ICOT Technical Memorandom: TM-1225

TM-1225

制約論理プログラミングシステム

相場　亮

October, 1992

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

制約論理プログラミングシステム

相場 亮
(財) 新世代コンピュータ技術開発機構

本稿の目的は，制約論理プログラミングシステムに関して，概観を与えることにある．制約論理プログラミングは，新しいプログラミング・パラダイムとして，近年，注目を集めているものであり，問題のとらえかた，記述方法において，従来のパラダイムとは一線を画すると共に，論理プログラミングという土台の上に展開されていることで，問題解決における新しい展開を期待させるものである．

本稿においては，論理プログラミングの初歩的知識を前提として，制約論理プログラミングについて，実際の処理系を簡単に紹介しながら解説を行う．

1 はじめに

「制約論理プログラミング」とは，「制約」という問題表現・解決のためのパラダイムを導入することによって論理プログラミングを拡張して得られるプログラミング・パラダイムのひとつである．

「制約 (*constraint*)」は拘束条件と訳されることもあるが，歴史的には，制約は画像理解の分野において，局所的な条件，この場合には，一頂点に集まる稜の見え方の組み合わせから，全体の無矛盾な解釈を導き出すという文脈で登場した．すなわち，局所的な条件の組み合わせの中から全体を矛盾なく説明するような局所条件の組を選び出すという，探索問題の形式化の一形態であると考えることができる．このとき，局所条件の許容される組み合わせを与える条件のことを「制約」と呼んでいた．これは，たとえばある局所条件により，ある稜が凹稜であるとされたのであれば，その稜の反対側においても凸稜と解釈されねばならないという条件である．言い換えれば，「制約」とは，この場合には局所条件の間に許される関係のことであった．

ある問題を解決しようとする場合，まずその問題自身を厳密に記述することが必要である．そのためには，問題の領域とその領域において問題を構成する対象，およびこれら対象間の関係を明確にしなければならない．制約とは，このような対象間の関係を宣言的に記述したものである．

このような問題記述の方法は，問題を構成する対象間に成立する関係 - 制約 - を，それらの関係をどのように満たせば良いのかということには触れずに述べる方法である．このような関係のみの表現を許す，あるいはそのような表現が可能であるような問題記述のパラダイムが制約パラダイムであり，これに従ったプログラミング言語を「制約プログラミング言語 (*Constraint Programming Language*)」と呼ぶ．

ここに，関係を記述することで問題を記述する言語が一つある．論理プログラミング言語である．論理プログラミング言語においては，すべては述語という形の関係で記述され，しかもこの述語によって表される関係はユーザが自由に定義できる．さらに，探索ということについて言えば，論理プログラミング言語は，バックトラックに基づく探索によってそのプログラムを実行しており，これらの意味で，論理プログラミングは，このような制約による問題解決を記述し，解くのに適した性質を持っていることが分かる．

2 制約機能と論理プログラミング

論理プログラミング，ここではPrologについて考えてみるが，これに制約機能を組み込むことが，論理プログラミングにとってどのような意味を持つことになるのかについて考えてみる．

Prologは本来，プログラミングの観点から見るとき，問題の性質を記述することによってその問題を解くという点にその特徴があった．たとえば，Prologの教科書にはほとんどといっていいほど載せられてい

るリストの連結のプログラム

```
append([],X,X).
append([H|T],X,[H|Y]) :- append(T,X,Y).
```

を例に考えてみる．このプログラムは，「リストの連結を行うプログラム」ではなく，「ある2つのリストと，それを連結して得られるリストとの関係」を記述したものである．したがって，このプログラムは，次のように評価すべきゴールに応じて，リストの連結にも，あるいはリストの分解にも使うことができる．

```
?- append([a,b],[c],X).

X = [a,b,c] ?

yes
?- append(L1,L2,[a,b,c]).

L1 = [],
L2 = [a,b,c] ? ;

L1 = [a],
L2 = [b,c] ? ;

L1 = [a,b],
L2 = [c] ? ;

L1 = [a,b,c],
L2 = [] ?

yes
```

ところが，同様の手法を用いて，たとえば2数の和と差についての問題を解こうとすると，この考え方は破綻してしまう．たとえば，

```
add(X,Y,Z) :- Z = X+Y.
```

というプログラムに対しては，

```
?- add(2,3,X).
X = 2+3 ?
yes

?- add(X,3,5).
no
```

といった結果になり，思うような解は得られない．ここではもちろん，最初のゴールに対しては X=5 が，次のゴールに対しては X=2 が望む解である．

この原因は，=の用法にある．プログラマは，この=を，数に関する等号と解釈しているのに対して，Prolog のプログラムにおける=は，ユニフィケーション，すなわち項の間の構文的同値性を表す等号として実行される．そのため，2+3 の+が関数記号とみなされ，加算の意味を持たないためにこの式から5は得られない．また 5=X+3 においても+は加算の意味を持たず，左辺は +(X,3) という形の項に，右辺は5というアトムとみなされ，ユニフィケーションに失敗している．

