ソフトウェア基礎論　34－3
（1990．　3．　9）

モデル推論による制約付き再帰的図形の学習

劉樹苓
京都大学　数理解析研究所

コンピュータ・グラフィックスの分野では、制約を用いて図形を定義することが一般的である。特に、再帰的に定義される図形においては、制約が再帰的構造を構成する要素となっている。本論文では、そのような制約付き再帰的図形を定義するプログラムを、図形の正例と反例から合成する学習問題について考察する。まず、制約付き再帰的図形を記述するために、$CLP(\mathrm{X})$ の一つである $CLP(\mathcal{G})$ に制限と拡張を加えた制約論理プログラミング言語 $CLP'(\mathcal{G})$ を定義する。次に、Shapiro のモデル推論の方法を用いて、図形の正例と反例から $CLP'(\mathcal{G})$ プログラムを合成する方法について述べる。特に、洗練操作子としては、Muggleton の absorption と intra-construction 操作子を拡張したものを用いる。

Model Inference of Constrained Recursive Figures

Shuling Liu
Research Institute for Mathematical Sciences, Kyoto University
Kitashirakawa,Sakyo-ku,Kyoto 606,Japan

In computer graphics, the constraint technique is widely used to define a class of figures. For recursively defined figures, constraints work as components of their recursive structure. In this paper, we study the problem of synthesizing a program that defines a class of constrained recursive figures from true and false examples of the class. We first define a constraint logic programming language $CLP'(\mathcal{G})$ for defining constrained recursive figures, which is based on $CLP(\mathcal{G})$, a member of the family $CLP(\mathrm{X})$ of constraint logic programminglanguages. We then extend Shapiro's model inference method to synthesize a program in $CLP'(\mathcal{G})$ from true and false examples. As a refinement operator, we also extend Muggleton's absorption and intra-construction operators.

1 はじめに

制約とは、システムが満たさなければならない関係のことであり、物理系のシミュレーション、知識表現、CAD、ユーザ・インターフェース、コンピュータ・グラフィックスなどの分野において用いられる概念である。特に、コンピュータ・グラフィックスの分野においては、制約を用いて図形を定義する方法が一般的である。例えば、平行四辺形は、互いに向かい合う辺が等しいという制約が付加された四辺形であると定義される。すなわち、制約は図形の構造を構築するための手段の一つであるということができる。従って、以下で述べるarch の例のように、制約が図形の再帰的構造の要素になることは極めて自然なことである。一方、制約付き図形を入力するための graphics editor が数多く開発されている [12]。これらの editor を用いると、図形を描きながら同時に制約も簡単に入力することができる。しかし、これらの editor は、制約を含む再帰的図形を入力するための一般的な手段を提供してはいない。従って、現状では、制約を含む再帰的図形を入力するには、何らかのプログラミング言語を用いる方法しか存在しない。このような状況に対処するため、本論文では、制約付き再帰的図形の定義を、その正例と反例を用いて合成するという学習問題を考察するものである。

本論文の学習はモデル推論問題の一つの応用例ということができる。従って、本論文では図形を学習するために、モデル推論の解決方法を使う。また、制約付き再帰的図形を認識するために、$CLP(\mathbf{X})$ を基礎とする制約論理プログラミング言語 $CLP(\mathcal{G})$ に制限と拡張を加えた $CLP'(\mathcal{G})$ を用いる。

本論文の概要は以下の通りである。第 2 節ではモデル推論問題と本論文で用いる incremental なモデル推論アルゴリズムの枠組について簡単に紹介する。第 3 節では制約と制約付き再帰的図形を定義する。第 4 節では制約付き再帰的図形を認識するためのプログラミング言語 $CLP'(\mathcal{G})$ を詳しく定義する。第 5 節は本論文の中心である制約付き再帰的図形の学習の実現について述べる。最後に、第六節ではこの学習システムの問題点を分析し、将来の拡張について述べる。

2 モデル推論アルゴリズム

モデル推論問題は 60 年代から研究され始め、現在でも盛んに研究されている。モデル推論問題を解決するアルゴリズムはモデル推論アルゴリズムといわれる。論文 [1] では、sentence が Horn 節で、具体例が ground atom であるモデル推論問題に対して、incremental なモデル推論アルゴリズムの枠組と具体的なアルゴリズムを提案している。この枠組は学習の方法として本論文でも用いる。

incremental なモデル推論アルゴリズムの枠組

T は Horn 節の有限集合とする
repeat
　次の事実を調べる
　repeat
　　while **T** は強過ぎる　*do* **T** を弱める
　　while **T** は弱過ぎる　*do* **T** を強める
　until 既知の事実に対して、
　　T は強過ぎないし、弱過ぎない
　T を出力する
forever

アルゴリズム (1)

T が強過ぎるとは反例を含意することである。**T** が弱過ぎるとは正例を含意できないことである。次の二つの方法を与えたとき、アルゴリズム (1) は有効となる。一つは強過ぎる **T** を弱くする方法であり、もう一つは弱過ぎる **T** を強くする方法である。第 5 節でこれらの二つの方法について詳しく説明する。

3 制約付き再帰的図形の紹介

本節では学習の対象とする制約付き再帰的図形について紹介する。まず、制約の概念について述べよう。制約を厳密に定義する言葉はないが、一般的に言えば、制約とは、システムに対して成り立たなければならない関係を表す。制約を基礎とする制約プログラミング言語はいくつかの論文において提案されており、中でも、制約のコンピュータ・グラフィックスに対する応用に関する論文が数多く発表されている。特に、論文 [12] では、Juno という名前の graphics editor について述べられている。このシステムは、五種類の制約を使って、図形を定義するものである。これらの制約は $CONG, PARA, HOR, VER$ と CC である。以下の部分で、Juno で使われている五種類の制約を用いて、本論文で学習の対象とする制約付き再帰的図形を定義する。

可算個の点 $p, q, r, p_1, q_6, q_8, r_{13}, \ldots$ が存在することを仮定する。これらの点は平面上の点定数である。図形は線分とドットの multi-set によって定義される。$line(p_1, p_2)$ とは点 p_1 と点 p_2 との間に存在する線分を項の形で表現したものである。$dot(q_1)$ とは点 q_1 のところに存在するドット (黒小丸) を項の形で表現したものである。従って、図 -1 の図形は次のように表現できる。

$$E = \{line(p_1, p_2), line(p_2, p_3), line(p_3, p_4), line(p_4, p_1), dot(p_1)\}$$

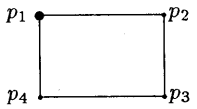

図 -1

図形に対して用いる制約は五種類あり、次のように解釈される。

$cong(p, q, r, n)$: 点 p から点 q までの距離は点 r から点 n までの距離と同等である。
$para(p, q, r, n)$: 点 p から点 q までの方向は点 r から点 n までの方向と平行である。
$hor(p, q)$: 点 p から点 q までの方向は水平である。
$ver(p, q)$: 点 p から点 q までの方向は垂直である。
$cc(p, q, r)$: 点 p から点 q を経て、点 r まで回る方向は時計の回る方向と反対である。

五つの制約を導入したので、制約付き図形を二つ組 $< E, C >$ によって定義することができる。C は図形 E が満たさなければならない制約の集合である。例え

