設計計算の範例とＡＤＬにおける実現法につ いて

長澤　勲＊　古川　由美子＊　荒牧　重登＊＊
＊九州大学中央計数施設　　＊＊九州大学情報処理教育センター

１．まえがき

設計作業[*1]とは，一般に構成，解析，評価，最適化といった過程を経て，性能，信頼性，使い易さ，経済性，加工性などの項目の調和をはかって行くものである[1,2]．現在までに開発されてきた設計支援システムの多くは，これらの過程のうち主として解析過程の支援を目的としている．この理由は，解析過程では汎用解析モデルを用いることができ，自動化し易く，効果も大きいからであると考えられる．これに対して構成過程[*2]では，所要機能・性能に対して必要な構造・構造諸元を決定するのが主題であり，また考慮しなければならない設計要件も様々である．このため，構成過程の支援は解析過程ほど統一的に行えず，建築，機械，電子回路などの設計には従来から，取り扱う対象の種類，考慮すべき設計要件ごとに多数の設計資料，設計公式が準備され利用されている．たとえば，設計資料を調べ部品の強度計算をし，使用可能な部品をカタログから探したり，あらかじめ解っている構成方法の中から適当なものを選択することなどである．　そこで，筆者らは構成過程のうちこれらの作業を支援するシステムＡＤＬ（Ａ　Ｄｅｓｉｇｎｅｒ’ｓ　Ｌａｎｇｕａｇｅ）を開発した．本論文ではこのシステムの基本概念，実現法ならびに設計知識の表現法について述べるものである．

構成過程を支援するシステムの試みは，問題解決法を用いたもの[11]と拘束条件解法を基礎にしたもの[2,12,13,14]が提案されている．前者は構成過程のうち主として所要機能・性能に対して機械や電子回路に必要な構造を試行錯誤に構成する構造設計に適しており，複雑な機能をもつ対象の設計には不可欠な手法である．しかし一般にこのレベルの設計知識は現状では十分整理されておらず実用化は容易ではない．後者は，機能と構造，所要性能と構造諸元等の拘束関係[*3]を利用するものであり，構造諸元の決定やパターン化された構造の選択に適している．また，これに必要な設計知識は，すでに設計資料，設計公式，設計規約などの形で十分整理されている[4]．ここでは後者の方法をとることにする[*4]．拘束条件知識を基礎にして，設計支援システムを構成するには，設計公式の数値・数式処理，設計資料検索，生成検証法，伝播法，段階的詳細化法などの解探索法，利用者との柔軟な対話環境を実現する必要がある．しかし，現在までに提案されている拘束条件を基礎としたシステム[2,12,13,14]はこれらの機能を個別に取り扱っており，統一的方法で解決しているとは言い難い．一方，論理プログラミングは，(1)数式等に限定せず一般的な拘束条件[*5]を適切に表現できる．(2)多様なデータフローや試行錯誤な設計計算を容易に表現できる[*6]．(3)データベース検索[15]，数式処理[16]，対象指向プログラミング[5,6]への適応性が高い．などの特徴がありここでの目的に最適である．筆者らは，ｐｒｏｌｏｇを核言語にとり，設計支援に必要な種々の機能を実現することにした．

2．ＡＤＬシステムの概要

ＡＤＬシステムは，図１に示すように２つのモジュールＫＢＥ（Ｋｎｏｗｌｅｄｇｅ ｂａｓｅ ｅｄｉｔｏｒ）とＣＲＳ（Ｃｏｎｓｔｒａｉｎｔｓ ｒｅｄｕｃｔｉｏｎ ｓｙｓｔｅｍ）から構成される．設計知識ベースには，設計要求，設計結果および設計知識が記憶され，ＫＢＥはこれらの編集・管理に使用される．ＣＲＳはｐｒｏｌｏｇに拘束条件リダクションおよび対象指向プログラミング手法を拡張したものであり，設計計算，設計資料検索，設計の検証などに使用する．ＣＲＳはこのほか数値計算，図形処理などを行うため外部システムとのインターフェイスを持っている．以下本稿では，ＫＢＥの詳細は省略し，主としてＣＲＳについて述べる．

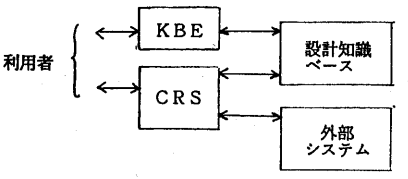


図1 ADLの構成

Fig. 1 An overview of ADL system.