このような問題を解くひとつの方法は，プログラマの考えている領域，この場合には数を，項を使ってコーディングするというものである．

たとえば，よく知られている自然数の 0 と後者関数 (*successor function*) とを用いたコーディング

$$\begin{aligned} 0 &\Leftrightarrow 0 \\ s(0) &\Leftrightarrow 1 \\ s(s(0)) &\Leftrightarrow 2 \\ s(s(s(0))) &\Leftrightarrow 3 \\ &\vdots \end{aligned}$$

による加算の定義，

```
add(0,X,X).
add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

においては，自然数の同値性を項の同値性に置き換えることによって，「ある 2 つの自然数の和がある自然数に等しい」という関係が記述されている．したがって，この関係を満たすような自然数を見つけることが可能である．たとえばゴールの第 2 引数と第 3 引数にコード化された自然数を与えることによって，減算としても利用することが出来る．このようにして，自然数を項を用いてコード化することにより，加算の関係を通常の Prolog のプログラムとして定義することは可能であるが，この方法では効率は良いとは言えず，実際的な方法とも言えない

もうひとつの方法は，通常の Prolog 処理系で利用可能な特別の述語"is" を使うものである．このとき，プログラムは次のようになる．

```
add2(X,Y,Z) :- Z is X+Y.
```

ところが，このプログラムは，X と Y が与えられて，Z の値を求めるときのみ利用可能である．したがって，あらゆる可能性に対処するためには，通常の手続き型言語によるプログラミング同様，組み合わせ的な数の節が必要となるか，あるいは述語 `freeze(X,P)`[1] を用いて次のようなプログラムを書かなければならない．

```
add(X,Y,Z) :-
      freeze(X, freeze(Y, Z is X+Y)),
      freeze(Y, freeze(Z, X is Z-Y)),
      freeze(Z, freeze(X, Y is Z-X)).
```

このような方法のかわりに，自然に加算の関係を表現することが制約論理プログラミング言語のねらいである．

3 制約論理プログラミング言語

Prolog は，1972 年に A. Colmerauer によって提唱されたが，それ以降，Prolog を母体にさまざまな改良や拡張の努力が続けられている．制約論理プログラミングはこのような拡張のひとつである．1980 年には Prolog II が発表され，1987 年には Prolog III [Colmerauer 87] が Colmerauer により発表された．この Prolog III と殆ど同時に，IBM の Jaffar や Lassez 等のグループにより，制約論理プログラミングスキーマ CLP(X) [Jaffar and Lassez 87] が発表された．また，このスキーマに基づいた制約論理プログラミング言語 CLP(R) [Jaffar and Michaylov 87] が発表されている．1988 年には ECRC の M. Dincbas 等のグループにより，CHIP が提唱されている。また，同じ 1988 年には ICOT の制約論理プログラミング言語 CAL が発表された [Aiba et al. 88].

上にも書いたように，論理プログラミングの特徴は，非決定的処理と，強力なユニフィケーション機構にある．このユニフィケーション機構と，それに由来する「パラメータの双方向性」とは，制約の持つ「関係」としての宣言的意味合いとの親和性が高い．これは，「パラメータの双方向性」が，論理型プログラミング言語の持つ「関係の記述」という側面に由来しているためである．

[1] 述語 `freeze(X,P)` は，述語 P の評価を，変数 X の値が確定するまで遅延させるという機能を持つ

Colmerauer は，制約の持つ「宣言性」に着目した．これが Prolog の述語のある意味での拡張であることから，解決技法を利用者に意識させないで，すべての制約を宣言的に書くことのできる制約論理プログラミングを提唱したのである．

3.1 制約論理プログラミング言語によるプログラムの評価

以下に示すのは，J. Cohen が与えた制約論理プログラミング言語のためのメタ・インタープリタである [Cohen 90]．

まず，節 Head :- Body,Constraints を，次のような単位節で表す[2]．

```
clause(Head, Body,Constraints)
```

ただし，ここで Head はリテラルで，Body はリテラルのリストである．

また，単位節は，次によって表す．