ば、図-1 の制約付き図形は $< E', C' >$ として定義される。

$$E' = E$$
$$C' = \{cong(p_1,p_2,p_4,p_3), para(p_1,p_2,p_4,p_3), hor(p_1,p_2), ver(p_2,p_3)\}$$

図-1 の制約付き図形を $< E'', C'' >$ として定義することもできる。

$$E'' = E$$
$$C'' = \{hor(p_1,p_2), hor(p_4,p_3), ver(p_1,p_4), ver(p_2,p_3)\}$$

本論文で学習の対象とする制約付き再帰的図形は本節で定義された制約付き図形の一部である。この一部は再帰的に定義できるような図形からなる。例えば、次の一連の制約付き図形は下の性質によって再帰的に定義できる。

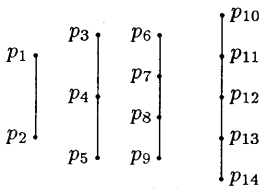

図-2: 制約付き再帰的図形の例

再帰的な性質: 上の図形はいくつかの順番につながっている線分からなる。ただし、これらの線分は互いに長さが等しい。

制約付き再帰的図形は、同じ再帰的な性質によって定義された一連の図形と考えられる。これらを本論文の学習対象とする。

4 学習ためのプログラミング言語

本論文では制約付き再帰的図形を認識するために、制約論理プログラミング言語を用いる。J.Jaffar と J-L.Lassez は一連の制約論理プログラムの基礎を与えるために、$CLP(\mathbf{X})$ という枠組を提案した (論文 [14])。$CLP(\mathbf{X})$ 中の $\mathbf{X}$ に適当なドメインを具体例として与えることによって、様々な特徴を持ったプログラミング言語が得られる。

$CLP(\mathcal{G})$ は、制約付き再帰的図形を認識するために、$CLP(\mathbf{X})$ の枠組の上に構成された制約論理プログラミング言語である。本論文で使う $CLP'(\mathcal{G})$ は $CLP(\mathcal{G})$ に制限と拡張を加えたものである。

4.1 制約論理プログラミング $CLP(\mathcal{G})$ と $CLP'(\mathcal{G})$

many-sorted structure $\mathcal{G}$ には二つの sort と三つの関数がある。二つの sort は点 (*point*) と図形 (*figure*) である。関数は、*line*(2 引数)、*dot*(1 引数)、+(2 引数) であり、次のように定義域と値域が定義される。

$$line: \ point \times point \longrightarrow figure$$
$$dot: \ point \longrightarrow figure$$
$$+: \ figure \times figure \longrightarrow figure$$

また、可算個の点定数があり、各定数は平面上の特定の点に対応している。

以下では、$p, q, r, ...$ で点定数、$P, Q, R, ...$ で点変数、$X, Y, Z, ...$ で図形変数を表すことにする。

関数 *line* は可換性を満たす。

$$line(p,q) = line(q,p)$$

関数 + は可換性と結合性の両方ともを満たす。

$$X + Y = Y + X$$
$$(X + Y) + Z = X + (Y + Z)$$

$\mathcal{G}$ には六つの制約が用意されている。それぞれ、*cong*(4 引数)、*para*(4 引数)、*hor*(2 引数)、*ver*(2 引数)、*cc*(3 引数) と =(2 引数) である。最初の五つについては前節で説明した。これらは図形制約と呼ばれる。図形制約の引数は点に制限されている。= は二つの項が等しいことを意味する。

$CLP(\mathcal{G})$ のプログラムはルールの有限集合である。ルールは下の形をしている。

$$p_0(t_0) :- \ c_1(u_1), ..., c_n(u_n), p_1(t_1), ..., p_m(t_m).$$

ただし、$p_i (0 \le i \le m)$ は $\mathcal{G}$ 中の制約とは異なる述語記号であり、$c_j (0 \le j \le n)$ は $\mathcal{G}$ の制約であり、$t_k (0 \le k \le m)$ は図形の列、$u_l (0 \le l \le n)$ は点の列である。$CLP'(\mathcal{G})$ のプログラムのルールは次の制限を加えたルールである。ルールの述語はいずれも一引数であり、ルールの本体に出現する変数は頭部にも出現する。

$CLP'(\mathcal{G})$ のプログラムのゴールは二つの部分に分かれている。

$$? - \ c'_1, ..., c'_l \ \rightarrow \ c_1, ..., c_n, p_1, ..., p_m$$

前半はゴールの条件部分で、後半にあるのはルールの本体と同じものである (後で詳しく説明をする)。

$CLP'(\mathcal{G})$ プログラムのゴールにはさらに次の制限がある。すなわち、変数はゴールに出現しない。

以降が $CLP'(\mathcal{G})$ の実行解釈系の説明である。導出列 (derivation sequence) は、制約が解決可能であるようなゴールからなる。導出ステップ (derivation step) の例を挙げよう。次のゴールとルールがあれば、

$$? - \ C \ \rightarrow \ c_1(t_1), p(t_2)$$
$$p(t_3) :- \ c_2(t_4), q(t_5), r(t_6)$$

次のゴールを導出できる。

$$? - \ C \ \rightarrow \ c_1(t_1), t_2 = t_3, c_2(t_4), q(t_5), r(t_6)$$

ただし、制約 $c_1(t_1)$ & $t_2 = t_3$ & $c_2(t_4)$ は $\mathcal{G}$ において C の下で解決可能であるとする。最後のゴールが制約だけからなるとき、導出列は成功したという。導出列の最後のゴールが導出できなくなるとき、この導出列は有限的に失敗するという。

以上の説明からわかるように、$CLP'(\mathcal{G})$ の実行解釈系は二つの部分からなる。一つは $\mathcal{G}$ ための制約解決系 (constraint solver)、もう一つは論理プログラミングのゴール還元法 (goal-reduction technique) である。

ここで、$CLP'(\mathcal{G})$ のプログラムの例を挙げよう。

$$arch(line(P,Q) + line(Q,R) + line(R,N)) :- \ hor(Q,R), vchain(line(P,Q)), vchain(line(R,N)).$$
$$arch(X + line(P,Q) + line(Q,R) + line(R,N) + Y) :-$$

$arch(line(P,Q)+line(Q,R)+line(R,N)),$
$vchain(X+line(P,Q)),$
$vchain(Y+line(R,N)).$
$vchain(line(P,Q)):-\ ver(P,Q).$
$vchain(line(P,Q)+line(Q,R)):-\ ver(P,Q),$
$ver(Q,R),cong(P,Q,Q,R).$
$vchain(X+line(P,Q)+line(Q,R)):-$
$ver(Q,R),cong(P,Q,Q,R),$
$vchain(X+line(P,Q)).$

例の意味については後で説明する。

4.2 $CLP'(\mathcal{G})$ の制約解決系

$CLP(\mathcal{G})$ は再帰的図形を認識するためのプログラミング言語である。$CLP(\mathcal{G})$ 上の項の解釈は点あるいは図形であり、関数 $line$ と dot は、それぞれ図形 $<E,C>$ 中の線分 ($line$) とドット (dot) を表し、関数 + は図形上の union operater を表している。五つの図形制約は、集合 C の要素と同じものである。以下では、前節に挙げられた例を説明しながら、$CLP(\mathcal{G})$ のプログラムがどのように使われるかについて述べる。便宜上、$CLP(\mathcal{G})$ プログラムは、いずれも一つの目的述語を持っているとする。これについては $CLP'(\mathcal{G})$ でも同じである。前の例では $arch$ が目的述語である。