3． 拘束条件リダクションシステム

設計問題は拘束条件集合の形で与えられる．ＣＲＳはこれをリダクションによって解くシステムである．ＣＲＳは拘束条件解法のうち，生成検証法，拘束条件伝播法[12,13]などの実現を意図しており，自動バックトラック，データフロー制御，コスト評価による効率改善等の機能を持っている．

3．1 記述形式

ＣＲＳはｐｒｏｌｏｇの手続の集まりとして実現し，特別の記述形式を用いない＊．

［拘束条件］拘束条件は素論理式で表現する．この論理式を，一般の論理式と区別するため以下あらためて拘束条件とよぶ．

［ＣＲＳの呼び出し］ＣＲＳの呼び出しにはｓｏｌｖｅ述語を用い，次のように記述する．

：－（ｓｏｌｖｅ Ｃ１ Ｃ２ … Ｃｎ）．

Ｃ１，Ｃ２，…，Ｃｎは拘束条件である．これらは，記述順序に意味を持たず集合として取り扱われる．

［リダクション・ルール］リダクション・ルールはｐｒｏｌｏｇの手続で表現する．リダクション・ルールを構成するｃｌａｕｓｅは次の形式で与えられる．

Ａ：－Ｇ１，…，Ｇｉ，！！，Ｇｉ＋１，…，Ｇｎ．

または，Ａ：－Ｇ１，…，Ｇｎ．（ｎ≧０）

Ａ，Ｇ１，…，Ｇｎは素論理式である．リダクション・ルールとしての意味を補強するため２つの基本述語！！，ｄｅｖａｌを導入する．！！はカット・オペレータの一種であり，リダクション・ルールの中にだけ書くことができる．！！は通常のカットの作用以外に，後述するように拘束条件の評価の意味を修飾する．ｄｅｖａｌ述語は，Ｇ１，…，Ｇｎを評価中に呼び出すことができ，リダクションによって置換すべき拘束条件集合をＣＲＳに通知するのに使用される．ｄｅｖａｌ述語の形式は（ｄｅｖａｌ Ｄ１ Ｄ２ … Ｄｎ）である．Ｄ１，…，Ｄｎは拘束条件である．

［変数修飾子］データフローを表現するために論理変数に修飾子を付加し，ｕｎｉｆｉｃ

＊以下，本論文では，ｌｉｓｐのＳ式を用いる．また，英大文字で始まる文字列は変数，英小文字で始まる文字列は定数である．

ａｔｉｏｎを制限する．変数に前置された？を入力修飾子，変数に後置された？を出力修飾子とよぶ＊＊．修飾子は変数の一部ではなく，たとえばＸを変数とするとき，？Ｘ，Ｘ？，Ｘは同一変数である．また，修飾子は変数の値が変数である限り遺伝する＊＊＊．ｕｎｉｆｉｃａｔｉｏｎにおける変数の代入規則を表１に示す．

表1　修飾付変数の代入規則

Table 1　Substitution rule of annotated variables

	非変数	Y?	?Y	Y
非変数	*	×	↑	↑
X?	×	×	×	↑
?X	←	×	×	×
X	←	←	×	←

×　失敗　　X?, Y?　出力修飾付変数
←　手続側変数への代入　　?X, ?Y　入力修飾付変数
↑　ゴール側変数への代入　　X, Y　通常の変数
＊　通常の項の統一化

３．２　リダクション手続

まず，一つの拘束条件を評価する手順について述べる．今，リダクションを試みる拘束条件をＣとする．

ＣはＣＲＳによってｐｒｏｌｏｇのｇｏａｌとみなして評価されるが，評価結果は次の３つに分類される．

(1) ｆａｉｌ　Ｃの評価がｐｒｏｌｏｇの意味で失敗であり，！！が実行されているとき．この場合積極的な否定情報を持つとみなす．

(2) ｓｕｃｃｅｓｓ　Ｃの評価に成功し，その評価中に（ｄｅｖａｌ　Ｄ１　…　Ｄｎ）を評価するとＣは｛Ｄ１，…，Ｄｎ｝θにリダクションされる．ただしθはＣの評価中に行った代入，｛｝は集合を表わす．ｄｅｖａｌを評価していなければＣは空集合にリダクションされる．

(3) ｓｕｓｐｅｎｄ　上記以外の場合，即ちｐｒｏｌｏｇの意味で失敗であり！！を実行していないとき．この場合何ら積極的情報を持たないとみなす．

＊＊修飾子の入力・出力はｕｎｉｆｉｃａｔｉｏｎにおける変数の代入方向を表わしている．

＊＊＊Ｃｏｎｃｕｒｒｅｎｔ　ｐｒｏｌｏｇを参考にしている．

```
(solve¦Cs):-(reduce Cs nil).

(reduce nil Ws):-(monitor Ws).
(reduce (C¦Cs) Ws):-
    (suspendp C),!,
    (reduce Cs (C¦Ws)).
(reduce (C¦Cs) Ws):-
    (exec C Ds),
    (append Cs Ds Cs1),
    (append Cs1 Ws Cs2),
    (reduce Cs2 nil).
```

