Vol. 27 No. 1 情報処理学会論文誌 Jan. 1986

拘束条件リダクション法を用いた機械設計計算支援システム†

長澤 勲†† 古川由美子††

現在，機械設計の現場では，FORTRAN などでかかれた専用的な設計システムが実用されている．近年，多品種少量生産傾向が強まるとともに，保守性，拡張性，多目的性に優れた汎用設計システムの必要性が認識されてきた．このためには，設計の方法論に基づいた設計システムを開発する必要がある．筆者らは，先に，論理プログラミングを用いた拘束条件リダクション法とよぶ計算モデルを提案し，これが設計計算のモデルとして優れた表現力をもつことを示した．本研究では，この計算モデルを基礎に機械設計のうち基本設計とよばれている部分を対象とした汎用的機械設計計算支援システムを開発した．ここでは，まず設計システムを実用化するうえでの問題点と解決策を示し，次に歯車減速機を例にとり設計システムのプログラミング技術について述べている．

1. まえがき

現在，機械設計の現場では，FORTRAN などでかかれた専用的な設計システムが実用されている．近年，多品種少量生産傾向が強まるとともに，保守性，拡張性，多目的性に優れた汎用設計システムの必要性が認識されてきた．このためには，設計の方法論に基づいた設計システムを開発する必要がある．機械設計に適した設計の方法論の一つに拘束条件[1),2)]を基礎にした方法がある．すでにこのような試みは，沖野らの TIPS[3),4)] において行われている．しかしながら，計算モデルの立場からは必ずしも十分に研究されているとは言いにくい．筆者らは，先に，論理プログラミングを用いた拘束条件リダクション法とよぶ計算モデルを提案し，これが設計計算のモデルとして優れた表現力をもつことを示した[5)]．本研究では，この計算モデルを基礎に機械設計のうち基本設計とよばれている部分を対象とした汎用的機械設計計算支援システム（以下設計システム）を開発した．ここでは，まず設計システムを実用化するうえでの問題点と解決策を示し，次に歯車減速機を例にとり設計システムのプログラミング技術について述べている．

2. システムの目的

本論文では機械設計のうち基本設計とよばれている部分を対象としている．基本設計においては，機械の全体的構造はあらかじめ与えられていることが多く，設計作業とは所要性能，寿命，信頼性，経済性などの各項目のバランスを考えながらしだいに構造や，構造諸元を絞り込んでいく過程と見ることができる．一般に，設計解には，ある程度自由度があり，機械が使用される環境や生産設備なども考慮に入れるため，設計要求間のバランスの判断は当面，設計者が行うのが適当であると思われる．しかし設計公式や設計資料を多用し，試行錯誤に行われる設計計算そのものは自動化することができる．本システムは，この設計計算を設計の方法論を基礎に統一的に支援することを目的にしている．

3. 設計の方針

3.1 設計計算のモデルの使用

現在，現場で使用されている専用的設計システムは FORTRAN などの手続型言語でかかれている．この場合，問題となるのは，仮定生成・検証，データの流れなどを制御の流れとして表現するため，大変繁雑になることである．このことは設計システムの保守性や拡張性を悪くする原因となっている．この困難を解決するため，拘束条件リダクション法（設計問題を拘束条件の集合として与え，これをより簡単な拘束条件の集合に分解することによって解く方法）とよぶ設計計算の汎用モデルを使用する．この方法の利点は，歯車や軸など，個々の設計システムを構成するには設計資料や設計公式を拘束条件の形で与えるだけでよく，計算の制御を考える必要がないことである．

3.2 対象指向プログラミング手法[6),7)]の導入

機械設計では各種の設計公式が使用されており，各

† A Machine Design Calculation Support System, Using the Method of Constraints Reduction by ISAO NAGASAWA and YUMIKO FURUKAWA (Computation Center, Kyushu University).

†† 九州大学中央計数施設

現場によって少しずつ設計の方法が異なっている．このため単一の設計システムを用意するだけでは不十分であり，各現場において設計システムを自由に拡張・修正できることが望ましい．対象指向プログラミング* の方法は拡張性に富む方法として知られている．ここでは，知識表現の枠組としてこの方法を用いる．

3.3 デバッグ機能の整備

機械設計の計算は多くの場合，機械の構造や材料などを仮定し試行錯誤に進められるので，設計解の探索を行うシステムが必要になるが，このようなシステムのデバッグは一般にかなり面倒である．ここでは，設計計算に多用される生成・検証法のデバッグを容易にするため，データの流れ，仮定の流れを管理したデバッグシステムを備えている．