4.2.1 $CLP'(\mathcal{G})$ プログラムのゴールの構成

$CLP'(\mathcal{G})$ は再帰的図形のためのプログラミング言語であるから、再帰的図形に対する計算を行わなければならない。例えば、下のような再帰的図形 $<E,C>$ を与える。

$$\begin{aligned} E &= \{line(p_1,p_2),line(p_2,p_3),line(p_3,p_4),\\ &\quad line(p_4,p_5),line(p_5,p_6)\}\\ C &= \{ver(p_1,p_2),ver(p_2,p_3),cong(p_1,p_2,p_2,p_3),\\ &\quad hor(p_3,p_4),ver(p_4,p_5),ver(p_5,p_6),\\ &\quad cong(p_4,p_5,p_5,p_6)\} \end{aligned}$$

この再帰的図形に対して、前の例のプログラムを使うと、ゴールは次のように構成される。

$$?-\ C_0 \rightarrow\ arch(E_0).$$

E_0 は E を $CLP'(\mathcal{G})$ の項の形で表現したものである。

$$\begin{aligned} E_0 &= line(p_1,p_2)+line(p_2,p_3)+line(p_3,p_4)+\\ &\quad line(p_4,p_5)+line(p_5,p_6) \end{aligned}$$

C_0 は $\mathcal{G}$ の図形制約の並びであり、再帰的図形 $<E,C>$ の C の部分の論理積を意味する。C_0 がゴールの条件部分で、$arch(E_0)$ がゴールの本体である。ゴールの解釈は再帰的図形 $<E,C>$ の制約 C が成り立っているという条件の下で、$arch(E_0)$ を実行することである。

$CLP'(\mathcal{G})$ の重要な部分の一つは制約解決系である。$CLP'(\mathcal{G})$ の制約は二つの種類に分けられるので、制約解決系も二つに分けて説明する。

4.2.2 等式制約の解決

導出ステップに現れる等式の制約について考えてみよう。等式の形はいつも下のようである。

$$Term_1 = Term_2$$

左の項はゴールに出現した項で変数がない。右の項はルールに出現した項である。$Term_1$ と $Term_2$ の中には可換性と結合性を満たす関数が存在する。従って、この等式の変数の値を決めることは、パターン・マッチングの問題となる。J.M.Hullot は論文 [16] に完全性を持つ AC マッチング・アルゴリズムを発表した。本論文では、等式制約を解決するために、この AC マッチング・アルゴリズムを用いる。次の例を見てみよう。等式の制約

$$line(p_1,p_2)+line(p_3,p_4) = X+line(P,Q)$$

において、左の項がゴールに出現し、右の項がルールに出現する。マッチング・アルゴリズムを実行すると、次のような四つの substitution が得られる。

$$\{<X,line(p_1,p_2)>,<P,p_3>,<Q,p_4>\}$$
$$\{<X,line(p_1,p_2)>,<P,p_4>,<Q,p_3>\}$$
$$\{<X,line(p_3,p_4)>,<P,p_1>,<Q,p_2>\}$$
$$\{<X,line(p_3,p_4)>,<P,p_2>,<Q,p_1>\}$$

4.2.3 図形制約の解決

図形制約の真偽値を判断するために、もう一度、$CLP'(\mathcal{G})$ のゴールの形式を見てみよう。

$$?-\ C_0 \rightarrow \mathbf{p}(E_0).$$

C_0 がゴールの条件部分で、記号 $\rightarrow$ の右が一般の論理プログラムのゴールに相当する。前に述べた通り、条件 C_0 に再帰的図形の制約部分が代入され、$CLP'(\mathcal{G})$ プログラムの図形制約式の真偽値は、条件 C_0 に依存する。つまり、図形制約式が条件 C_0 の中にあれば真であり、存在しなければ偽である。図形制約式が真である時、この図形制約式をゴールから取り除く。偽である時、一般の論理プログラミングと同じようにバックトラックして、別の解を探す。

しかし、制約式は次のような関係を満足している。

1. 交換性
   $ver(P,Q) \Rightarrow ver(Q,P)$
   $hor(P,Q) \Rightarrow hor(Q,P)$
   $cong(P,Q,R,N) \Rightarrow cong(Q,P,R,N)$
   $cong(P,Q,R,N) \Rightarrow cong(P,Q,N,R)$
   $para(P,Q,R,N) \Rightarrow para(Q,P,R,N)$
   $para(P,Q,R,N) \Rightarrow para(P,Q,N,R)$
2. 同値性
   $cong(P,Q,R,N) \Rightarrow cong(R,N,P,Q)$
   $cong(P,Q,R,N) \wedge cong(R,N,M,L) \Rightarrow$
   $cong(P,Q,M,L)$
   $para(P,Q,R,N) \Rightarrow para(R,N,P,Q)$
   $para(P,Q,R,N) \wedge para(R,N,M,L) \Rightarrow$
   $para(P,Q,M,L)$
3. 制約間の関係
   $ver(P,Q) \wedge ver(R,N) \Rightarrow para(P,Q,R,N)$
   $hor(P,Q) \wedge hor(R,N) \Rightarrow para(P,Q,R,N)$
   $ver(P,Q) \wedge para(P,Q,R,N) \Rightarrow ver(R,N)$
   $hor(P,Q) \wedge para(P,Q,R,N) \Rightarrow hor(R,N)$

以上の関係に対処するために、ここで提案する方法は、上の関係によって生成される制約式をすべて$CLP'(\mathcal{G})$の制約解決系に知らせるものである。すなわち、上のような制約式の間の関係を演繹ルールとして、ゴールの条件部分に代入された再帰的図形の制約式集合の演繹閉包(dudective closure)を求める。制約解決系がゴールにある制約式の真偽を判断するのは、この演繹閉包に依存する。従って、入力されたゴール

$$?- \ C_0 \rightarrow \mathbf{p}(E_0).$$

に対して、まず、ゴールの条件部分C_0の演繹閉包C_cを求めて、次のゴールを前と同じように実行する。

$$?- \ C_c \rightarrow \mathbf{p}(E_0).$$

本論文で使う再帰的図形の例では制約部分が無矛盾であると仮定する。なお、上の演繹ルールには関係の矛盾性が含まれていない。例えば、$ver(p_1,p_2)$と$hor(p_1,p_2)$を同時に満たすことは不可能である。図形の学習の時には、制約間の矛盾性を使うことができる。verとhorとは矛盾するので、水平の線分を学習するために、垂直の線分を反例として与えることができる。

5 学習システム

制約付き再帰的図形を学習する場合に、$CLP'(\mathcal{G})$のプログラムを生成するために、制約付き再帰的図形の正例と反例が必要である。正例とは、求める$CLP'(\mathcal{G})$のプログラムによって解釈できる(実行して真である)例のことをいう。反例は解釈できない(実行して偽である)例である。