```
clause(Head,[ ])
```

このとき，制約論理プログラミング言語のインタープリタは，次のように書くことが出来る．

```
solve_list([ ],C,C).
solve_list([Goal|Goals],C_Old,C_New) :-
          solve_goal(Goal,C_Old,C_Inter),
          solve_list(Goals,C_Inter,C_New).
solve_goal(Goal,C_Old,C_New) :-
          clause(Goal,Body,C_Current),
          c_s(C_Old,C_Current,C_Temp),
          solve_list(Body,C_Temp,C_New).
```

ここで，述語 solve_list の 3 つの引数は，それぞれ次のような意味を持っている．

1. ゴールのリスト
2. その時点における制約の集合 (の標準形)
3. 新しく得られた制約 (の集合) を，2 に付け加えて得られる新しい制約の集合 (の標準形)．

また，述語 c_s が，制約評価系である．

制約評価系は，ある制約の集合の標準形を保持していて，これに新しい制約が付け加わると，充足可能性を検査し，充足可能であれば新しい標準形を求めるという動作を行なう．

たとえば実数上の等式を制約として考えてみる．すると，たとえば新たに得られる制約が，< 定数 >=< 定数 >，あるいは < 変数 >=< 定数 > という形のみであるとするならば，上に述べたような制約評価は非常に容易になる．すなわち，値の同値性のチェックと，変数値の無矛盾性のチェックのみをすれば良いことになる．

ところが，一般にはこのような単純な制約のみを制約評価系に与えるとは限らない．したがって，与えられた制約に何等かの操作を施して，より単純な形へと変形，あるいは変換することが考えられる．たとえば，線形連立方程式を消去法で解くなどというのが，これにあたる．ここでは，このような単純な形への変形 / 変換のことを，簡約と呼ぶことにする．

このような簡約を行なうためには，そのためのアルゴリズムが存在して，それが制約評価系の中に実装されている必要がある．もし，この簡約アルゴリズムが扱えるような制約と，ユーザに記述を許している制約とが一致していれば，ユーザが書いた制約を直ちに簡約することが出来る．ところが，ユーザに記述を許している制約の中に，簡約アルゴリズムでは直ちに簡約出来ないものが含まれている場合には，この

[2] 制約は節毎にまとめて記述されるものとする．

簡約アルゴリズムをそのまま適用することが出来ない．このような制約は，ユニフィケーションや，他の制約評価の結果によって情報を得，その結果，簡約アルゴリズムを適用できるようになるまで待つのが一般的である．たとえば単純な消去法をしか簡約評価機構として持たない場合，非線形方程式制約を扱うことは出来ない．このような場合には，制約中のある変数が具体的な値を持つのを待って，その結果，制約が線形になれば，消去法で扱うことが出来るようになる．これを制御するのを「遅延評価機構」と呼ぶことにする．

Prolog II などにおいて見ることの出来る不等号制約や非等号制約は，制約中のすべての変数が具体的な値を持つようになってから評価される．これは制約評価のほとんどが遅延評価機構によるものであるとして位置付けられる．このような制約の扱いのことを「受動的な制約処理」と呼ぶ．このような受動的制約においては，それぞれの制約は個別に解かれるため，複数の制約にまたがるような関係を扱うことができない．一方，たとえば同一変数に対して2つ以上の制約が存在する場合，これらを組み合わせて新しい制約とすることができるような能力を簡約評価機構が持っている場合，これを「能動的な制約処理」と言う．たとえば，消去法による制約評価は，「連立」方程式のためのものであって，能動的なものである．

また，簡約アルゴリズムは「漸増性 (incrementability) [Sakai 89]」という性質を持っていることが望ましい．「漸増性」とは，次のことを言う．

> 今，制約 C_1，C_2，C_3 がこの順に得られたとする．C_1 を標準形 S_1 に変換するのに要する計算時間を t_1，C_1, C_2 を標準形 S_2 に変換するのに要する計算時間を t_2，さらに C_1, C_2, C_3 を標準形 S_3 に変換するのに要する計算時間を t_3 とする．このとき，厳密な定義ではないが，漸増性とは，S_1 と C_2 とから S_2 が計算時間 u_1 で，また S_2 と C_3 とから S_3 が計算時間 u_3 で求められ，t_2 と $t_1 + u_2$，および t_3 と $t_2 + u_3$ との間に大きな差がないことを言う．

たとえば，線形方程式を制約として記述できるとき，次のような制約が順に与えられるとする．これらが上の C_1，C_2，C_3 に対応する．ここで，標準形を，直観的に「最も単純な形」と考えてみる．最も単純な形とは，「変数 = 値」であるとする．

1. $X = 2$
2. $X + Y = 3$
3. $X + Y + Z = 5$

制約 $X = 2$ が入力されると，これは標準形なので，この制約はそのまま保持される．したがって，t_1 は代入0回である．ついで制約 $X + Y = 3$ が入力された時点で，$X + Y = 3$ は，保持されている制約 $X = 2$ を用いて，$Y = 1$ という標準形へと代入1回で変換される．すなわち，u_1 は代入1回である．この時点で，保持されている制約の標準形は $X = 2$ と $Y = 1$ になる．さらに次の制約 $X + Y + Z = 5$ が入力されると，保持されている制約の標準形を用いて，代入2回で標準形 $Z = 2$ が得られる．これにより，u_3 が代入2回となる．

一方，t_1 は代入0回，t_2 は代入1回，t_3 は代入3回となる．すると，$t_2 = t_1 + u_1$ であり，また $t_3 = t_2 + u_3$ である．よって，上より，このような制約評価系は漸増的であるということが出来る．

すなわち，漸増性とは，より直観的に言い換えると，ある制約の標準形にあらたに制約を付け加えた場合，これまでに得られている標準形については再計算の必要がないことをいう．

3.2 制約論理プログラミング言語の実例

この節においては，制約論理プログラミング言語の実例として，Prolog III，CLP($\Re$)，CHIP，および CAL をとりあげ，それぞれについて概説する．

3.2.1 Prolog III

Prolog III [Colmerauer 87] は 1987 年に A. Colmerauer によって提唱された制約論理プログラミング言語である．Prolog III では，次のような制約を扱うことができる．

1. 無限木

ここでいう無限木とは，一種の有向グラフで，Prolog III における基本的な計算領域である．Prolog III においては，この無限木上の等式 ($=, \neq$) と非等式を制約として扱うことができる．

2. 有理数

演算子 +，−，*，/ から構成される線形方程式，線形不等式および線形非等式を連立させて制約として扱うことができる．Prolog III においては，演算に制限が加えられており，たとえば乗算 ($*$) に関しては，その少なくとも 1 つの被演算子は有理数でなければならず，この制限によって，非線形の式は扱わないようになっている．

3. 真偽値

演算子 ∼，&，∨，⊃ から構成される等式および非等式を制約として扱うことができる．

4. リスト

その左側の被演算子が，長さが既知であるようなリストの連結演算子から構成される等式および非等式を制約として扱うことができる．なお，リストと無限木との間の変換が可能である．

次に制約系の例を示す．

有理数上の制約系の例

最初の例は有理数上の線形方程式，不等式の例である [Colmerauer 87]．鳩 (p) と鼠 (r) とで，合わせて 12 の頭と 34 の足を持つとき，それぞれの数を尋ねるようなゴールを Prolog III では次のように書く．

