HIERARCHY ( SPACE(31120 5))
    CLASS (EDGE)
  )

----- c. Point definition -----
(ASSERT
  ( LE_TO_RECTANGLE
    ( (DATA ((400 700)) )
      HIERARCHY ( SPACE(31120 6))
      CLASS (POINT)
    )
  )
  :
  ( RI_RECTANGLE
    ( (DATA ((600 700)) )
      HIERARCHY ( SPACE(31120 7))
      CLASS (POINT)
    )
  )
  :
  ( BO_RECTANGLE
    ( (DATA ((600 300)) )
      HIERARCHY ( SPACE(31120 8))
      CLASS (POINT)
    )
  )
  :
  ( LE_RECTANGLE
    ( (DATA ((400 300)) )
      HIERARCHY ( SPACE(31120 9))
      CLASS (POINT)
    )
  )

```

Fig. 10 Rectangle knowledges represented in frame

ある。結果を図9bに示す。

4.2 幾何学的知識の推論

理解の便のために、2次元図形の display の推論手続きを紹介したい。とりあげた図形は長方形で、長方形の持つ条件はすべて満足しており、図形の frame 表現は完全になされている。この知識を図10に掲げる。

display を起動する述語は Demonstrator で、公式推論の起動述語 ProblemSolver の logic と比較するならば、FormularSolver が VectorFinder に、AnswerPrinter が ScreenPlotter に、ProblemSolver が Demonstrator に変わっているだけで、つまり、知識型が space か formula かの使い分けにより、ProblemSolver と Demonstrator は同じ logic になることを意味している。

Vectorfinder は VECTOR, DATA slot を保持する frame にたどり着くまで探索を行う。これは、EDGE, POINT という slot-name に対応している demon が、下位レベル frame の VECTOR, DATA slot を見つける任務を持つからである。

VECTOR, DATA slot が存在すれば、それが値として呼び出した frame に返される。もし、存在しない場合には、最後の frame 名が値として返される。呼び出し側 frame は各 slot を返ってきた値が filler である DATA slot に変換後、自分自身を DataMemory に登録する。VectorFinder 実行結果のプリント出力を図11に掲げる。

ScreenPlotter は、VECTOR リストを平面を定義するベクトル列と見なし、Lisp で書かれた plot 10 emulator に順次リスト要素を引き渡し、Tektronix 4014 へ図形の入出力を行う。同様の操作は3次元図形に対しても拡張できる。

5. 展 望

frame 表現によって、数式と図形を知識ベースの中へ埋め込むことが

実証できたのであるが、本方式の特徴は、構造すべてにユーザが直接アクセスができる表現となっていることである。本稿では、その第一歩として、数値操作を主体にした従来の数値計算、図形表現を実現する推論を実験し、推論の有効性を確かめたにすぎない。

しかし、数式の場合には、式を構造的に変形すると

```

a. Question predicate
(ASSERT
  ( Question RECTANGLE )

b. Inferred frame
[ RECTANGLE
  ( (DATA (((400 700) (600 700))))
    (DATA (((600 700) (600 300))))
    : (DATA (((600 300) (400 300))))
      (DATA (((400 300) (400 700)))) )
    HIERARCHY (SPACE(31120))
    CLASS (FACE) ]

```

Fig. 11 Result of VectorFinder by Question

いう操作も、frameを直接書き換えるか、書換えルールを用意することで実現される。特に、設計条件の追加、変更がframe単位の操作で可能となるから、設計の最適化過程のインタラクションが効率化されると判断される。そのためには、式操作のルールを整理して、toolを作成していく必要があると考える。

一方、図形に関してはポインタだけでなく、Semantic Netにおけるような図形間の言語的関係の表現に道を開いたことである。形状モデラは数値的構造の表現には極めて強力であるが、人間の親しんでいる直観的な図形の配置関係をspace型知識で補完することで、柔軟なグラフィックスインタフェースの構成が可能と考える。今後、図形の数値表現と言語表現の対応を図ることが重要となろう。

6. 結 論

本報の結論を以下に要約する。

- (1) 設計公式の知識表現は、慣習的な計算の優先順序で階層化することにより実現できる。公式知識の階層表現は公式知識間の相互利用も可能にする。
- (2) 幾何学的な空間知識の表現は、多面体図形の場合、B-repsに基づく構造記述をframeに変換することで達成できる。更に、プロトタイプ図形の利用で、柔軟に知識表現をする方法を与えることが

できた。

- (3) 図形描画と数値計算の推論ルールはframeとdemonで知識表現できる。長方形表示と片持ちはりのたわみ計算の実行を推論例として検証した。表示、演算はLisp、試作plot 10 Lisp emulatorを利用している。

文 献

- 1) 渡辺 茂：設計論，基礎工学 10，岩波書店（1970）。
- 2) M. Minsky: A Framework for Representing Knowledge, in *The Psychology of Computer Vision*, McGraw-Hill, New York, N. Y., (1975) 211-277.
- 3) 岸 義樹：知的処理機構に基づいた概念解釈による設計の研究——知識表現と利用，昭和57年度精機学会春季大会学術講演会講演論文集（1982）No. 106.
- 4) 岸 義樹：人工知能とCAD，ロボットエンジニアリング，1, 3（1982）46.
- 5) 岸 義樹：概念設計における知識表現とその利用，グラフィックスとCADシンポジウム論文集，情報処理学会（1983, 12）231.
- 6) 岸 義樹：知識依存型概念設計システムの展望，昭和59年度製品科学研究所研究成果発表会資料，日本産業技術振興協会（1984, 12）81-100.
- 7) 岸 義樹：知識依存型概念設計システムの研究——機械構造の知識表現，精密機械，51, 11（1985）2077.
- 8) 岸 義樹：知的処理機構に基づいた概念解釈による設計の研究——自然語及び推論による概念形成の基礎，昭和58年度精機学会春季大会学術講演会講演論文集（1983）No. 302.
- 9) 岸 義樹，福井幸男：概念設計のためのProlog環境，昭和60年度精機学会春季大会学術講演会講演論文集（1985）No. 530.
- 10) 岸 義樹：知的処理機構に基づいた概念解釈による設計の研究——公式・図形の知識表現と利用，昭和59年度精機学会春季大会学術講演会講演論文集（1984）No. 818.
- 11) J. R. Anderson: *The Architecture of Cognition*, Harvard Univ. Press, MA, (1983).