3.4 設計システムの多目的性への配慮

本システムで使用する計算モデルは設計解の探索，設計の検証，設計の部分変更など多目的に使用できる性質をもっている．これらの機能が簡単に使用できるように配慮している．

4. 設計システムの概要

4.1 システムの構成

設計システムは対象指向プログラミング機能をもった拡張 Prolog 言語 ADL/KERNEL を用いて記述されている**．図1に示すように，システムの主要部は二つのオブジェクト CRS-SYS (Constraints Reduction System) と CRS-OBJ (Object with Constraints) にまとめられている．図中⇒および→は

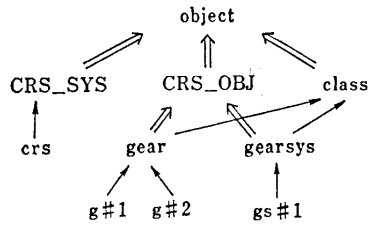

図 1 設計システムの構成
Fig. 1 An overview of the design system.

それぞれ subclass of および instance of 関係を示している．CRS-SYS には，拘束条件リダクション法が実現されており，設計解の探索，検証などに使用される．設計計算に用いられる設計公式や設計資料は，CRS-OBJ のサブクラスを定義することによって与えられる．また，個々の設計問題および設計解はこのサブクラスのインスタンスに与えられる．CRS-OBJ には，これらのインスタンスの編集や設計結果の表示に必要なメソッドが集められている．設計システムのメソッドおよび設計知識を記述するクラスとインスタンスの記述形式を**表1**，**図2** に示す．

5. 拘束条件リダクション・システムの機能拡張

拘束条件リダクション・システムの表現能力は設計計算の汎用モデルとして，十分であることは前論文[5]で示した．しかし，実際の設計システムに応用する場合問題となったのは，(1)解探索能力が十分でないこと，(2)設計システムのデバッグが困難であること，であった．本章では，data dependency[8),9)] を管理す

表 1 設計システムの主要なメソッド
Table 1 Principal methods of the design system.

リダクション・システムのメソッド	
(? crs trace)	リダクション過程をトレースモードにする．
(? crs untrace)	トレースモードを解除する．
設計対象のメソッド	
(? Ins show given)	設計の与件 (given ファセット値) を表示する．
(? Ins show result)	設計の結果を表示する．
(? Ins set Slot Val)	指定スロットの given ファセット値を Val に設定する．
(? Ins cancel Slot)	指定スロットの given ファセット値を取り消す．
(? Ins to-given)	設計結果を与件と見なし given ファセットに設定する．
(? Ins solve)	インスタンスの拘束条件集合を crs に依頼して解く．
(? Ins verify)	インスタンスの拘束条件集合を crs に依頼して検証する．

* この用語は通常，対象を用いた知識表現のモジュール化，メッセージによる計算モデルの二つの意味で使われている．本論文では，さらに，拘束条件リダクション法という計算モデルを追加した意味で用いる．

** ADL システムの初版では対象指向プログラミングの機能が実現されていなかった．したがって前論文[5] とはシステムの構成方法が少し変更されている．

```
(class 〈クラス名〉
       (supers 〈上位クラス名〉‥〈上位クラス名〉)
       (class 〈クラス名〉)
       (cvar 〈状態変数〉‥〈状態変数〉)
       (ivar 〈状態変数〉‥〈状態変数〉)
       (crs  (Self 〈スロット〉‥〈スロット〉)
             (proc 〈拘束条件〉‥〈拘束条件〉))
       (methods 〈メソッド名〉‥〈メソッド名〉)
       (rules   〈ルール名〉‥〈ルール名〉)
       (locals  〈ローカル手続名〉‥〈ローカル手続名〉))
(instance 〈インスタンス名〉
     (class 〈クラス名〉)
     (ivar (crs (〈スロット名〉〈ファセット〉)‥
                (〈スロット名〉〈ファセット〉))
           〈状態変数〉‥〈状態変数〉))
〈クラス名〉              ::=〈文字定数〉
〈インスタンス名〉        ::=〈文字定数〉
〈スロット〉              ::=(〈スロット名〉〈変数〉)
〈スロット名〉            ::=〈文字定数〉
〈拘束条件〉              ::=〈素論理式〉|〈インスタンス参照〉
                            |〈ルール呼び出し〉|〈wait 記述〉
〈インスタンス参照〉      ::=(fget (〈識別子〉 (class 〈クラス名〉)
                              〈スロット参照〉‥〈スロット参照〉))
〈スロット参照〉          ::=(〈スロット名〉〈prolog の項〉)
〈識別子〉                ::=〈文字定数〉|〈変数〉
〈ルール呼び出し〉        ::=(:r Self 〈素論理式〉)
〈wait 記述〉             ::=(wait (〈変数〉‥〈変数〉)
                                  〈拘束条件〉‥〈拘束条件〉)
〈状態変数〉              ::=〈変数を含まない項〉
〈ファセット〉            ::=(given 〈値〉)|(derived 〈値〉)
〈値〉                    ::=〈変数を含まない項〉
〈ローカル手続〉          ::=〈Prolog 手続〉
〈メソッド〉              ::=〈Prolog 手続〉
〈リダクション・ルール〉  ::=〈Prolog 手続〉
〈メソッド呼び出し〉      ::=(? 〈インタンス名〉〈セレクタ〉
                              〈引数〉‥〈引数〉)
```