```
{p >= 0, r >= 0, p+r=12, 2p+4r=34}?
```

ただし，Prolog III では乗算の "*" を省略出来る．これに対する答えは次のようになる．

```
{p=7, r=5}
```

リスト上の制約系の例

次は，長さ 10 のリストに対して，リスト <1, 2, 3> を左から連結した結果とリスト <2, 3, 1> を右から連結した結果が等しいものを求めるためのゴールである．

```
{z:10, <1, 2, 3> . z = z . <2, 3, 1>}?
```

これに対する答えは次のようになる．

```
{z = <1, 2, 3, 1, 2, 3, 1, 2, 3, 1>}
```

制約の評価にあたっては，各変数が一意に定まる場合だけでなく，次のように複数の値が制約系を満たす場合がある．

```
{ 1/2 <= x, x <= 3/4}
```

このようなゴールに対してシステムは，スラック変数[3]を入れた形式

```
{X = -S$3 + 3/4, -S$3 + 1/4 >= 0, S$3 >= 0}
```

という解を返す．ここで S$3 がスラック変数である．

さらに，これらの値が制限を全く持たない場合，すなわち任意の値について制約系が満たされるような場合には，制約評価アルゴリズムによって単純化された系は空の制約系となる．空の制約系は { } で表わされる．

Prolog III のプログラム例

[3] シンプレックス法において、不等式を等式にするために導入される正の値をとる変数のこと．

Prolog III における各節は次の形をしている.

$$t_0 \text{ — } t_1, t_2, \ldots, t_n, S;$$

ここで, $t_0, \ldots, t_n$ は項であり, S は制約系である. むろん $t_1, \ldots, t_n$ が存在しなかったり, S がない (その場合には空の制約系とみなされる) 場合もある.

次に示す例は, 前菜, メインディッシュ, デザートからなる食事で, それぞれにいくつかの選択枝が与えられており, 合計のカロリーがある値を下回るような組み合わせを求めるようなプログラムである [Colmerauer 8

```
LightMeal(a,m,d) —
    Apptizer(a,i), Main(m,j),
    Dessert(d,k),
    {i >= 0, j >= 0, k >= 0, i+j+k <= 10};
Main(m,i) — Meat(m,i);
    Main(m,i) — Fish(m,i);
Appetizer(radishes,1) —;
    Appetizer(salad,6) —;
Meat(beef,5) —;
Meat(pork,7) —;
Fish(sole,2) —;
Fish(tuna,4) —;
Dessert(fruit,2) —;
Dessert(icecream,6) —;
```

Prolog III においては, Prolog, Prolog II と続く, いわゆるマルセイユ系の Prolog の流れをくんで, アルファベットの大文字, 小文字の区別は意味を持たず, 変数はアルファベット 1 文字か, あるいはアルファベットを先頭に持つ文字列で表わされる. また, 与えられたゴール列を満す解を次々に表示するようになっている.

したがって, ゴール `LightMeal(a,m,d)?`
を行なうと, これを満す次のような組み合わせ結果が得られる.