5.1 反例と正例によるプログラムの診断

プログラムの診断とはプログラムに存在する誤りを取り除く操作のことをいう。プログラムの誤りを検出するために、いくつかの事実を例として与えて、プログラムをこれらの例に対して実行する。従って、実行中の一つ一つのステップにおいてプログラムの動きの正しさを判断する機構が存在することが必要である。モデル推論では、人間あるいはユーザがプログラムの動きの正しさを判断することを担当する。制約付き再帰的図形の学習でもユーザがプログラムの動きの正しさを判定する。すなわち、診断はユーザと会話する形式をとる。

この部分でよく使われるいくつかの概念があるので、概念の定義から始める。頭部に同じ述語記号を持つルールの集まりをプロシージャという。この述語記号はプロシージャの名前として使われる。プログラムを一般に$\mathbf{P}$とし、プロシージャの名前を$p_1,p_2...p_n$とする。例えば、前節のプログラム例のプロシージャには$arch$と$vchain$の二つがある。プロシージャpが$<p,x,y>$を包含するとは、入力がx、出力がyで、プロシージャpを実行することができることである。

プログラム$\mathbf{P}$(目的述語は$\mathbf{p}$)と、反例$<E_0,C_0>$に対して、次のようなゴールを実行する。

$$?- \ C_0 \rightarrow \mathbf{p}(E_0).$$

もし、真という結果が得られば、プログラム$\mathbf{P}$がこの反例を説明できない。従って、、プログラム$\mathbf{P}$は必ずどこかで間違っていることが分かる。

今、$CLP'(\mathcal{G})$プログラムを実行する時にプロシージャ$\mathbf{q}$が呼び出されたとする。プロシージャ$\mathbf{q}$が呼び出された後、次のような制約式が生成される。

$$Term_1 = Term_2, c_1, c_2, ..., c_k$$

$c_1,c_2,...,c_k$がプロシージャ$\mathbf{q}$の制約部分である。反例に対する実行が成功した場合、プログラムの誤りには二つの可能性がある。一番目はプロシージャの呼び出しによって生成された等式の制約が解ける場合であり、このことをルールのパターンが弱過ぎるという。二番目はプロシージャの呼び出しによって生成された制約がC_cに属しているようになる場合であり、このことをルールの制約部分が弱過ぎるという。両方ともに対して、プロシージャ$\mathbf{q}$に包含される三つ組$<q,x,y>$が正しいプログラムには出現しない。これはユーザの判断である。従って、プログラムの誤りを検出するにはユーザと会話的な形式で診断を行うことが必要である。反例によるプログラムの診断のアルゴリズムは次のように示される。

反例によるプログラムの診断アルゴリズム

入力 反例$<E_0,C_0>$、$CLP'(\mathcal{G})$のプログラム$\mathbf{P}$

アルゴリズム 入力$<E_0,C_0>$でプログラム$\mathbf{P}$の実行をシミュレートする。入力xでプロシージャ$\mathbf{q}$が呼び出されるとき、ユーザに質問をする。「$<\mathbf{q},x,\emptyset>$が正しいプログラムに存在するか」($CLP'(\mathcal{G})$では出力はいつも空である)。もし、「いいえ」という答えがあったならば、続いて、「パターンの弱過ぎか制約の弱過ぎか」をユーザに選ばせる。パターンの弱過ぎであれば、$K=3$、制約の弱過ぎであれば、$K=4$とする。そして、$(<\mathbf{q},x,\emptyset>,K)$を返してシミュレートが終る。そうではなければ(「はい」の答えである)、続いて実行をシミュレートする。

アルゴリズム(2)

アルゴリズム(2)の実行の例を挙げよう。プログラムの例(1)に対して、四番目のルールが間違っているとする。

$$vchain(line(P,Q)+line(R,N)):-\\ ver(P,Q),ver(R,N),cong(P,Q,R,N).$$

制約付け再帰図形の反例は$<E_0,C_0>$である。

$$\begin{aligned} E_0 &= line(p_1,p_2)+line(p_2,p_3)+\\ &\quad line(p_3,p_4)+line(p_4,p_5)+line(p_6,p_7)\\ C_0 &= ver(p_1,p_2),ver(p_2,p_3),cong(p_1,p_2,p_2,p_3),\\ &\quad hor(p_3,p_4),ver(p_4,p_5),ver(p_6,p_7),\\ &\quad cong(p_4,p_5,p_6,p_7) \end{aligned}$$

アルゴリズム(2)を使って不完全のルールを検出する。ユーザに質問する順番を下に示す。

$$query: vchain(line(p_2,p_3))?y.\\ query: vchain(line(p_4,p_5))?y.\\ query: arch(line(p_2,p_3)+line(p_3,p_4)+\\ \quad line(p_4,p_5))?y.\\ query: vchain(line(p_1,p_2)+line(p_2,p_3))?y.\\ query: vchain(line(p_4,p_5)+line(p_6,p_7))?n.$$

正しくないプロシージャ(ルール)は

$$vchain(line(P,Q)+line(R,N)):-\\ ver(P,Q),ver(R,N),cong(P,Q,R,N).$$

である。これは $K=3$ の誤りである。

次に、正例によるプログラムの診断について説明する。プログラム $\mathbf{P}$ に対して、正例 $< E_0, C_0 >$ の上で、次のようなゴールを実行する。

$$?- \ C_0 \ \rightarrow \mathbf{p}(E_0).$$

もし、偽という結果が得られれば、プログラム $\mathbf{P}$ はこの正例を説明することができない。従って、プログラム $\mathbf{P}$ がどこかで間違っていることが分かる。以前に反例に対して行った分析と同様の分析をすると、正例に対してプログラムの実行が失敗する原因は、正しいプログラムに存在する三つ組がプログラム $\mathbf{P}$ に包含されないことである。この原因には大きく分けて二つの場合がある。一つはプロシージャを呼び出したときに生成された等式の制約が解けないことである。これには呼び出されるルールが存在しない場合も含まれる。これを、パターンが強過ぎるという。もう一つはプロシージャを呼び出したときに生成された図形制約式の少なくとも一個がゴールの C_c に属さないことである。この場合はプロシージャのルールの制約部分が強過ぎるという。反例と同じように、三つ組 $< q, x, y >$ が正しいプログラムに存在するかどうかの判断はユーザに任せる。正例によるプログラムの診断においても、ユーザと会話的な形式で行うことが必要である。正例の場合、反例によるプログラムの診断と比べて、ユーザが答えるべき情報が多い。なぜかというと、反例に対して誤りのあるプログラムを実行して成功したときは、誤りを検出できる実行のシミュレーションを自動的に生成することができる。これに対して、正例に対しては、診断アルゴリズムは誤りを探索できるようなシミュレーション経路が分からない。ユーザが実行の計算木を与えないと、誤りを探すことができない。要するに、正しいプログラムの計算木を追跡しながら不完全なプログラムの誤りを調べなければならない。正例によるプログラムの診断アルゴリズムは、次のように示される。

正例によるプログラムの診断アルゴリズム

入力 正例 $< E_0, C_0 >$、$CLP'(\mathcal{G})$ プログラム $\mathbf{P}$(目的述語は $\mathbf{p}$ である)