図 2 オブジェクトの形式
Fig. 2 Syntax of object.

ることによってこの問題を解決する一方法を示す.

5.1 リダクション手続の拡張

解探索効率を改善する試みは prolog においてもすでにいくつか試みられており，知的バックトラック技術[10),11)]として知られている．しかしこの技術は，記憶スペース，計算時間のオーバヘッドが大きいという問題点がある．ここでは，拘束条件リダクション法というマクロな計算過程に限定してこの方法を適用する．これは次の理由から効果的であると考えられる．

(1) 設計問題や設計対象のクラスに与えられる拘束条件集合は個別に記述されるため，これらの相互作用をあらかじめ知ることは困難である．このため解探索効率改善およびデバッグの支援のため必要性が高い．

(2) 個々の拘束条件を解くリダクション・ルール(以下ルール) は Prolog 手続としてかかれるが，小規模でありプログラミング上の困難は少ない．

リダクション過程

data dependency を管理するためリダクション手続を拡張する．リダクションの各段階の拘束条件集合を S_i で表すと，リダクション過程における拘束条件集合の系列は $S_0, S_1, \cdots, S_i, S_{i+1}, \cdots, S_n$ とかける．S_0 は初期拘束条件集合，各 S_{i+1} は S_i からリダクションによって得られた拘束条件集合である．添字 i をステップ番号とよぶ．以下，リダクション・ステップを単にステップとよび，ステップ番号で代表する．

ステップ i は次のように行われる．S_i からある拘束条件 $C \in S_i$ を選び，リダクションを試みる．C の評価結果は次の三つに分類される．(1) fail 積極的な否定とみなされ，後戻りを引き起こす．(2) suspended 何ら積極的情報がなく，別の拘束条件を試みる．(3) success C は別の拘束条件集合 D_s に置き換えられ，リダクションは成功する．

非決定的に success したステップを仮定ステップとよぶ．仮定ステップは後続するステップにおいて fail が生じたときの後戻り点となっている．拘束条件 C に出現する変数は先行するいくつかのステップにおいて束縛されている．このステップの集合をステップ i の入力集合 I_s とよぶ．ステップ i の仮定集合 A_s を次のように定義する．入力集合 I_s の各要素 $j \in I_s$ の仮定集合を A_j とするとき，

(1) ステップ i が仮定ステップならば

$$A_s = \{i\} \cup \left(\bigcup_{j \in I_s} A_j \right)$$

(2) 仮定ステップでなければ

$$A_s = \bigcup_{j \in I_s} A_j$$

また，$A_{in} = \bigcup_{j \in I_s} A_j$ をステップ i の入力仮定集合とよぶ．入力仮定集合はリダクションの結果に関わりなく定義される．リダクションの fail の原因は入力仮定集合にある．そこでこのうち最もステップ番号の大きい仮定ステップに後戻りできれば無駄な探索を避けることができる．ここでは，少数のプリミティブを用いてこの機能を実現する．data dependency の表現には変数に補助情報を格納する．また，仮定ステップへの後戻りには remote cut を用いる．リダクション手続および使用するプリミティブを**図 3**, **表 2** に示す．図中基本手続[5)] に追加された部分には | が付してある．また reduce 述語の第一引数はステップ番号，… → … ; … は if…then…else… を表している．

表 2 リダクション手続のプリミティブ
Table 2 Primitives of the reduction procedure.