```
{a=radishes, m=beef, d=fruit}
{a=radishes, m=pork, d=fruit}
{a=radishes, m=sole, d=fruit}
    ⋮
```

3.2.2 CLP(ℜ)

CLP(ℜ) [Jaffar and Lazzez 86] は, CLP(**X**) という, 制約論理プログラミングのスキーマのインスタンスとして開発された言語である. CLP(**X**) の特徴は, この種の言語として多ソート一階論理に基づき, 論理的枠組みの中で初めて意味論を与えたものであることと, 充足完全性, 解コンパクト性といった, この言語スキーマに適合するための条件を明らかにした点である. CLP(ℜ) の研究は, J-L. Lassez, J. Jaffar, M. Maher 等 IBM Thomas J. Watson Research Center のグループと, オーストラリアの Monash 大学のグループとの共同で行なわれてきた.

CLP(ℜ) で扱うことの出来る制約は浮動小数点数上の線形方程式, および線形不等式系である. これらのうち, 等式は消去法を用いて解かれ, 不等式はシンプレックス法を用いて解かれる. このような制約評価アルゴリズムによって, CLP(ℜ) の処理系は制約系が矛盾することがなければ解と `yes` を, 矛盾すれば `no` を表示する. また, 制約の中に最終的に非線形のままであるようなものが含まれる場合には制約系と `maybe` が表示される.

CLP(ℜ) においても, Prolog III と同様, 変数の値を確定するのに充分な制約が無い場合には, 変数間の関係が出力される. また, 制約の集合を表示するなどのいくつかのメタ機能を持っている.

CLP(X) において，Jaffar と Lassez は，制約の領域における理論とモデルに関する以下の 2 つの性質を提唱した．ただし，この理論とモデルとにおいて，制約の充足可能性は，「ある制約が理論 T において充足可能であるのは，その制約がモデル $\mathcal{M}$ において充足可能であるとき，かつそのときに限る」という性質を持つ．

1. 理論 T が充足完全である．

   任意の制約について，その制約が充足可能であるか，あるいはそうではないかが証明可能であること．

2. モデル $\mathcal{M}$ が解コンパクトである．

   制約の領域 D の任意の要素は，制約の (無限集合を含む) 集合によって表されること．また，任意の制約について，その否定が制約の (無限集合を含む) 集合によって表されること．

ここで注意しなければならないことは，我々の持つ制約記述のための言語が否定を表す記号を含めて充足完全であるならば，この解コンパクト性は必ずしも要請されないということである．すなわち，解コンパクト性の後半の条件は，いわゆる「失敗としての否定」を導入する際に必要となるものである．このような場合，CLP プログラムの失敗をもって与えたゴールが否定されたと解釈することが可能となる．

これらを満たすような制約の領域については，論理プログラミングの持っていた以下の性質が成り立つというのが，彼らの重要な結果である．

1. プログラムの論理的意味と操作的意味は等しい
2. プログラム P とグラウンドであるようなゴール Q に対して，Q が P の最小不動点集合 $T \mid \omega$ に属し，かつ $T \mid \omega$ が Q 以外の要素を持たないとき，かつそのときに限り Q は P の論理的帰結となる．
3. プログラムの実行が失敗した場合，ゴールが否定された．

これらの結果は，プログラムの完備化により，否定的結果を返すようなプログラムとゴールについても適用可能となる．

3.2.3 CLP(ℜ) のプログラム例

この節においては，CLP(ℜ) のプログラム例をいくつか示す．

CLP(ℜ) の各節は次の形をしている．

$$t_0 : -t_1, t_2, \ldots, t_n$$

ただし，ここで $t_1, t_2, \ldots, t_n$ は，それぞれ項，あるいは制約である．

次に示すのは CLP(ℜ) で記述した複素数の積に関するプログラムである．

```
zmul(c(R1,I1), c(R2,I2), c(R3,I3)) :-
    R3 = R1 * R2 - I1 * I2,
    I3 = R1 * I2 + R2 * I1.
```

このプログラムに対して，次のような一連のゴールが評価可能である．

```
?- zmul(c(1,1), c(2,2), Z).
?- zmul(c(1,1), Y, c(0,4)).
?- zmul(X, c(2,2), c(0,4)).
```

最初のゴールの評価は，直観的にも明らかなように，本体中の等式の右辺が計算され，左辺とユニファイされる．一方，次の 2 つのゴールを評価するには，複素数の除算が必要である．しかし，いずれの場合もユニークな解が得られ，したがって，ゴールの評価は遅延されることはない．一方，次のようなゴールについて考えてみる．

```
?- zmult(c(X,Y), c(X,Y), c(-3,4)).
```

このゴールは，次のような制約系に展開される．

```
X*Y - Y*Y = -3,
2*X*Y = 4.
```

これらの制約は非線形であるので，その評価は遅延させられる．もし．これらの変数が具体化されるようなことがあると，その評価は再開されることになる．もし具体化されなければ，maybe となる．

また，次に示す例は，電気回路の問題をプログラムしたものである．次の節は抵抗における電圧と電流の間の局所的な関係を示す節である．

```
resistor(V, I, R) :- V = I*R.
```

この節を用いて，２つの抵抗を並列，および直列に接続した場合の回路の状態は，次の節によって示される．このとき，上の局所的な性質と，共有変数とを用いて，大域的な性質が表現される．このような方法は，多くの制約論理プログラミング言語で用いることの出来る技法である．

```
par_circuit(V, I, R1, R2) :-
    I1 + I2 = I,
    resistor(V, I1, R1),
    resistor(V, I2, R2).
ser_circuit(V, I, R1, R2) :-
    V1 + V2 = V,
    resistor(V1, I, R1),
    resistor(V2, I, R2).
```

ここでVは，この回路によって生じる電圧降下である．このような方法を用いて，さらに複雑な回路を記述することが可能である．

```
par_series(V, I, R1, R2, R3, R4) :-
    V1 + V2 = V,
    par_circuit(V1, I, R1, R2),
    par_circuit(V2, I, R3, R4).
```

この節に対して，