(a)　リダクション手続

```
(monitor Ws):-
    (pick_equations Ws Equs Rest),
    (solve_equations Equs),!,
    (reduce Rest nil).
(monitor Ws):-(ans_oblist).
```

(b)　デッドロック解消手続

図2　リダクション手続

Fig. 2　Reduction procedure

［リダクション過程］リダクションの各段階における拘束条件集合をＳｉで表わすと，リダクションにおける拘束条件集合の系列は，Ｓ０，Ｓ１，Ｓ２，…，Ｓｉ，…，Ｓｎとかける．Ｓ０は初期拘束条件集合，各ＳｉはＳｉ－１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．評価してもｓｕｃｃｅｓｓまたはｆａｉｌする拘束条件がなくなったときリダクションは停止する．リダクション手続を図２(a)に示す．図中ｒｅｄｕｃｅ述語の二つの引数はそれぞれ未評価の拘束条件集合，ｓｕｓｐｅｎｄされた拘束条件集合を示す．またｓｕｓｐｅｎｄｐは引数がｓｕｓｐ

ｅｎｄか否かテストする述語，（ｅｘｅｃ　Ｃ　Ｄｓ）はＣをＤｓにリダクションする述語である．

［デッドロックの解消］拘束条件集合は個別にリダクションする方法では解けないことがある．しかし，連立方程式のように専用の解法を用いれば解ける場合，システムの使用者の介入によって拘束条件を追加すれば解ける場合などがある．図２(b)は連立方程式の解法を組み込んだ例である．

［リダクション過程の例］ここでは簡単な例を用いてシステムの動作を説明する．

図３は理想ダイオードと抵抗器の直列回路をダイオードの状態を仮定することにより解く例である．同図(c)において，拘束条件集合(1)の（ｄｉｏｄｅ＿ｓｔａｔｅ　Ｖｄ　Ｉ　Ｓ）はリダクション・ルール［ｄ１］を用いて，電流Ｉ＝０，状態Ｓ＝ｏｆｆと仮定し，（Ｖｄ≦０）にリダクションされ，(2)がえられる．以下同様にリダクションが進行するが(4)のリダクションはｆａｉｌし，別のリダクションの可能性を求めてバックトラックする．今度は，(1)のｄｉｏｄｅ＿ｓｔａｔｅは［ｄ２］のリダクションルールを用いてＶｄ＝０，Ｓ＝ｏｎを仮定し，（Ｉ≧０）にリダクションされる．以下，同様に繰り返し(8)で停止する．［ｄ１］，［ｄ２］の適用において変数値が決定されて行く順序が異なることに注意されたい．

４．　構造物の表現

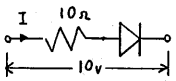
フレーム表現[8]は，構造物を表現する手段として適しており，次のような特徴があることが知られている．(1)対象指向な知識表現の自然な単位である．(2)手続附加により宣言的知識表現と，手続的知識表現が融合できる．(3)遺伝階層の利用により，知識表現がモジュール化できる．

機械や電子回路の設計分野では，対象指向に設計法や設計公式が開発されており，これらの知識をフレームを用いて表現するのは自然である．ここでは，遺伝階層と拘束条件リダクション手法を融合したフレームを提案する．図４にフレームの記述形式を示す．スロットは対象の属性や，部分を示すのに使用しスロット名，変数の対からなる．スロットと拘束条件は変数を共有する．ａｋｏスロットは上位フレームを参照し，スロットおよび拘束条件を継承する．フレームの機能はｆｇｅｔ述語により使用する．この述語は個々の対象物の表現であるフレーム・インスタンスの管理と，拘束条件リダクションの２つの機能を持っている．

［ｆｇｅｔ手続の概要］　システムには生成されたフレーム・インスタンスを管理する対象リスト（＊ｏｂｌｉｓｔ）というデータ構造がある．ｆｇｅｔ述語は次の作業を行う．(1)フレーム参照の中で指定した識別子に対応するフレーム・インスタンスが対象リストになければ新たにフレーム・インスタンスを生成し対象リストにのせる．またフレームの拘束条件部を取り出す．(2)フレーム参照とフレーム・インスタンスのスロット値を，ｕｎｉｆｙする．図５にｆｇｅｔ手続を示す．図中ｖａｌｕｅ，ｓｅｔ＿ｏｎｂｔはそれぞれ，データ構造の参照，書き替えを行う述語である．また，バックトラック時にデータ構造は復元される．