アルゴリズム 入力 $< E_0, C_0 >$ でプログラム $\mathbf{P}$ の実行をシミュレートする。プログラム $\mathbf{P}$ が $\emptyset$ であれば、$K=0$ と返して終了する。そうではなければ、次に行く。プロシージャ $\mathbf{q}$ が呼び出されたとき、ユーザは、正しいプログラムに存在する三つ組 $< \mathbf{q}, x, \emptyset >$ の x を与える。もし、プロシージャ $\mathbf{q}$ が $< \mathbf{q}, x, \emptyset >$ を包含すれば、次のプロシージャの呼び出しを続いて調べる。そうではなければ、x の上でプロシージャ $\mathbf{q}$ の実行をシミュレートする。プロシージャ $\mathbf{q}$ を呼び出しが失敗したとき、ユーザに質問をし、「パターンの強過ぎか制約の強過ぎか」をユーザに選ばせる。パターンの強過ぎであれば、$K=1$、制約の強過ぎであれば、$K=2$ とする。そして、$(< \mathbf{q}, x, \emptyset >, K)$ を返して終了する。制約が解けると、必ずプロシージャ $\mathbf{q}$ は他のプロシージャを呼び出しているはずである。そのプロシージャの呼び出しを $\mathbf{q}$ と同じように調べる。

アルゴリズム (3)

アルゴリズム (3) の実行の例を挙げよう。プログラムの例 (1) に対する間違ったプログラムには四番目のルールが存在しない。制約付き再帰的図形の正例は $< E_0, C_0 >$ である。

$$\begin{aligned} E_0 &= line(p_1,p_2) + line(p_2,p_3) + \\ &\quad line(p_3,p_4) + line(p_4,p_5) + line(p_5,p_6) \\ C_0 &= ver(p_1,p_2), ver(p_2,p_3), cong(p_1,p_2,p_2,p_3), \\ &\quad hor(p_3,p_4), ver(p_4,p_5), ver(p_5,p_6), \\ &\quad cong(p_4,p_5,p_5,p_6) \end{aligned}$$

この正例の上で、間違ったプログラムが実行すれば、失敗になる。そこで、アルゴリズム (3) を使って不完全なプロシージャを検出する。実行の過程とユーザに与える質問を順番に示す。

$query : arch(line(p_1,p_2) + line(p_2,p_3) + line(p_3,p_4) + line(p_4,p_5) + line(p_5,p_6))$
$call\ rule\ arch(line(P,Q) + line(Q,R) + line(R,N))?y.$
$P,Q,R,N?\ P=p_2, Q=p_3, R=p_4, N=p_5$
$arch(line(p_2,p_3) + line(p_3,p_4) + line(p_4,p_5))$
$success$
$query : arch(line(p_1,p_2) + line(p_2,p_3) + line(p_3,p_4) + line(p_4,p_5) + line(p_5,p_6))$
$call\ rule\ vchain(X + line(P,Q))?y.$
$X,P,Q?\ X=line(p_1,p_2), P=p_2, Q=p_3$
$vchain(line(p_1,p_2) + line(p_2,p_3))\ uncovered$

プロシージャ $vchain$ が不完全である。これは $K=1$ の誤りである。

5.2 洗練操作子 (refinement operator)

アルゴリズム (2) とアルゴリズム (3) によって、プログラムの誤りを発見した後、プログラムを書き直すことが必要である。このために、モデル推論システムにおいて用いられるのは洗練操作子である。洗練操作子とは、簡単にいえば、プログラムに誤りがあるところで適当に何かを書き換えて、プログラムがより正しくなるようにするための操作である。詳しい定義については [2] を参照して欲しい。本論文で用いる洗練操作子には二種類ある。一つは ρ_1 で、次のように定義される。

$\mathbf{M}$ を $CLP'(\mathcal{G})$ のルールとする。$\mathbf{N}$ が $\rho_1(\mathbf{M})$ に属する必要かつ十分な条件は、次の項目のうちの一つが成り立つことである。

1. $\mathbf{M} = \square$、かつ、$\mathbf{N} = a(X)$。ただし、a は述語記号であり、X は図形変数である。
2. $\mathbf{N}$ は $\mathbf{M}$ の中の二つの変数を一致させた後に得られたルールである。
3. $\mathbf{N}$ は $\mathbf{M}$ の中の図形変数 X を項 $line(P,Q)$ あるいは $dot(P)$ と書き換えた後に得られたルールである。ただし、P,Q は $\mathbf{M}$ に出現しない点変数である。
4. $\mathbf{N}$ は $\mathbf{M}$ に一つの制約式を付け加えた後に得られたルールである。ただし、この制約式は集合 $\mathbf{S}$(後ろで説明する) に属する。
5. $\mathbf{N}$ は $\mathbf{M}$ にある一つの制約式が削除された後に得られたルールである。

以上の洗練操作子はモデル推論システムで使われる洗練操作子を参照し、$CLP'(\mathcal{G})$ プログラミング言語の特徴を加味して考えられたもので、非常に単純な洗練操作子である。$CLP'(\mathcal{G})$ は再帰的図形学習のためのプログラミング言語であるから、ルールに再帰性があるはずである。ルールを再帰的にするために、洗練操作子 ρ_1 のみを使うのは十分ではない。そこで、論文 [17] の CIGOL システムで使われた absorption operator と intra-construction operator を参照し、二番目の洗練操作子 ρ_2 を考える。ρ_2 は次のように定義される。

1. $\mathbf{M}_1, \mathbf{M}_2$ が $CLP'(\mathcal{G})$ のルールであるとき、$\mathbf{N}$ が $\rho_2(\{\mathbf{M}_1, \mathbf{M}_2\})$ に属する必要かつ十分な条件は、$\mathbf{N}$ が $\mathbf{M}_1$ と $\mathbf{M}_2$ の上で absorption operator をした後に得られたルールであるという条件である。
2. $\mathbf{M}_i (i \geq 2)$ が $CLP'(\mathcal{G})$ のルールであるとき、$\mathbf{N}$ が $\rho_2(\{\mathbf{M}_i\})(i \geq 2)$ に属する必要かつ十分な条件は、$\mathbf{N}$ が $\{\mathbf{M}_i\}(i \geq 2)$ の上で intra-construction operator をした後に得られたルールであるという条件である。

洗練操作子 ρ_2 で重要なものは absorption operator と intra-construction operator であるから、次にこれらの二つの operator を詳しく説明しよう。

論文 [17] に述べでいるように、refutation tree は図-3 のような $\mathbf{V}$ から構成されている。一つの $\mathbf{V}$ は一つの resolution inference を表している。resolution では C_1 と C_2 を与えると、C が導出される。論文 [17] で紹介された absorption operator は C_1 と C を与えたとき、C_2 を生成する。