(step S)	prolog 処理系の推論ステップ番号（リダクションステップではない）を取り出す.
(cut S)	推論ステップ番号 S からこの述語が評価されるまでに記録されているすべての後戻り点を除去する. remote cut とよぶ.
(detp S YesNo)	推論ステップ S からこの述語が評価されるまでに後戻り点が設定されたか否かをテストし yes/no を返却する.
(exec C D_s)	拘束条件 C を拘束条件集合 D_s にリダクションする. このとき値が束縛されたすべての変数には補助情報を格納する記憶セルが用意される.
(setB S I A_s)	上記記憶セルに prolog 処理系のステップ番号 S, ステップ番号 I, 仮定集合 A_s を格納する.
(input C S_m I_s A_{in})	拘束条件 C の変数の補助情報を取り出す. S_m, I_s, A_{in} はそれぞれ prolog 処理系の推論ステップ番号, 入力集合, 入力仮定集合である. S_m は入力仮定集合のうち最も最近の後戻り点の直後を示している.
(failp C)	拘束条件 C が fail か否かをテストする.
(nondetp C)	拘束条件 C が非決定的か否かをテストする.

```
(solve | Cs) :- (reduce 1 Cs nil).
(reduce _ nil Ws) :- (monitor Ws).
(reduce I (C|Cs) Ws) :-
  (suspendp C), !,
  (reduce I Cs (C|Ws)).
(reduce _ (C|_) _) :-
  (failp C),
  (input C S Is Ain),
  (cut S), fail.
(reduce I (C|Cs) Ws) :-
  (input C S1 Is Ain),
  ((nondetp C) → As=(I|Ain); As=Ain),
  (add 1 I I1),
  (step S2),
  (exec C Ds),
  (detp S2 YesNo),
  (YesNo=yes → S3=S1; (step S3)),
  (setB S3 I As),
  (append Ws Cs Cs1),
  (append Ds Cs1 Cs2),
  (reduce I1 Cs2 nil).
```

図 3 リダクション手続の拡張
Fig. 3 The extension of reduction procedure.

5.2 リダクションの優先順位*

拘束条件集合の評価はデータフローの制限を満足すれば任意の順序に行うことができる. しかし拘束条件評価の順序を考慮すれば解探索効率および計算手順の理解しやすさを改善することができる. ここでは次のような優先順位を用いる. (1)インスタンス参照 fget**, (2)決定性の拘束条件, (3)非決定性の拘束条件, (4)wait 記述

(1)はインスタンスの拘束条件を優先的に得るためである. (3)は仮定生成をできるだけ遅らせるためである. このことによって決定性のリダクション・ステップの入力仮定集合は必ず最も最近の仮定ステップを

```
(for M ?M ?M): -!.
(for M ?M ?K).
(for M ?I ?K): -
      (I<K), (add 1 I J), (for M J K).
(teeth_no_gen Za Zb): -
      (for Za 14 20),
      (deval (integer Zb)).
```

(a) リダクション・ルール

```
(1) | (for M 3 5)
      (teeth_no_gen Z1 Z2)
      (for Bm 11 12)
      (Z1==Z2*1/2)
      (100≦(Z1+Z2)*M≦200)
      (40≦Bm*M≦50)
            Is=Ain={ }
            σ={M=3}
            As={(1)}
(2) | (teeth_no_gen Z1 Z2)
      (for Bm 11 12)
      (Z1==Z2*1/2)
      (100≦(Z1+Z2) *M≦200)
      (40≦Bm*M≦50)
            Is=Ain={ }
            σ={Z1=14, Zb=Z2}
            As={(2)}
(3)   (for Bm 11 12)
    | (Z1==Z2*1/2)
      (100≦(Z1+Z2)*M≦200)
      (40≦Bm*M≦50)
      (integer Zb)
            Is=As=Ain={(2)}
            σ={Z2=28}
(4)   (for Bm 11 12)
    | (100≦(Z1+Z2)*M≦200)
      (40≦Bm*M≦50)
      (integer Zb)
            Is={(1) (2) (3)}
            As=Ain={(1) (2)}
            σ={ }
(5)   (for Bm 11 12)
      (40≦Bm*M≦50)
    | (integer Zb)
            Is={(2) (3)}
```

* 前論文[5]から変更されている.
** 前論文のフレーム参照に同じ.

```
                    As=Ain={(2)}
                    σ={ }
(6)a | (for Bm 11 12)
       (40≦Bm*M≦50)
                    Is=Ain={ }
                    σ={Bm=11}
                    As={(6)}
(7)a | (40≦Bm*M≦50)
            fail Is=Ain={(1) (6)}
(6)b | (for Bm 11 12)
       (40≦Bm*M≦50)
                    Is=Ain={ }
                    σ={Bm=12}
                    As={[6]}
(7)b | (40≦Bm*M≦50)
            fail Is=Ain={(1) [6]}
```

(b) リダクション過程

図4 リダクション過程の例
Fig. 4 An example of reduction process.