［フレームの使用例］図６は図３の問題をフレームを用いて表現したものである．同図(c)(1)のｆｇｅｔを評価すると(b)のｓｅｒｉｅｓ

```
:-(solve (Vd + Vr = 10)
         (resistor_state Vr I 10)
         (diode_state Vd I S)).
```



(a)問題の表現

```
[d1]   (diode_state Vd 0 off) :- (deval (Vd <= 0)).
[d2]   (diode_state 0 Id  on) :- (deval (Id >= 0)).
[r]    (resistor_state Vr Ir R) :- (Vr = Ir * R).
[e1]   (X = Y) :- (free_of_var (X = Y)),!!,(test_equ (X = Y)).
[e2]   (X = Y) :- (single_var (X = Y)),(solve_equ (X = Y)).
[>=]   (?X >= ?Y) :- !!,(greater_or_equal X Y).
[<=]   (?X <= ?Y) :- !!,(less_or_equal X Y).
```

(b)リダクション・ルール

```
(1)  (Vd + Vr = 10)
     (resistor_state Vr I 10)
     (diode_state Vd I S)
          ⇓  [d1] I=0 S=off
(2)  (Vd + Vr = 10)
     (resistor_state Vr 0 10)
     (Vd <= 0)
          ⇓  [r]  Vr=0
(3)  (Vd +0 = 10)
     (Vd <= 0)
          ⇓  [e2]  Vd=10
(4)  (10 <= 0)
          ⇓  [<=] fail
          ×
```

```
          ⇓  [d2] Vd=0 S=on
(5)  (0 + Vr = 10)
     (resistor_state Vr I 10)
     (I >= 0)
          ⇓  [e2] Vr=10
(6)  (resistor_state 10 I 10)
     (I >= 0)
          ⇓  [r] I=1
(7)  (1 >= 0)
          ⇓  [>=]
(8)     nil
```

(c)リダクション過程

図3リダクション過程の例

Fig. 3 An example of reduction process.

```
<フレーム> ::=
  (frame<フレーム名>
     (Obj [(ako <フレーム名>・・・・<フレーム名>)]
          <スロット>・・・・<スロット>)
     (proc <拘束条件>・・・・<拘束条件>)))
<フレーム名>::= <文字定数>
<スロット> ::= (<スロット名><変数>)
<スロット名>::= <文字定数>
<フレーム参照>::=
  (fget<オブジェクト参照>・・・・<オブジェクト参照>)
<オブジェクト参照>::=
  (<識別子> [(ako <フレーム名>)]
   <スロット参照>・・・・<スロット参照>)
<識別子>     ::=<文字定数>|<変数>
<スロット参照>::= (<スロット名><prologの項>)
```

図4 フレーム定義と参照の形式

Fig. 4 Syntax of frame definition and referrence.

```
(fget|Objs):-(match Objs Cs),
    (deval|Cs).

(match nil nil).
(match (Obj|Objs) Cs):-
    (value *oblist Obl),
    (search_obl Obj Obl Ins),
    (frame_unify Obj Ins),
    (match Objs Cs).
(match (Obj|Objs) Cs2):-
    (make_instance Obj Ins Cs),
    (frame_unify Obj Ins),
    (value *oblist Obl),
    (set_onbt *oblist (Ins|Obl)),
    (match Objs Cs1),
    (append Cs Cs1 Cs2).
```

図5 fget手続

Fig. 5 The fget procedure

__cir, resistor, diodeフレームと一致がとられ①のフレーム・インスタンスs1, r1, d1が生成される. 同時に各フレームに含まれる拘束条件集合を集めて⑵にリダクションされる. ⑵のfgetは再び優先的に評価されるが, 今度はフレーム・インスタンスが既に存在するのでスロットの値がunifyされるのみである. 以下図3と同様に評価が進行する.

5. CRSの表現能力

本章では, 設計計算の範例を取り上げ, CRSの表現能力について述べる.