```
?- par_series(50, I, 10, 10, 10, 10).
```

なるゴールを評価すると，解I = 5が得られる．

3.2.4 CHIP

CHIP[Dincbas et al. 88] は ECRC (European Computer-Industry Research Centre) において M. Dincbas 等によって開発された言語である．その目的は制約付き探索問題を解くことにある．CHIP の特徴は，前に紹介した２つの言語と異なり，有限領域をも計算領域に取り込んだ点にある．

CHIP で扱うことの出来る制約は次の３種類である．

1. 有理数上の線形方程式，および線形不等式系
2. ブール値に関する関係式系
3. 値域が有限領域に限定された変数を含むような関係式系

CLP(℟) とは異なり，CHIP において算術制約の値域として Prolog III 同様，有理数を用いている．この理由は，浮動小数点演算を行なうことに伴なう計算誤差の問題を排除するためである [Dincbas et al. 88]．

CHIP においては，有理数上の線形方程式，および線形不等式系はシンプレックス法によって解かれる．また，ブール値上の関係式系は，ブーリアン・ユニフィケーション・アルゴリズムによって解かれる．

値域が有限領域に限定された変数を含むような関係式系について以下に述べる．

有限領域上の制約としては，算術的制約，記号的制約，利用者定義制約，および高階制約の４種類がある．

1. 算術的制約としては，領域変数上の算術項に関する通常の関係を制約として記述，評価することができる．たとえばX，Yを項とするとき，X>Y，X>=Y，X<=Y，X<Y，X=Y，X~=Yはいずれも算術制約である．

2. 記号的制約の代表的なものは，varがListのNb番目の要素であるときに成立するelement(Nb,List,Var)，リストListのすべての要素が互いに異なるとき，成立するalldifferent(List)の2つである．

3. 利用者定義制約は，利用者が定義した述語を，下で述べる「無矛盾性の技法」を用いて評価するような制約として定義することが出来る．

4. 高階制約としては，ある式をある評価関数に対して最大化，あるいは最小化するような解を求めるような制約(minimize(Goal，Function)のことである．

さて，有限領域上の制約は，「無矛盾性の技法」を用いて評価される．これは，フォワード・チェッキングとかルック・アヘッドと呼ばれる技法である．たとえば，次のような制約が与えられたとする [Dincbas et al. 88]

```
R + E + 1 = 10 + T
```

ここでR $\in \{0, n1\}$ であり，かつE,T $\in \{0, 2, 3, 4, 5, 6, 7, 8, 9\}$ であるとすると，T = 0 およびE $\in \{8, 9\}$ が得られる．

この他，CHIPにおいては，遅延宣言，局所伝播，条件付き伝播などをデモンを用いて実現している．

3.2.5 CHIPのプログラム例

この節においては，CHIPのプログラム例をいくつか示す．CHIPの節はCLP(Ψ)同様，Prologと同様の構文を持っており，制約は本体中の任意の位置に置くことができる．したがって，CHIPの各節は次の形をしている．

$$t_0 :- t_1, t_2, \ldots, t_n$$

ここで$t_1, t_2, \ldots, t_n$は，それぞれ項または制約である．

CHIPにおいてはユニフィケーションを行なう際に，通常の構文的ユニフィケーションとブーリアン・ユニフィケーションの両方を用いる．したがって，システムに対して，どの引数にブーリアン・ユニフィケーションを用いれば良いのかを宣言してやる必要がある．このためにdeclareを用いる．例として次のandを見てみる．

```
?- declare and(h,h,bool,bool,bool).
   and(M,N,X,Y,X\&Y).
```

この例では，最初の2つの引数については通常のユニフィケーションを用い，次の3つの引数についてはブーリアン・ユニフィケーションを用いるということが，declareを用いて宣言されている．また，この節では最後の引数が第3引数と第4引数の論理積になっていなければならないということが述べられている．

ブーリアン・ユニフィケーションを行なった場合には，通常のユニフィケーションにおいて構文的な等式が得られ，それを満たすような最汎ユニファイア(mgu)が求められるのと同じように，ブール等式が得られ，制約評価によってその解が求められる．

次に示すのは，xorゲートの検証の例である [Dincbas et al. 88].

```
?- declare eq(bool,bool).
eq(X,X).

?- declare n_switch(bool,bool,bool).
n_switch(Drain,Gate,Source) :-
    eq(Drain&Gate,Gate&Source).
```

```
?- declare p_switch(bool,bool,bool).
p_switch(Drain,Gate,Source) :-
    eq(Drain&not(Gate),not(Gate)&Source).

?- declare xor(bool,bool,bool).
xor(A,B,X) :-
    p_switch(1,A,T1),
    n_switch(0,A,T1),
    p_switch(B,A,X),
    n_switch(B,T1,X),
    p_switch(A,B,X),
    n_switch(T1,B,X).
```

この回路の出力を記号的に計算する場合には，各スイッチ毎に与えられたブール等式を解く必要がある．その結果，X と T1 とに，あるブール項が束縛される．各ステップ毎の X と T1 の値を示す．下線で始まる変数はブーリアン・ユニフィケーションのアルゴリズムによって導入された変数である．

```
?- xor(a,b,X).