含むことになる．(4)については後述する．

5.3 リダクション過程の例

ここでは簡単な例を用いてリダクション過程を説明する．**図4**は歯車システムの形状パラメータを決定する問題である．歯車の歯数 Z_1, モジュール M, 歯幅モジュール比 B_m がそれぞれ，14～20枚，3～5 mm，10～12の範囲で独立に変化できるとき，軸間距離 $D=(Z_1+Z_2)*M$ と歯幅 $B=B_m*M$ がそれぞれ $100 \leqq D \leqq 200$ および $40 \leqq B \leqq 50$ の条件を満足する組合せを求める．仮定生成はルール for および，teeth-no-gen によって行う．ステップ(1)においてまず (for M 3 5) を評価するものとする．変数 M には何も束縛されていないので入力集合 I_s，および入力仮定集合 A_{in} はともに空集合である．また図中 σ は変数の束縛を示している．このステップは仮定ステップであるので仮定集合は $A_s=\{(1)\}$ である．以下同様に(2)(3)(4)(5)が実行される．ステップ(6)a は仮定ステップである．まず，$B_m=11$ となるが別解 $B_m=12$ が残されているので後戻り点でもある．ステップ(7)a は fail である．このステップの入力仮定集合は {(1)(6)}であるからまずステップ(6)に後戻りする．図中後戻りの実行を(6)b で表している．このステップは仮定ステップではあるが，ほかに別解が残されていないので後戻り点とはならない．図中後戻り点と区別するため[6]のように表す．(7)b の fail はステップ(1)に後戻りすることになる．

6. 設計システムのプログラミング

6.1 設計プログラムの例

図5に歯車システムの設計プログラムの一例を示す．歯車のクラス gear には7個の拘束条件がある．歯車の歯数，モジュール，材料はそれぞれ，リダクション・ルール teeth-no-gen, module-gen, material-gen によって仮定される．また，歯の曲げ強さをルイスの公式 lewis，歯車直径と軸直径の関係を不等式>=によって検証している．:=は通常の代入文である．

```
(class gear
 (supers crs_obj)
 (class class)
 (crs
  (Self
   (power_kw Kw)
   (rpm N)
   (no_of_teeth Z)
   (module M)                          } 属性諸元
   (tooth_width B)
   (pitch_circle_diameter D)
   (shaft_diameter Sd)
   (material Mat))
  (proc
   (:g Self (teeth_no_gen (Kw? N?) Z))
   (:g Self (module_gen   (Z?)    M))
   (:g Self (material_gen (Z? M?) Mat))
   (:p Self (B := 10 * M))             } 拘束条件集合
   (:p Self (D := M * Z))
   (:t Self (D? >= 2 * Sd?))
   (:t Self (lewis M? Z? N? B? D? Kw?
                              Mat?)))))
 (locals material ... )
 (rules teeth_no_gen ... ))
```

(a) 歯車のクラス

```
(lewis ?M ?Z ?N ?B ?D ?Kw ?Mat):-
    (toothshape_coef Z Kz),
    (V := 3.141593 * D * N / 60000),   } 曲げ強さ
    (velocity_coef V Kv),              } の検証
    (T := 97400 * Kw / N),
    (Fs := 2 * T / D),
    (material Mat _ _ Sigwb _ _),
    (Sig := 2.4 * Fs / (Kv * B * M * Kz)),
    (Sig <= Sigwb).

(teeth_no_gen ?Wait Z):-
    (freeofvar Wait),
    (for Z 14 100).

(module_gen ?Wait M):-                 } 歯数，モジュール
    (freeofvar Wait),                  } 材料の仮定
    (select M (2.5 3 ... 25)).

(material_gen ?Wait Mat):-
    (freeofvar Wait),
    (select Mat (fc20 fc30 ... sncm26)).
```

(b) 歯車のリダクション・ルール

```
(class gearsys
 (supers crs_obj)
 (class class)
 (crs
  (Self
   (reduction_ratio U)
   (in_power_kw Kw1)
   (in_rpm N1)
   (out_power_kw Kw2)
   (out_rpm N2)
   (driving_gear G1)
   (drived_gear G2))
  (proc
   (:p Self (Z1 == Z2 * U))
   (:p Self (Kw2 == Kw1 * 0.98))
   (:p Self (N2 == N1 * U))
   (:g Self (for Z1 14 20))
   (fget (G1 (class gear)
             (power_kw Kw1)
             (module M)
             (no_of_teeth Z1?)
             (rpm N1)))
   (fget (G2 (class gear)
             (power_kw Kw2)
             (module M)
             (no_of_teeth Z2?)
             (rpm N2))))))
```

部分(歯車)の参照

(c) 歯車システムのクラス

図 5 設計プログラムの例

Fig. 5 An example of design program.