[生成・検証法] 機械や電子回路では, 従来から設計公式を用いた設計計算が行われている. この設計計算の問題点の一つは, 設計仕様や設計公式は一般に非線形方程式, 不等式であることが多く, 解析的には容易に解けないことである. 幸い, 設計問題の場合, 一般解が必要ではないので, 生成・検証技法 (解またはその一部を仮定し, 方程式を解き, 不等式を用いて検証する方法) を用い, 解を試行錯誤に探索することが行われている[3]. CRSでは, 仮定の生成・伝播・検証はすべてリダクション・ルールによって表現でき, 生成・検証法の実現は容易である. 図7(a)は一組の平歯車の設計プログラムである. この例では拘束条件を便宜的に回転数と速度比

```
:-(solve (fget
          (s1 (ako series_cir)
              (volt 10)
              (ele_a r1)
              (ele_b d1))
          (r1 (ako resistor)
              (resistance 10))
          (d1 (ako diode))))
```

(a)問題の表現

```
(frame tte
   (Obj (volt V)
        (amp I)
        (power P))
   (proc (P = I * V)))

(frame diode
   (Obj
    (ako tte)
    (volt Vd)
    (amp Id)
    (state S))
   (proc (diode_state Vd Id S)))

(frame resistor
   (Obj
    (ako tte)
    (volt Vr)
    (amp Ir)
    (resistance R))
   (proc (Vr = Ir * R)))

(frame series_cir
   (Obj
    (ako tte)
    (volt V)
    (amp I)
    (ele_a Ea)
    (ele_b Eb))
   (proc
    (V = Va + Vb)
    (fget (Ea (ako tte)
              (volt Va)
              (amp I))
          (Eb (ako tte)
              (volt Vb)
              (amp I)))))

    tte : two_terminal_element
```

(b)フレーム表現

(1)拘束条件集合

```
(fget
 (s1 (ako series_cir)
     (volt 10)
     (ele_a r1)
     (ele_b d1))
 (r1 (ako resistor)
     (resistance 10))
 (d1 (ako diode)))
```

①対象リスト

```
(d1 (ako diode)
    (volt  Vd)
    (amp  Id)
    (power  P3)
    (state  S))
(r1 (ako resistor)
    (volt Vr)
    (amp Ir)
    (power P2)
    (resistance 10))
(s1 (ako series_cir)
    (volt 10)
    (amp I1)
    (power P1)
    (ele_a r1)
    (ele_b d1))
```

⇓

(2)拘束条件集合

```
(P1 = I1 * 10)
(10 = Va + Vb)
(fget
 (r1 (ako tte)
     (volt Va)
     (amp  I1))
 (d1 (ako tte)
     (volt Vb)
     (amp  I1)))
(P2 = Ir * Vr)
(Vr = Ir * 10)
(P3 = Id * Vd)
(diode_state Vd Id S)
```

②対象リスト

```
(d1 (ako diode)
    (volt Vb)
    (amp I1)
    (power P3)
    (state S))
(r1 (ako resistor)
    (volt Va)
    (amp I1)
    (power P2)
    (resistance 10))
(s1 (ako series_cir)
    (power P1)
    (ele_a r1)
    (ele_b d1))
```

⇓

(3)拘束条件集合

```
(P1 = I1 * 10)
(10 = Va + Vb)
(P2 = I1 * Va)
(Va = I1 * 10)
(P3 = I1 * Vb)
(diode_state Vb I1 S)
```

(c)フレームのリダクション過程

```
(d1 (ako diode)
    (volt 0)
    (amp 1)
    (power 0)
    (state on))
(r1 (ako resistor)
    (volt 10)
    (amp 1)
    (power 10)
    (resistance 10))
(s1 (ako series_cir)
    (volt 10)
    (amp 1)
    (power 10)
    (ele_a r1)
    (ele_b d1))
```

(d)解

図6　フレーム表現とリダクション過程の例

Fig. 6 An example of frame representation and its reduction process

```
(frame gear
   (Obj (rpm N)
        (power_kw L)
        (module M)
        (no_of_teeth Z)
        (material Mat))
   (proc (gear_shape M Z D)
         (lewis M Z N D L Mat)))

(frame gearsys
   (Obj (reduction_ratio U)
        (power_kw L)
        (driving_gear Ga)
        (drived_gear Gb))
   (proc (fget
          (Ga (ako gear)
              (module M)
              (no_of_teeth Za)
              (rpm Na)
              (power_kw L))
          (Gb (ako gear)
              (module M)
              (no_of_teeth Zb)
              (rpm Nb)
              (power_kw L)))
         (Nb = Na * U)
         (reduction_ratio U Za Zb)))


(reduction_ratio ?U Za ?Zb):-!!,
   (Zb * U = Za),(13 < Za),(Za < 101),
   (13 < Zb),(Zb < 101).
(reduction_ratio U ?Za ?Zb):-!!,
   (Zb * U = Za),(13 < Za),(Za < 101),
   (13 < Zb),(Zb < 101).
(reduction_ratio ?U Za Zb):-!!,
   (for Za 14 100),(Zb * U = Za),
   (13 < Zb),(Zb < 101).

(gear_shape M ?Z D):-
   (module M),
   (D = M * Z).

(lewis ?M ?Z ?N ?D ?L Mat):-
   (60000 * V = 3.141592 * D * N),
   (toothshape_coef Z Kz),
   (velosity_coef V Kv),
   (gear_material Mat _ _ Sigwb _ _),
   (N * T = 9.74 * 10000 * L),
   (M ** 3 * Kv * 10 * Z * Kz * Sig =
    2 * 1.25 * T * 10),
   (Sig <= Sigwb).
```

