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†

長澤 勲†† 古川由美子†† 荒牧重登†††

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．筆者らは，設計公式に代表される知識を一般に拘束条件知識としてとらえ，リダクション手法によって解探索を行うシステムを開発した．このシステムは設計公式の数値・数式処理，設計資料検索，解探索を統一的に表現でき，設計計算をプログラム化するのに有効である．この結果，本手法による設計システムは，FORTRAN などで書かれた現存の設計システムに比べ，次のような優れた性質をもっている．(1) 設計者は設計知識や仕様を拘束条件の形で設計システムに与え，これを満足する解をシステムの支援のもとに探索することができる．このため設計者は設計，設計の検証，設計の部分変更など多目的に設計システムを利用できる．(2) 拘束条件解法はリダクション手法を用いており，種々の設計計算を統一的に表現できる．(3) 設計知識はシステム内で対象指向に管理され，高度にモジュール化されており可用性，拡張性，保守性がよい．なお，本システムの主要部は Prolog によって，実現されている．

1. まえがき

設計作業* 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，使いやすさ，経済性，加工性などの項目の調和を図っていくものである[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，自動化しやすく，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**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さまざまである．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，あらかじめわか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)~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*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**．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)~14)]はこれら

† ADL: A Designer's Language Based on Logic Programming by ISAO NAGASAWA, YUMIKO FURUKAWA (Computation Center, Kyushu University) and SIGETO ARAMAKI (Educational Center for Information Processing, Kyushu University).

†† 九州大学中央計数施設

††† 九州大学情報処理教育センター

* 建築，機械，電子回路などの設計を意図している．

** 構成[1)]，モデリング[2)]，合成[12),13)] 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．

* 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，ウォーム歯車減速機（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，焼入れ硬さ（構造諸元）との間には拘束条件がある．

** 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．

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いいにくい．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* 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**．(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**に示すように二つのモジュール 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手法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フェースをもっている．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．

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もっている．

3.1 記述形式

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

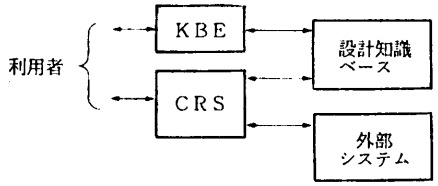


図 1 ADL の構成
Fig. 1 An overview of ADL system.

[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改めて拘束条件とよぶ．

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

:−(solve C_1 C_2 ⋯ C_n).

$C_1, C_2, \cdots, C_n$ は拘束条件である．これらは，記述順序に意味をも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．

$A :- G_1, \cdots, G_i, !!, G_{i+1}, \cdots, G_n.$

または，$A :- G_1, \cdots, G_n.$ $(n \geqq 0)$

$A, G_1, \cdots, G_n$ 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二つの基本述語!!, 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，$G_1, \cdots, G_n$ 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 D_1 D_2 ⋯ D_n) である．$D_1, \cdots, D_n$ 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，unification 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．

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．いま，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，評価結果は次の三つに分類される．

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

---

* 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，ウォーム)，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．

* 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．

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

*** Concurrent Prolog*を参考にしている．

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

（2） **success** Cの評価に成功したとき．Cの評価中に (deval $D_1 \cdots D_n$) を評価するとCは$\{D_1, \cdots, D_n\}\theta$にリダクションされる．ただし$\theta$はCの評価中に行った代入，{ }は集合を表す．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） **suspended** 上記以外の場合，すなわち Prolog の意味で失敗であり!!を実行していないとき．この場合なんら積極的情報をもたないとみなす．

[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 S_i で表すと，リダクションにおける拘束条件集合の系列は，$S_0, S_1, S_2, \cdots, S_i, \cdots, S_n$ と書ける．S_0 は初期拘束条件集合，各 S_i は S_{i-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 suspended 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，($Vd \leqq 0$) にリダクションされ，(2)が得られる．以下同様にリダクシ

```
(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).
```

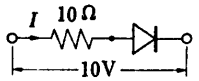
（a） リダクション手続

```
(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 of ADL/CRS.
(A B|C) などは Edinburgh Prolog のリスト表現 [A B|C] と同じ意味である．

```
:-(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.

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，($I \geqq 0$) 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い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[18]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)手続き付加により宣言的知識表現と，手続き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二つの機能をもっている．

[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)フレーム参照のなかで指定した識別子に対応

```
〈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 reference.

```
(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.

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_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(2)にリダクションされる．(2)の fget は再び優先的に評価されるが，今度はフレーム・インスタンスがすで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

```
:-(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

```
(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

```
(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)))
```

⇓

(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)
```

⇓

(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)
```

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 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))
```

(c) フレームのリダクション過程

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） 一演算子式への分解

図 7 プログラムの例題
Fig. 7 Programming examples.

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（=），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，効率に大幅な影響を与える．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 $c=n\cdot s$ 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 OS IV/F4 LISP 上に実現されている[8),9)]．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 構造エディ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所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いっそう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，東京 (1972).
2) 沖野教郎：自動設計の方法論，p. 192，養賢堂，東京 (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，東京 (1977).
4) 小川 潔：機械設計システム，p. 240，森北出版，東京 (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フンァレンス (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, No. 2, pp. 137-142 (1983).
8) 富士通(株)：計算機マニュアル FACOM OS IV /F4 LISP 手引き書 (1977).
9) 長澤，古川：Prolog 系言語 ADL(1)—ADL の概要とインタプリター，九州大学大型計算機センタ広報，Vol. 16, No. 3, 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. (ed.), *Artificial Intelligence and Pattern Recognition in Computer Aided Design*, pp. 227-245, North Holland, Amsterdam (1978).
12) Stallman, R. M. and Sussman, G. J.: Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis, *Artif. Intell.*, Vol. 9, pp. 135-196 (1977).
13) Sussman, G. J. and Steele Jr., G. L.: CONSTRAINTS—A Language for Expressing Almost-Hierarchical Descriptions, *Artif. Intell.*, Vol. 14, pp. 1-39 (1980).
14) Borning, A.: THINGLAB—An Object-Oriented System for Building 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*, pp. 3-30, Plenum, New York (1978).
16) Bundy, A. and Welham, B.: Using Meta-level Inference for Selective Application of Multiple Rewrite Rule Sets in Algebraic Manipulation, *Artif. Intell.*, 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 (1977).
19) Stefik, M.: Planning with Constraints (MOLGEN: Part 1), *Artif. Intell.*, Vol. 16, pp. 111-139 (1981).

付録　CRS インタプリタの概要

CRS は，四つのデータ構造をもっている．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．

付図 1　CRS インタプリタの概要
Fig. A. 1　Outline of CRS interpreter.

(昭和 58 年 8 月 1 日受付)
(昭和 59 年 2 月 14 日採録)