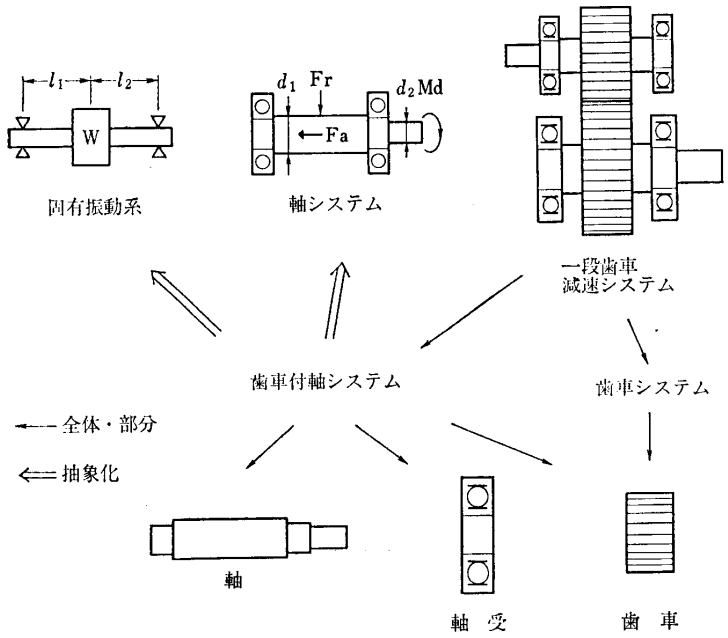

図 6 歯車システムの階層

Fig. 6 Hierarchy of reduction gear system.

歯車システムのクラス gearsys には6個の拘束条件がある．歯車システムは2個の歯車から構成されているのでこれらをインスタンス参照 fget によって参照している．fget 中，2個の歯車のモジュール M は同一である．また歯数 Z_1, Z_2 回転数 N_1, N_2 減速比 U は，二つの方程式==によって拘束されている．

6.2 階層構造の利用

モジュール性，可用性に優れた設計システムを作成するため二つの階層構造が利用できる．

部分・全体階層

設計対象はいくつかのサブシステムや要素に分割して取り扱える．たとえば一段歯車減速システムは**図6**に示すように歯車システムと二つの歯車付軸システムに，さらに歯車システムは二つの歯車に，歯車付軸システムは歯車，軸，二つの軸受に分割できる．一般にこの分割は必ずしも排他的ではない．設計プログラムにおいてこの分割を表現するには，図5に示すようにクラスの拘束条件部に部分を参照するインスタンス参照 fget を記述すればよい．分割された各部分は fget に含まれる変数を共有することによって結合されている．

抽象化階層

設計公式や設計資料は機械と機械が動作する環境を抽象化し，モデル化することによって得られている．このため一つの設計対象は，抽象化の各レベルにおいて表現することができる．たとえば図6に示す歯車付軸システムは軸と軸受設計のためには，歯車は重要ではなく，曲げおよび捩り応力を考慮した軸システムに抽象化すればよい．また共振に対する安全性を検証するには固有振動系に抽象化できる．軸システム，固有振動系はポンプ軸などの設計にも使用することができる．この抽象化階層はクラス階層として表現すればよい．インスタンス参照 fget によって下位クラスのインスタンスが参照された場合，このクラス階層によって上位クラスの拘束条件集合が集められる．

6.3 生成検証法

設計問題を表す拘束条件集合を解く場合，直接的解法が存在することが望ましい．しかし機械設計においては，非数値的拘束関係の存在，非線形な設計公式の使用，寸法・材料の標準化などの理由から生成・検証法が最もよく使用されている．

生成・伝播・検証の分離

仮定の生成・伝播・検証を別々のルールによって行う．これは仮定の生成，検証がどのルールによって行われたか明示的に知ることができ，設計システムのデ

```
(:g Self C) :- (freeofvar C), !.
(:g Self C :-
     (setof Set C (:r Self C)),
     (select C Set).
(:p Self C) :- (:r Self C), !.
(:t Self C):-
     (freeofvar C), !!,
     (:r Self C).
(select C (C)):- !.
(select C (C|_)).
(select C (_|Rs)):- (select C Rs).
```

図 7 リダクション・ルール呼び出しの拡張例
Fig. 7 An extension of reduction rule call.