(a) 生成検証法

```
(frame gear
 (Obj (rpm N)
      (power_kw L)
      (module M)
      (no_of_teeth Z)
      (material Mat)
      (tooth_width B)
      (pitch_circle_diameter D)
      (pitch_circle_force Fs)
      (shaft_hole_diameter Ds)
      (type Gt)
      (shape Sp))
 (proc (gear_shape M Z D B)
       (lewis M Z N D L Mat Fs)
       (wait (M Z Ds Mat Fs)
         (select Gt (web_type arm_type))
         (fget Sp
            (ako Gt?)
            (module M)
            (no_of_teeth Z)
            (teeth_width B)
            (material Mat)
            (pitch_circle_force Fs)
            (shaft_hole_diameter Ds)))))

(wait Vars|Cs):-(free_of_var Vars),(deval|Cs).
```

(b) 段階的詳細化法

```
(X=Y) :- (free_of_variable (X=Y)),!!,
         (approx_equal X Y).
(X=Y) :- (single_variable (X=Y)),!,
         (solve_equation (X=Y)).
(X=Y) :- (select_variable (X=Y) Z),
         (solve_equation_with (X=Y) Z).
```

(c) 拘束条件伝播法

```
(X=Y) :- (free_of_variable (X=Y)),!!,
         (approx_equal X Y).
(X=Y) :- (single_variable (X=Y)),!,
         (solve_equation (X=Y)).
(X=Y) :- (multiple_operator (X=Y)),
         (decompose_equ (X=Y) Eq1 Eq2),
         (deval Eq1 Eq2).
```

(d) 一次算子式への分解

図7 プログラムの例題

Fig. 7 Programming examples

（＝），歯数と速度比（ｒｅｄｕｃｔｉｏｎ＿ｒａｔｉｏ），歯車形状（ｇｅａｒ＿ｓｈａｐｅ），歯の曲げ強さ（ｌｅｗｉｓ）に分けている．歯数対，モジュール，歯車材料をそれぞれ，ｒｅｄｕｃｔｉｏｎ＿ｒａｔｉｏ ｇｅａｒ＿ｓｈａｐｅ，ｌｅｗｉｓの各ルールによって仮定し，歯の曲げ強さの検証はｌｅｗｉｓのルールによって行っている．

［段階的詳細化法］段階的詳細化法[19]は解探索の計算量を減少するのに効果的である．ＣＲＳでは変数に値が代入されることを待って詳細化の時点を制御することができる．図7(b)は歯車の基本設計が完了するのを待って詳細形状の設計を行う例である．

［状態仮説法］状態仮説法は電子回路の素子モデル，材料力学の部材モデルなどに使用される．前出の図３，６はこの手法の例である．

［伝播法］Ｓｕｓｓｍａｎらは線形連立方程式を解く一方法[12,13]を提案している．次の問題は数値を伝播させて解くことはできないが連立方程式の変数消去法を用いれば容易に解くことができる（図7(c)）．

：－（ｓｏｌｖｅ　（Ｘ＋Ｙ＝３）（Ｙ＝２＊Ｘ））

これには方程式を解くリダクション・ルールに次の機能があればよい．(1)変数がなければ等式を検証する．(2)一変数方程式は無条件に解く．(3)多変数方程式は一変数について解く．［非線形連立方程式解法］野口らは設計計算に必要な非線形方程式の解法として数式を単項演算子式に分解し，しかる後，数値計算を行う方法[7]を提案している．この方法は数式処理と数値計算を組み合わせたものである．前半の処理は図7(d)のリダクション・ルールによって，後半は図2(b)のデッドロック解消手続ｍｏｎｉｔｏｒから数値計算ライブラリを呼び出すことによって容易に実現できる．

6．ＣＲＳの最適化

拘束条件集合の評価は，データフローの制限を満足しさえすればどのような順序で行っても同じ結果を得ることができる．しかし，生成検証法のようにバックトラックを多用する応用では，効率に大巾な影響を与える．ここでは拘束条件に動的コストを定義し，これを用いて拘束条件の評価順序を決定する．