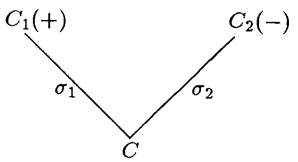

図-3

C は次のように書ける。

$$C = (C_1 - \{L_1\})\sigma_1 \cup (C_2 - \{L_2\})\sigma_2 \qquad (1)$$

論文 [17] では、C_2 を求めることを簡単化するために、$C_1 = \{L_1\}$ と設定する。従って、式 (1) に代数的な処理をほどこすと、C_2 を求める式は次のようになる。

$$C_2 = (C \cup \{\bar{L}_1\}\sigma_1)\sigma_2^{-1} \qquad (2)$$

ただし、σ_2^{-1} は σ_2 の逆 substitution である。論文 [17] では、式 (2) を基礎として absorption を計算する非決定的なアルゴリズムを提案している。このアルゴリズムは $C_1 = \{L_1\}$ が成り立つときに使われているが、$CLP'(\mathcal{G})$ プログラムのルールには制約式が存在するので、$C_1 = L_1 : - U_1, ..., U_k$ という場合が多い。そこで、本論文では論文 [17] の absorption operator を拡張して、$CLP'(\mathcal{G})$ プログラムのルールに対して使えるようにする。まず、$C_1 = L_1 : - U_1, ..., U_k (k \geq 0)$ と設定する。$U_i\ (1 \leq i \leq k)$ はルール C_1 中の制約式である。これを上の式 (2) に代入すると、次のように C_2 を求める式 (3) が得られる。

$$C_2 = ((C - \{U_i\}\sigma_1) \cup \{\bar{L}_1\}\sigma_1)\sigma_2^{-1}\ (1 \leq i \leq k) \quad (3)$$

上の式 (3) で特に $C - \{U_i\}\sigma_1$ に注目して考えよう。$U_i\ (1 \leq i \leq k)$ が k 個の制約式であるから、ルール C にある制約式の数は k より多くなければならない。そして、ルール C から適当に k 個の制約式を選び、U_i と一致するように σ_1 を部分的に決められる。ただし、注意しなければいけないのは、AC マッチング・アルゴリズムを使っているので、逆 substitution が唯一に存在することがいえないことである。原因は関数 $+, line$ と制約 $cong, para, hor, ver$ がいずれも交換性を満たすことである。σ_1 の一部を求めるのは交換性を考えないとうまくいかない。この問題を解決するために、ルールの引数の順番を適当に変えなければならない。C_2 を求める前に、入力とする C と C_1 に対して、関数と制約の引数の順番を適当に変更する。得られた新しい C と C_1 については交換性を認めない。σ_1 の残り部分は論文 [17] のアルゴリズムと同じように決められる。σ_2^{-1} も同じように決められる。具体的なアルゴリズムは次のように示される。

absorption 計算の非決定的なアルゴリズム

入力 ルール C とルール $C_1 = L_1 : - U_1, ..., U_k$

アルゴリズム ルール C と C_1 に対して、適当に引数の順番を変更して、それぞれ同等のルール C' と $C_1' = L_1' : - U_1', ..., U_k'$ に変える。
$\sigma_1' = \sigma_1'' = \emptyset$
$k = 0$ であれば、2. に行く
そうでなければ、1. に行く

1. ルール C' から適当に k 個の異なる制約式を選んで、それぞれ $V_1, ..., V_k$ とする
   σ_1' は次の条件によって決められる
   $$V_i = U_i'\sigma_1'\ (1 \leq i \leq k)$$
   2. へ行く
2. $TP = \{\ (t, p) \mid t$ は $(C' \cup \{\bar{L}_1'\})$ 中の位置 p の項である $\}$
   TP' は TP の部分集合とする
   次のように TP' の分割 Σ を求める
   下の条件を満たすようなブロック $B = \{(r, p_1), ..., (r, p_n)\} \cup \{(s, q_1), ..., (s, q_m)\}$ を Σ の要素とする
   s は r を subsume する、(r, p_j) は C 中の項で、かつ、(s, q_l) は $\{\bar{L}_1'\}$ 中の項である
   上の全てのブロック B に対して、σ_1'' は次の条件によって決められる
   $$r = s\sigma_1''$$
   $\sigma_1 = \sigma_1' \cup \sigma_1''$ とする
   上の全てのブロック B に対して、
   $\sigma_2^{-1} = \cup\{(r, \{p_1, ..., p_n, q_1, ..., q_m\})/v\}$ とする (v は新しい変数である)

出力 $C_1' = L_1' : - U_1', ..., U_k'$,
$C_2 = ((C' - \{U_i'\}\sigma_1) \cup \{\bar{L}_1'\}\sigma_1)\sigma_2^{-1}$

アルゴリズム (4)

次にアルゴリズム (4) の応用例を一個示す。
$C = vchain(line(P, Q) + line(Q, R) + line(R, N)) : - ver(P, Q),$

$ver(Q,R), cong(P,Q,Q,R),$
$ver(R,N), cong(Q,R,R,N).$
$C_1 = vchain(line(P',Q') + line(Q',R')):-$
$ver(P',Q'), ver(Q',R'), cong(P',Q',Q',R').$
$L_1 = vchain(line(P',Q') + line(Q',R'))$、
$U_1 = ver(P',Q')$、 $U_2 = ver(Q',R')$、
$U_3 = cong(P',Q',Q',R')$、 $k = 3$
この例ではルールの引数の順番を変更しない、
$C = C'$,$C_1 = C_1'$ とする
ルール C の制約部分から $V_1 = ver(P,Q)$、
$V_2 = ver(Q,R)$、 $V_3 = cong(P,Q,Q,R)$ を選ぶ
$\sigma_1' = \{P'/P, Q'/Q, R'/R\}$ が決められる
$TP = \{(P, < 1,1,1 >), (P, < 2,1 >), ...\}$ とする
$TP' = \{(line(P,Q), < 1,1,1 >),$
$(line(P',Q'), < 1,1 >)\}$ を選ぶ
すると、 $\sigma_1'' = \{P'/P, Q'/Q\}$ となる
$\sigma_1 = \sigma_1' \cup \sigma_1'' = \{P'/P, Q'/Q, R'/R\}$
$\sigma_2^{-1} = \{(line(P,Q), \{< 1,1,1 >, < 1,1 >\})/X\}$
$C_2 = vchain(X + line(Q,R) + line(R,N)):-$
$ver(R,N), cong(Q,R,R,N),$
$vchain(X + line(Q,R))$

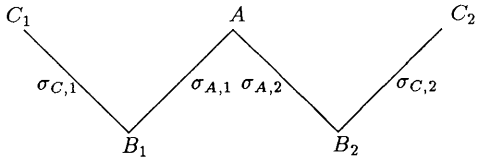

図-4

続いてもう一つの operator を紹介しよう。図-3 の resolution V が二つながって、図-4 の W のような二つの resolution step を構成することができる。図-4 は C_1 と C_2 が A の同じリテラル L 上で resolve して B_1 と B_2 を生成することを表している。論文 [17] の CIGOL では図-4 を利用して B_1 と B_2 を与えた時に節 A, C_1 と C_2 を求めることについて考えていた。リテラル L が A の本体に出現する場合に、この操作は intra-construction といわれる。図-4 の W は二つ以上の V に拡張しても構わない。 B と C が n 個であると仮定して、 $BB = \{B_1, ..., B_n\}, CC = \{C_1, ..., C_n\}$ と書く。式 (1) を使って、式 (4) が得られる。

$$B_i = (A - \{L\})\sigma_{A,i} \cup (C_i - \{L_i\})\sigma_{C,i} \qquad (4)$$

簡単のために、論文 [17] では次のように仮定している。

$$C_i = \{L_i\} \qquad (5)$$

節 B_i から共通の部分 $(A - \{L\})$ を取り出して節 B とする。従って、

$$A = B \cup \{L\} \qquad (6)$$

となる。この式に基づいて、論文 [17] では intra-construction を計算する非決定的なアルゴリズムを提案している。このアルゴリズムも $C_i = \{L_i\}$ が成り立つ時と仮定しているが、 $CLP'(\mathcal{G})$ プログラムで使えるように、 $C_i = L_i :- U_{i1}, ..., U_{ik_i}$ と設定する。 U_{ij} $(1 \le j \le k_i)$ はルール C_i の制約部分である。式 (7) は次のようになる。

$$B_i = (A - \{L\})\sigma_{A,i} \cup (C_i\sigma_{C,i} - \{\bar{L}\}\sigma_{A,i}) \qquad (7)$$