1) T1 = 1 # a # _A&a
2) T1 = 1 # a
3) X = b # _C&a # a&b
4) X = b # _C&a # a&b
5) X = a # b # _D&a&b
6) X = a # b
X = a # b
```

計算が終了すると，値 X = a # b が得られる．

3.2.6 CAL

CAL(*Contrainte Avec Logique*) は ICOT (Institute for New Generation Computer Technology) において開発された言語である [Aiba et al. 88, Sakai and Aiba 89].

CAL では，次の 4 種類の制約を扱うことが出来る．

1. 複素数上の線形・非線形代数方程式 (代数制約)
2. 真偽値上の方程式 (ブール制約)
3. 実数上の線形方程式・不等式 (線形制約)
4. 集合とその要素に関する関係式 (集合制約)

代数制約は Buchberger アルゴリズムによって評価され，Gröbner 基底が制約の標準形として求められる．また，ブール制約のためには 2 種類のアルゴリズムが用意されている．ひとつは Buchberger アルゴリズムを元にして，ブール値に対して適用出来るように，ICOT で開発されたアルゴリズムによって評価され，Boolean Gröbner 基底が求められる．もうひとつはブーリアン・ユニフィケーションを元にした，インクリメンタル・ブーリアン・エリミネーション・アルゴリズムによるものである．線形制約のためには，CLP(ℜ) や Prolog III と同様，シンプレックス法にもとづいたアルゴリズムを用いて評価を行う．最後の集合制約に関しては，ブール制約のためのアルゴリズムをさらに ICOT で拡張したアルゴリズムを用いている．このような制約評価アルゴリズムによって CAL の処理系は制約系が矛盾することがなければ解制約 (最終的に得られた制約の標準形) と yes を，矛盾すれば no を表示する．

3.2.7 CALのプログラム例

この節においては，CALのプログラム例をいくつか示す．

CALの構文は制約が存在することを除いてPrologのそれとほぼ同じである．CALプログラムには2種類の変数がある．ひとつは論理変数であり，これはPrologの論理変数と同様に大文字で始まる英数字の列で表される．もうひとつは制約変数であり，これは小文字で始まる英数字の列で表される．制約変数は大域変数であり論理変数はそれが出現している節内の局所変数である．制約変数を大域変数とすることによりインクリメンタルなゴールが容易となる．

次に示すのは代数制約を用いたCALの簡単なプログラム例である．このプログラムは三角形に関する3つの既知な関係，すなわち

1. 三角形の面積の公式
2. 直角三角形に関する三平方の定理
3. 任意の三角形が2つの直角三角形に分割出来るという定理(図1参照)

から新しい関係，すなわち三角形の三辺の長さと面積とのヘロンの公式として知られる関係を導き出すものである．

```
:- public triangle/4.

surface_area(H,L,S) :- alg:L*H=2*S.
right(A,B,C) :- alg:A^2+B^2=C^2.
triangle(A,B,C,S) :-
        alg:C=CA+CB,
        right(CA,H,A),
        right(CB,H,B),
        surface_area(H,C,S).
```

最初の節 "surface_area" は上の1に対応し，次の節は三平方の定理を表している．また，最後の節は上の3に対応する．

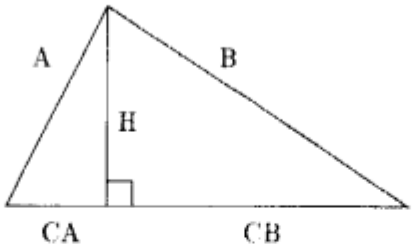

図 1: 任意の三角形は2つの直角三角形に分割可能である

以下のゴールにおいてheronは，このプログラムが定義されているファイル名である．

ゴール

```
?- alg:pre(s,10), heron:triangle(a,b,c,s).
```

は三角形の三辺の長さとその面積との一般的な関係を求めるものである．

このゴール中，alg:pre(s,10)は変数sの優先順位を10にするものである．代数制約評価系においてはブフバーガ・アルゴリズムを用いているため単項間の順序が重要となる．このコマンドはその際の変数の優先順位を変更するためのものであり，各変数は初期状態として優先順位0を持つため，この場合に

は変数 s の優先順位が他の変数よりも上げられたことになる．結果的に，このゴールに対する解は，変数 s に対して解かれる．

このゴールに対してシステムは次を出力する[4]:

```
s^2 = -1/16*b^4+1/8*a^2*b^2-1/16*a^4
      +1/8*c^2*b^2+1/8*c^2*a^2-1/16*c^4.
```

実際この等式は三角形の三辺から面積を求めるヘロンの公式の展開形になっている．

またゴール

```
?- heron:triangle(3,4,5,s).
```

に対しては CAL システムは次を出力する．

```
s^2 = 36}
```

このとき，解の個数が有限個の場合には 1 変数の等式を含むような解が得られる．したがって 1 変数方程式の実根の近似値を求める機能があればそれを次々と適用することによって，すべての解を求めることが出来る．

そのため CAL に 1 変数方程式の実根の近似値を求める機能を付加した．これはまずそれぞれ実根を 1 つしか含まないような区間に分割し，それぞれの区間において実根の近似値を必要な精度で求めるという機能である．

このようにして得た近似値を他の制約を単純化するために利用したいという応用上の要求から我々は変数とその近似値から成る等式を制約として入力できるような枠組を用意した．このためにもともとの制約評価アルゴリズムを近似値を扱うことが出来るように修正している．

次のようなゴールについて考えてみる．