バッグに効果的である．またこのようにルールの機能を分離しても表現能力は同じである．仮定の生成，伝播，検証を行うルールをそれぞれ，generator, propagator, tester とよぶ．図 7 にルール呼出し手続の拡張例を示す．図中 :g, :p, :t はそれぞれ，generator, propagator, tester を示している．

仮定生成の順序制御

拘束条件集合は一般に複数個の解をもっている．設計システムの通常の使用においては，最初に得られる解が重要であるため，仮定生成の順序についてよく考慮しておくことが必要である．これは，generator 間に直接，あるいは間接に data dependency を導入することによって行う．前出の図 5 gear クラスは歯車の歯数，モジュール，材料の順に仮定生成を行った例である．

仮定生成の衝突

二つ以上の generator が共有変数に異なる値を束縛する可能性があるとき仮定生成の衝突が生じたという．この場合，generator 間に data dependency が定義されていなければ，システムによって偶然先に評価された generator が仮定を生成し，他方は suspend される．このようなことを防ぐには，いずれか一方の generator の実行を優先し，他方を無視すればよい．たとえば図 5 では，gearsys クラスと gear クラスの歯数を生成する generator, for と teeth-no-gen が衝突している．インスタンス参照 fget によって gear を参照するとき，修飾付変数 $Z_1?$, $Z_2?$ を用いれば teeth-no-gen の仮定生成を禁止し for を優先することができる*.

6.4 段階的詳細化法

少し大きな設計問題では設計計算の大略の流れは定まっていることが多い．この情報は，解探索量の減少，デバッグや計算手順の理解しやすさのために役立つ．これは与えられた拘束条件集合を一度に解くのではなく，いくつかの部分集合に分けて解くことによって表現できる．ここでは wait 述語を用いる．wait 述語は第一引数のすべての変数に値が束縛されたとき動作し，第二引数以下に与えられた拘束条件集合をシステムの拘束条件集合に併合する．wait 述語はリダクションの優先順位が最も低くとられている．したがってほかにリダクション可能な拘束条件がなくなったときに評価される．図 8 は一段歯車減速システムの拘束条件部である．まず歯車システムの拘束条件集合が展開されリダクションが試みられる．このなかでリダクション可能なものがすべて評価され終わると順次，軸システムの評価に移る．図 5 に示す歯車システムの拘束条件のうち (:t Self ($D?$ >= $2*S_d?$)) は軸直径が定まるまで suspend され，軸システムの拘束条件集合と併合して評価されることになる．

```
(:g Self (La := L / 2))
(wait ()
      (fget (Gsa (class gearsys)
                 :
(wait (Ta Fsa L La Na C)
      (fget (Ssa (class shaftsys)
                 (torque Ta)
                 (force Fsa)
                 :
(wait (Tb Fsb L La Nb C)
      (fget (Ssb (class shaftsys)
                 :
```

図 8 段階的詳細化法
Fig. 8 Stepwise refinement.

7. むすび

本論文では，拘束条件リダクション法とよぶ設計計算のモデルを基礎に開発した汎用的機械設計システムについて述べた．そして歯車減速機を例にとりその設計システムを記述し，保守性，拡張性，システムの使い勝手ともに優れていることを確認できた．現在このシステムは 40 klips の prolog 処理系上で二段歯車減速システムを 30 秒ないし数分で設計できる能力をもっており，小規模な応用には実用に耐えると考えている．しかし大規模な応用には，拘束条件集合の直接的解法やプラン知識を導入する必要があり，今後の課題である．