式 (6) を見れば、 B_i が $(A - \{L\})\sigma_{A,i}$ の部分だけを持つのではなく、 $(C_i\sigma_{C,i} - \{\bar{L}\}\sigma_{A,i})$ も持っていることが分る。すると、 B_i の共通の部分 B は B_i の一部になるので、 B を選ぶことは前より難しい。適当に $(A - \{L\})$ らしくなるように B を選ばなくてはならない。次に新しい述語 $p(t_1 + ... + t_r)$ を構成する。 $CLP'(\mathcal{G})$ プログラムの述語の引数は関数 + によって表現される。だから、 p の引数を構成する際に工夫をしないとうまくいかない。まず、 $\sigma_{A,i}$ 中の変数集合から部分集合 $\{v_1, ..., v_m\}$ を適当に選ぶ。そして、 v_1 から v_m を用いて t_i $(1 \le i \le r)$ を次のように構成する。

$r = 0$
1. $line(v_i, v_j)$ が B の頭部に存在すれば、
$r = r + 1$、かつ、 $line(v_i, v_j)$ を t_r とする
2. $dot(v_i)$ が B の頭部に存在すれば、
$r = r + 1$、かつ、 $dot(v_i)$ を t_r とする
3. v_i が B の頭部に + の引数として存在すれば、
$r = r + 1$、かつ、 v_i を t_r とする

$\bar{L}$ は $p(t_1 + ... + t_r)$ とする。論文 [17] のアルゴリズムと同じように、 $\sigma_{C,i}^{-1}$ は空集合とする。 C_i を求める式もできた。

$$C_i = (B_i - B\sigma_{A,i}) \cup \{\bar{L}\}\sigma_{A,i} \qquad (8)$$

式 (6) と (8) を基づいて、intra-construction を計算する新しいアルゴリズムは次のように示される。ただし、ルール BB に引数の順番について、アルゴリズム (4) と同じように適当な変更が必要であるのを注意して欲しい。

intra-construction 計算の非決定的なアルゴリズム

入力 ルール $BB = \{B_1, ..., B_n\}$

アルゴリズム ルール B_i に対して、適当に引数の順番を変更して、同等のルール B_i' に変える
$(1 \le i \le n)$、 $BB' = \{B_1', ..., B_n'\}$ とする
BB' 中のルールからいずれも共通の部分を取り出して B とする、しかも、次の等式を満たす
$\sigma_{A,i}$ は次の条件を満たすようなものから決められる

$$B_i' = B\sigma_{A,i}$$

$\sigma_{A,i}$ 中の変数集合から適当に部分集合 $\{v_1, ..., v_m\}$ を選ぶ
前の方法によって、新しい述語の引数 $t_1 + ... + t_r$ を構成する
新しい述語記号 p を取って、 $\bar{L} = p(t_1 + ... + t_r)$ とする

出力 $BB' = \{B_1', ..., B_n'\}, A = B \cup \{L\},$
$C_i = (B_i' - B\sigma_{A,i}) \cup \{\bar{L}\}\sigma_{A,i}$

アルゴリズム (5)

次にアルゴリズム (5) の応用例を一個示す。
$BB = \{B_1, B_2\}$ とする
$B_1 = varch(line(P,Q) + line(Q,R) +$
$line(R,N)):- hor(P,Q),$
$ver(Q,R), ver(R,N), cong(Q,R,R,N)$
$B_2 = varch(line(P',Q') + line(Q',R')$
$+line(R',N') + line(N',M')):- hor(P',Q'),$
$ver(Q',R'), ver(R',N',), ver(N',M'),$
$cong(Q',R',R',N'), cong(R',N',N',M')$
この例についてはルールの引数の順番を変更しない、

$BB' = BB$ とする
$B = varch(line(P'', Q'') + line(Q'', R'') + X)$:−
$hor(P'', Q'')$ を選ぶ
$\sigma_{A,1} = \{P''/P, Q''/Q, R''/R, X/line(R, N)\}$
$\sigma_{A,2} = \{P''/P', Q''/Q', R''/R',$
$X/line(R', N') + line(N', M')\}$
部分集合 $\{Q'', R'', X\}$ を選んで、
新しい述語の引数を構成する
そして、$t_1 + t_2 = line(Q'', R'') + X$ とする
新しい述語記号を p とすれば、
$\bar{L} = p(line(Q'', R'') + X)$ となる
$A = varch(line(P'', Q'') + line(Q'', R'') + X)$:−
$hor(P'', Q''), p(line(Q'', R'') + X)$
$C_1 = p(line(Q, R) + line(R, N))$:−
$ver(Q, R), ver(R, N), cong(Q, R, R, N)$
$C_2 = p(line(Q', R') + line(R', N')+$
$line(N', M'))$:− $ver(Q', R'),$
$ver(R', N'), ver(N', M'),$
$cong(Q', R', R', N'), cong(R', N', N', M')$

5.3 学習アルゴリズム

この節では、プログラム診断のアルゴリズムと洗練操作子 ρ_1, ρ_2 を使って、制約付き再帰的図形を学習するトップ・レベルのアルゴリズムを紹介する。**P** はプログラム、すなわち、ルールの集合である。**S**(洗練操作子 ρ_1 の 4 番目のところで使われている) は制約式の集合である。**p** は目的述語である。本学習システムのトップ・レベルのアルゴリズムは次のように示される。

学習アルゴリズム

アルゴリズム $\mathbf{P} = \emptyset, \mathbf{S} = \emptyset$
目的述語を **p** とする
repeat
次の例を与える
正例の場合、C の制約式を **S** に入れる
repeat

1. 正例 $< E_0, C >$ の上で **P** の実行が失敗したら、アルゴリズム (3) を使って、プログラム **P** を診断する。番号 K によって洗練操作子 ρ_1 を使って、アルゴリズム (3) が返した三つ組を包含するルールを生成する
2. 反例 $< E_0, C >$ の上で **P** の実行が成功したら、アルゴリズム (2) を使って、プログラム **P** を診断する。番号 K によって洗練操作子 ρ_1 を使って、アルゴリズム (2) が返した三つ組を包含しないルールを生成する

until 入力されたすべての例に対して、
プログラム **P** は正しい
洗練操作子 ρ_2 を使って、
プログラム **P** を更新する
生成されたプログラム **P** を出力する
until 入力とする図形の例はなくなる