各拘束条件に動的コストｃ＝ｎ・ｓを定義する．ｓはリダクション・ルールに定義したコストであり，ルール評価に必要な計算量，非決定性の程度によって決定しておく．ｎは拘束条件に出現する変数の個数であり，リダクションの進行とともに動的に変化する．ＣＲＳ起動時および，ｄｅｖａｌ手続き中では，拘束条件集合をコストｃの小さい順にソートする．リダクションの各段階では最小コストの拘束条件を求めている．

コスト制御の効果を調べるため歯車減速機の基本設計を行うプログラム[10]を用い簡単な実験を行った．制御を行わない場合に比較して１．８ないし３倍程度の効率改善が見られた．またオーバーヘッドは１６～２５％であった．以上の結果により，コスト評価による効率改善法はある程度効果があるといえる．

7．システム・プログラム

ＡＤＬは，ＦＡＣＯＭ　ＯＳⅣ／Ｆ４　ＬＩＳＰ[8,9]上に実現されている．ＫＢＥはシステムで取り扱うすべての対象を編集できるようにｌｉｓｐ構造エデイタを拡張したものであり，ｌｉｓｐで実現されている．ＣＲＳはｌｉｓｐ内装アセンブラを用いて実現されたｐｒｏｌｏｇシステム上に実現されている．ＣＲＳの大略を付録１に示す．

8．　あとがき

本論文では，設計システム記述言語ＡＤＬを提案し，基本概念，実現法，表現能力について述べた．筆者らは，ＡＤＬを用いて機械設計の教育を目的として，歯車減速機の設計システムを開発した[10]．この開発を通して，設計計算を統一的方法で表現し，可用性，拡張性，保守性の高い設計システムの構成法を考案するという初期の目的はひとまず達成できたことを確認している．しかし，より実際的な，設計システムを実現するにはより一層の機能拡張が必要である．機能拡張の一つとしてＴＡＬＫと呼ぶ対象指向プログラミングの手法を用いた会話型モジュールの開発を始めている．また，教育用応用システムとして材料力学のコンサルテーションシステムの開発を行う予定である．

謝辞　本論文をまとめるにあたり，文献を提供され，貴重な助言をいただいたＩＣＯＴ第二研究室古川康一室長を始め同研究室の諸氏，ならびに日頃助言をいただく，九州大学工学部吉田将教授，牛島和夫教授，中央計数施設大槻説乎助教授，大型計算機センター松尾文碩講師に謝意を表わす．

参考文献

1）和久井，下村：電子回路のＣＡＤ，ｐ．２３６，日刊工業新聞社（１９７２）．

2）沖野　教郎：自動設計の方法論，ｐ．１９２，養賢堂（１９８２）．

3）小川　潔：　機械設計システムのプログラミング，ｐ．２０６，森北出版（１９７７）．

4）小川　潔：　機械設計システム，ｐ．２４０，森北出版（１９７３）．

5）竹内，古川，Ｓｈａｐｉｒｏ，Ｅ．Ｙ．：Ｃｏｎｃｕｒｒｅｎｔ　ｐｒｏｌｏｇによるオブジェクト指向プログラミング，ロジックプログラミングコンファレンス（１９８３）．

6）服部　隆：オブジェクト指向的なｐｒｏｌｏｇプログラミング，ロジックプログラミングコンファレンス（１９８３）．

7）野口，安藤：非線形連立方程式の自動求根プログラム，情報処理学会論文誌，Ｖｏｌ．２４，ｐｐ．１３７－１４２（１９８３）．

8）計算機マニュアル　ＦＡＣＯＭ　ＯＳⅥ／Ｆ４　ＬＩＳＰ手引き書．

9）長澤，古川：ｐｒｏｌｏｇ系言語　ＡＤＬ（１）－　ＡＤＬの概要とインタプリタ－，九州大学大型計算機センタ広報，Ｖｏｌ．１６，ｐｐ．２５２－２７９　（１９８３）．

１０）長澤，古川，他：ＡＤＬによる機械設計システム，情報処理学会知識工学研資３１－８（１９８３）

11)McDermott,D.:Circuit Design as Problem Solving, in Latombe,J.C.,(eds) Artificial Intelligence and Pattern Recognition in Computer Aided Design, North Holland,Amsterdam,pp.227-245 (1978).

12)Stallman,R.M. and Sussman,G.J. : Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis, Artificial Intelligence,Vol.9,pp.135-196 (1977).

13)Sussman,G.J. and Steele Jr,G.L.: CONSTRAINTS-A Language for Expressing Almost-Hierarchical Descriptions,Artificial Intelligence, Vol.14,pp.1-39 (1980).

14)Borning,A.:THINGLAB--An Object-Oriented System for Bulding Simulation using Constraints, Proc. 5th IJCAI, pp.497-498 (1977).