謝辞 日頃ご討論いただく九州大学情報処理教育センター荒牧重登氏に感謝する．

参考文献

1) Sussman, G. J. and Steele, G. L., Jr.: CONST

* 修飾付き変数 $Z?$ は変数とはユニファイされるが非変数とは値をもつまでユニファイされない．

RAINTS—A Language for Expressing Almost-Hierarchical Descriptions, *Artif. Intell.*, Vol. 14, pp. 1-39 (1980).
2) Borning, A.: THINGLAB—An Object-Oriented System for Building Simulation Using Constraints, Proc. 5th IJCAI, pp. 497-498 (1977).
3) 沖野教郎：自動設計の方法論，p. 192, 養賢堂，東京 (1982).
4) Okino, N., Kakazu, Y. and Kubo, H.: TIPS—Technical Information Processing System for CAD/CAM in Kitakawa, T. (ed.): *JARECT*, Vol. 7, *Computer Science & Technologies*, pp. 204-224, Ohmsha, Ltd. and North-Holland Pub. Co., Tokyo (1983).
5) 長澤，古川，荒牧：論理プログラミングを基礎とした設計システム記述言語 ADL，情報処理学会論文誌，Vol. 25, No. 2, pp. 606-613 (1984).
6) Goldberg, A. and Robson, D.: *Smalltalk 80—The Language and Its Implementation*, p. 714, Addison-Wesley, Massachusetts (1983).
7) Bobrow, D. G. and Stefik, M.: *The Loops Manual*, *Preliminary Version*, Xerox Corp. (1983).
8) Steel, G.: The Definition and Implementation of a Computer Programming Language Based on Constraints, AI Labo., TR-595, Cambridge MIT (1979).
9) Doyle, J.: A Truth Maintenance System, *Artif. Intell.*, Vol. 12, No. 3, pp. 231-272 (1979).
10) Bruynooghe, M. and Pereira, L. M.: Deduction Revision by Intelligent Backtracking, in Campbell, J. A. (ed.): *Implementation of Prolog*, pp. 194-215, Ellis Horwood, Chichester (1984).
11) Cox, P. T.: Finding Backtrack Points for Intelligent Backtracking, in Campbell, J. A. (ed.): *Implementation of Prolog*, pp. 216-233, Ellis Horwood, Chichester (1984).

付録　設計システムの使用例

ここでは設計システムを用いて一段歯車減速システムを設計する例を示す．

設計問題

次のような一段歯車減速システムを設計せよ．入力動力 110 kW，減速比 2/5，入力回転数 600 rpm，運転条件 8 時間連続運転，軸長 150 mm，歯車を軸の一端から 60 mm の位置に置くものとする．また歯車のモジュール 5 mm，歯幅・モジュール比 15 とし，小

```
(a) (? rs#1 show given)
    (rs#1 one_step_reduction_sys
          (teeth_no_gen (14 20))
          (module_gen (5 5))
          (bm_ratio_gen (15 15))          ⇐ 設計要求
          (gear_material_gen all)
          (shaft_material_gen all)
          (in_power_kw 110)
          (reduction_ratio (2 / 5))
          (in_rpm 600)
          (length 150)
          (length_a 60)
          (condition 8h_continuous))

(b) (? rs#1 solve)
    ((fail 160) (generate 241) (propagate 294))
    time     4074ms.

(c) (? rs#1 show result)
    (rs#1 one_step_reduction_sys
          (parts ssa53 ssb57 gsa42))
                :
    (gsa42 gearsys
          (parts ga043 gb044))            ⇐ 設計解
    (ga043 gear
          (no_of_teeth 16)
          (module (mm) 5)
          (tooth_width (mm) 75)
          (material s35c))
                :

(d) (? rs#1 to_given)

(e) (? rs#1 set in_power_kw 140)          ⇐ 設計要求の変更・検証

(f) (? rs#1 verify)
    (fail: (:t ga043 (lewis 5 16 600 75 80
           5.681665e+02  2.513273e+00 s35c)))
    (fail: (:t bb056 (life_time 2.272668e+02
           0 600 8h_continuous 6207)))
    (fail: (:t sa058 (torsional_strength
           5.568033e+04  s30c 42)))
    ((fail 3) (generate 0) (propagate 67))
    time     735ms.

(g) (? ga043 cancel material)
    (? sa058 cancel material)
    (? bb056 cancel jis_name)

(h) (? rs#1 solve)
    ((fail 13) (generate 27) (propagate 59))
    time     950ms.

(i) (? rs#1 show result)
                :
    (bb056 bearing (jis_name 6307))
                :
    (sa058 type1_shaft                    ⇐ 再設計解
                :
           (material sf55))
                :
    (ga043 gear
           (material s45c))
                :
```

図 A1　設計システムの使用例
Fig. A 1　A sample design session.

歯車の歯数を14から20の範囲とせよ.

設計問題は**図 A1**(a)のように与える. 図中 rs#1 はこの設計対象につけたインスタンス名である. 設計解の探索は, このインスタンスに solve メッセージ(b)を送ることによって行われ, 設計解(c)を得ている. この結果は, 歯車の最小の歯数, 軸・歯車の最も強度の低い材料, 最も小型の軸受を示している. 次に(d)では設計結果(c)を与件とみなし, (e)では入力動力を 140 kW に変更している. この状態で設計解を検証してみると入力側歯車 ga043 がルイスの公式による歯の曲げ強さに, 出力側軸 sa058 が捩りモーメントに, 軸受 bb056 が寿命条件に, それぞれ耐えないことがわかる(f). 歯車システムの寸法を変更しないで設計条件を満足させるには, 歯車材料, 軸材料, 軸の JIS 記号名を再設計すればよい (g), (h). (i)においてこれらが変更されているのがわかる.

(昭和 60 年 3 月 13 日受付)
(昭和 60 年 7 月 18 日採録)