アルゴリズム (6)

次にこの学習アルゴリズムを使って、実際に制約付き再帰的図形を学習する例を一つ挙げよう。学習したいのは *vchain* と呼ばれる制約付き再帰的図形である。

```
| ?- cgmis.
Object predicate ? vchain.
Next fact ? (< line(p1,p2),{ver(p1,p2)} >,true).
Error: missing solution,creating rule
vchain(X).
Next fact ? (< dot(q1),∅ >,false).
Error: wrong solution,modifying rule
vchain(line(P,Q)).
Next fact ? (< line(q2,q3),{hor(q2,q3)} >,
   false).
Error: wrong solution,modifying rule
vchain(line(P,Q)):- ver(P,Q).
   . . .
vchain(line(P,Q)):- ver(P,Q).
vchain(line(R,N)+line(N,M)):-
   ver(R,N),ver(N,M),cong(R,N,N,M).
   . . .
vchain(line(P,Q)):- ver(P,Q).
vchain(line(R,N)+line(N,M)):-
   ver(R,N),ver(N,M),cong(R,N,N,M).
vchain(line(P1,P2)+line(P2,P3)+line(P3,P4)):-
   ver(P1,P2),ver(P2,P3),ver(P3,P4),
   cong(P1,P2,P2,P3),cong(P2,P3,P3,P4).
Using absorption operator
vchain(line(P,Q)):- ver(P,Q).
vchain(line(R,N)+line(N,M)):-
   ver(R,N),ver(N,M),cong(R,N,N,M).
vchain(line(P1,P2)+line(P2,P3)+X):-
   ver(P1,P2),cong(P1,P2,P2,P3),
vchain(line(P2,P3)+X).
Next fact?n.
```

プログラム診断の例があるので、上の例ではシステムとユーザの会話部分を省略した。

6 将来の展望

本論文で用いた $CLP'(\mathcal{G})$ は $CLP(\mathcal{G})$ に制限を加えたものである。$CLP(\mathcal{G})$ を実現した場合、その中で使う制約解決系は今の制約解決系よりもっと複雑である。この問題を本論文の一つの将来の拡張とする。制約 *cong*,*para*,*hor*,*ver*,*cc* の解決に対しては、これらの制約がゴールの条件部分 C_c に属するかどうかによって判断される。従って、これらの制約の解決は C_c を求めるルールに依存する。今の場合は、演繹閉包 C_c を求めるルールが不完全である。例えば、第三節の図-1 については、制約集合

$$C = \{cong(p_1, p_2, p_3, p_4), para(p_1, p_2, p_3, p_3), hor(p_1, p_2), ver(p_2, p_3)\}$$

によって、長方形が表現できる。次の制約集合

$$C' = \{hor(p_1, p_2), hor(p_4, p_3), ver(p_1, p_4), ver(p_2, p_3)\}$$

によっても、長方形が表現できる。従って、制約 *hor* と *ver* によって制約 *cong* が推論できる。また、制約 *cong* と制約 *para* との間に推論ルールも存在する。これらの

については非常に複雑である。完全な制約解決系を得ることは難しい。この問題を将来のもう一つの拡張とする。三番目の拡張としては、$CLP(\mathcal{G})$ に代数的な計算とほかの制約を加えることである。

謝 辞

日頃に沢山の助言を頂いた中島玲二教授に感謝致します。本論文を完成する際、貴重な示唆、教示を頂いた萩谷昌己助教授に感謝致します。また、お世話になっております中島研究室のほかの方々にも感謝致します。

参考文献

[1] Ehud Y.Shapiro: An Algorithm that Infers Theories from Facts. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence.* August,1981,pp.446-451

[2] Ehud Y.Shapiro: *Algorithmic Program Debugging.*The MIT Press,1982

[3] W.Leler: *Constraint Programming Languages -Their specifiction and Generation.*Addison-Wesley Publishing Company,Inc.1988

[4] Hans P.Zima: A Constraint Language And Its Interpreter.*Comput. Lang.* Vol.11,No.2,pp.65-83,1986

[5] A.Borning: The Programming Language Aspects of ThingLab,a Constraint- Oriented Simulation Laboratory.*ACM Transactions on Programming Languages and Systems.* Vol.3,No.4,October 1981,pp.353-387

[6] A.Borning: Defining Constraints Graphically. University of Washington Computer Science Department Technical Report No.85-09-06,Seattle,Wash.

[7] A.Borning,R.Duisberg and B.Freeman-Benson: Constraint Hierarchies. *OOPSLA'87 Proceedings.*October 4-8,1987,pp.48-60

[8] J.V.W.Christopher: A High-Level Language for specifyind.*ACM Transactions on Graphics.*Vol.1,No.2,April 1982,pp.163-182

[9] D.Epstein and Wilf R.LaLonde: A Smalltalk Window System Based On Constraints.*OOPSLA'88 Proceedings.*September 25-30,1988,pp.83-94

[10] Pedro A.Szekely and Brad A.Myers: A User Interface Toolkit Based on Graphical Objects and Constraints.*OOPSLA'88 Proceedings.* September 25-30,1988,pp.36-45

[11] M.Stefik: Planning with Constraints(MOLGEN:Part 1).*Artificial Intelligence.*Vol.16,1981,pp.111-140

[12] G.Nelson: Juno,a constraint-based graphics system. In Barsky,B.A.(editor), *SIGGRAPH'85 Conference Proceedings.*pp.235-243. ACM, San Francisco,July,1985.

[13] D.Giuse: KR:Constraint-Based Knowledge Representation.CMU-CS-98-142, April 1989

[14] J.Jaffar and J-L.Lassez: Constraint Logic Programming.*Proc 14th ACM POPL Conf.*Munich,January 1987,pp.111-119

[15] J.Jaffar and S.Michaylov: Methodology and Implementation of a CLP System. Technical Report, Computer Science Dept.,Monash University, June 1986

[16] J.M.Hullot: Associative Commutative Pattern Matching. *Proc. of the 6th International Joint Conference on Artificial Intelligence.* Tokyo,1979,pp.406-412

[17] S.Muggleton: Machine Invertion of First-order Predicates by Inverting Resolution. *Proc. of the 5th International Conference on Machine Learning.*Ann Arbor,June,12-14,1988,pp.339-352

[18] Alan K.Mackworth: Consistency in Networks of Relations.*Artificial Intelligence.*Vol.8,1977,pp.99-118

[19] R.Mohr and Thomas C.Henderson: Arc and Consistency Revisited.*Artificial Intelligence.*Vol.28,1986,pp.225-233

[20] R.Dechter and J.Pearl: Network-Based Heuristics for Constraint-Satisfaction Problems.*Artificial Intelligence.*Vol.34,1988,pp.1-38

[21] R.Dechter and J.Pearl: Tree Clustering for Constraint Networks.*Artificial Intelligence.*Vol.38,1988,pp.353-366

[22] 横井俊夫, 相場亮: 制約ロジック・プログラミング. 情報処理.Vol.30,No.1, Jan.1989,pp.29-38

[23] Mark E.Stickel: A complete unification algorithm for associative-commutative functions.*Proc. of the 4th IJCAI.*Tbilisi,1975