15)Gallaire,H.,Minker,J. and Nicolas J.M. : An Overview and Introduction to Logic and Data Bases, in Gallaire,H.and Minker,j.(eds),Logic and Data Bases, Plenum,NewYork,pp.3-30 (1978).

16)Bundy,A. and Welham,B. : Using Meta-level Inferrence for Selective Application of Multiple Rewrite Rule Sets in Algebraic Manipulation,Artificial Intelligence, Vol.16,pp.189-212 (1981).

17)Shapiro,E.Y.: A Subset of Concurrent Prolog and Its Interpreter, ICOT Technical Report TR-003 (1983).

18)Winston,P.H.: Artificial Intelligence, Adisson-Wesley,Reading(Mass.) (1977).

19)Stefik,M.:Planning with Constraints(MORGEN:Part 1), artificial intelligence,Vol.16,pp.111-139 (1981).

付録１．ＣＲＳインタプリタの概要

ＣＲＳは，４つのデータ構造を持っている．Ｒｅａｄｙ－Ｒｔ，Ｗａｉｔ－Ｗｔはそれぞれ，未評価な拘束条件集合，評価を行ったがｓｕｓｐｅｎｄされた拘束条件集合を表わし，それぞれｄ＿ｌｉｓｔにより実現している．＊ｃｌｉｓｔ，＊ｏｂｌｉｓｔはＣＲＳとリダクション・ルール間の交信に使用される広域的データ構造であり，拘束条件集合，フレーム・インスタンス集合を表わしている．Ｒｅａｄｙ－Ｒｔ，Ｗａｉｔ－Ｗｔ，＊ｃｌｉｓｔは効率改善のため６章で述べたコストによってソートされている．

```
(solve¦Constraints) :-
      (append Constraints Rt Ready),
      (sort Ready-Rt Sorted-St),
      (set_onbt *clist Rt-Rt),
      (reduce Sorted-St Wt-Wt).

(reduce Rt-Rt Wait-Wt) :-
      (var Rt),!,(monitor Wait-Wt).
(reduce (R¦Ready)-Rt Wait-Ready) :-
      (set_onbt *clist Wait-Rt),
      off_flag,R,!&,
      (value *clist Rdy1-Rt1),.
      (find_mincost_cs Rdy1-Rt1 Rdy2-Rt1),
      (reduce Rdy2-Rt1 Wt-Wt).
(reduce (R¦Ready)-Rt Wait-(R¦Wt)) :-
      off_flagp,(reduce Ready-Rt Wait-Wt).

(monitor Wait-Wt) :-
      (pick_equations Wait-Wt Equs Ready-Rt),
      (solve_equations Equs),!,
      (reduce Ready-Rt Wt-Wt).
(monitor Wait-Wt) :- (ans_oblist).

(deval¦Cs) :-
      (append Cs Rt Ready),
      (deval1 Ready-Rt).

(fget Objs) :-
      (match Objs Cs-Rt),
      (deval1 Cs-Rt).

(deval1 Cs-Ready) :-
      (value *clist Ready-Rt),
      (sort Cs-Rt Sorted-St),
      (set_onbt *clist Sorted-St).
```

＊　ｏｆｆ＿ｆｌａｇはシステムフラグを初期設定し，ｏｆｆ＿ｆｌａｇｐはテストする。リダクション・ルール実行中！！が評価されるとこのフラグが反転することを利用してｆａｉｌとｓｕｓｐｅｎｄの区別をしている。

＊＊　！＆はカットオペレータの一種であり，親レベルのバックトラック点だけを除去する。

図A. 1　CRSインタプリタの概要

Fig. A. 1 Outline of CRS interpreter.

＊1　建築，機械，電子回路などの設計を意図している．

＊2　構成，モデリング，合成などの用語が使用されているが，ここでは，対象分野に依存しない用語〝構成〟を用いる．

＊3　たとえば，歯車減速機の場合，直交する２軸間に動力を伝達する（機能）には，傘歯車，ウオーム歯車減速機（構造）が使用できる．また，所要伝達動力，回転数（性能）と歯車のモジュール，材料，焼き入れ硬さ（構造諸元）との間には拘束条件がある．

＊4　この方法の有効さは，対象領域の性質に依存する．機械設計の例では，実際に多用されているバリアントアプローチ（既存の機械の構造や構造諸元の一部を変更して用いる設計法）に有効である．

＊5　たとえば歯車減速機の場合，入出力軸の方向（平行，直交など）と歯車の種類（平，かさ，ウオーム），また減速比と減速機の構造には非数値的拘束関係がある．

ｕｎｉｆｉｃａｔｉｏｎおよびｂａｃｋｔｒａｃｋによる．