```
?- alg:set_out_mode(float),
   alg:set_error1(1/1000000),
   alg:set_error2(1/100000000),
   heron:triangle(3,4,5,s),
   alg:get_result(eq,1,nonlin,R),
   alg:find(R,S),
   alg:constr(S).
```

これに対して s = -6.000000099 をいう解が得られ，さらにもうひとつの解 s = 6.000000099 がバックトラックによって得られる．

上のゴールの 1 行目，alg:set_out_mode は出力モードを浮動小数点数にセットするためのものである．これをセットしない場合には出力は有理数で行われる．

次の 2 行目の述語 alg:set_error1 はグレブナ基底の計算における係数の近似精度を設定するものであり，3 行目の述語 set_error2 は実根の近似精度を設定するものである．

5 行目の alg:get_result によってグレブナ基底の中から 1 変数 (引数 1 によって指定されている) で非線形 (nonlin によって指定されている) であるような等式 (eq による) が選択される．

ついでそれらの等式はその中の等式の実根の近似値を求めるために述語 alg:find に渡される．得られた実根の近似値はリストの形で変数 S に与えられる．

次に，これらの近似値を用いてグレブナ基底中の他の制約を単純化するために S が述語 alg:constr/1 へと渡され，再び制約として入力される．その結果，2 つの解が得られる．

---

[4] この出力は等式

$$s^2 = -\frac{1}{16}b^4 + \frac{1}{8}a^2b^2 - \frac{1}{16}a^4 + \frac{1}{8}c^2b^2 + \frac{1}{8}c^2a^2 - \frac{1}{16}c^4$$

を表している

4 まとめ

これまで見てきたように，制約論理プログラミングは，「制約」に基づく問題解決に対して，記述の面で大変有用であるが，制約充足の能力には若干の問題がある．それは，処理系に組み込みになっているので，制約評価アルゴリズムを自由に変更したり，問題特有のヒューリスティクスを導入することが困難な点である．

すなわち，実際の問題に即した制約充足問題は極めて複雑な問題となり，これを解くためには，相当の手間を要することになる．したがって，このような制約充足問題を解くようなプログラムを書くにあたっては，様々なヒューリスティクスが必要であるし，それでもなお，充分な効率で解くことが出来ない場合もありうる．

すなわち，現在の制約論理プログラミング言語におけるひとつの大きな課題は，このような言語の枠組の中で，ヒューリスティクスをどのように表し，また扱うかということにあるように思われる．

現状においては，制約評価アルゴリズムを比較的熟知している人であれば，制約の呼びだしの順番などのいわばプログラミング技法を用いた多少の工夫が可能ではある．しかし，これでは「宣言性」を掲げている制約論理プログラミング言語としては，少々問題があるといわざるをえない．

たとえば制約評価系に対する様々なコマンドであるとか，あるいはメタ述語を用いた「制約集合」の認識と，それに基づく制御であるとかの機能をうまく取り込むことによって，ある程度のヒューリスティクスが表現可能なのではないかと考えている．

また，数値データは，多くの制約論理プログラミング言語で扱うものであるが，実際的なプログラミングを前提とすると，この扱い方にも問題はある．たとえば CLP($\mathcal{R}$) では浮動小数点を用い，それ以外の上述の言語においては有理数表現を用いる．有理数表現は一般に浮動小数点を用いるよりも効率が低下するが，浮動小数点には誤差の問題が常につきまとう．制約論理・プログラミングの場合，特に問題になるのが，数値の同値性が制御に大きな影響を与えるということである．また，もうひとつの大きな問題は，制約評価系内部がユーザに対して開放されていないため，最終的に出力された解の誤差解析がこのままでは非常に難しいという点である．

CAL の代数制約評価系におけるある程度の計算誤差を容認するようなバージョンを開発したが，その精度については現在ユーザまかせとなっており，この実装おいて，計算誤差の問題は大きな課題となった．

CAL においては，また，ある時点における制約集合をセーブしたりであるとか，ある制約集合をロードしてくるなどといった機能がある．ある問題をある様々な特定の事例について解くような場合，その問題を制約ロジック・プログラミング言語を用いて記述し，これを一般的な場合について解いておき，これをセーブしておく．さらに特定事例ごとにこれをロードして，その特定事例を表すような制約を付加して解くことによって，この問題を解くということを考えている．これは，特定事例ごとに全く別に問題を解かせる場合と比較して，効率的に有利になると考えている．

このような考え方は，解として得られた集合を一種のプログラムとみなすという立場であり，これは制約論理プログラミング言語の新しい使い方をもたらすものであると考えている．

もうひとつ制約論理プログラミング言語の今後の展開にとって重要なキーワードは「並列」である．制約論理プログラミング言語と並列とのかかわりあいかたとしては，ひとつは制約評価系の並列化が考えられ，もうひとつは並列制約論理プログラミング言語が考えられる．

制約評価系の並列化の目的は，制約評価の効率向上に他ならない．

一方，並列制約論理プログラミング言語の目的は，より柔軟な制御を実現し，さらに並列事象における制約問題解決を目指すことである．これについては，たとえばコミッテッド・チョイスに基づく言語へ制約を導入した ALPS の提案 [Maher 87]，PEPSys へ制約を導入した実験の報告 [Van Hentenryck 89]，並列論理プログラミング言語と制約プログラミング言語の統合を目指した並列制約 (cc) 言語の枠組の提唱 [Saraswat 89] などがある．ICOT では Committed-Choice に基づき，cc の枠組に従う並列制約論理プログラミング言語 GDCC を実装中した．

また，制約論理プログラミング言語の研究で，もうひとつ興味深いテーマとして，制約階層がある [Bor-89]．

これまで述べてきたような制約論理プログラミング言語においては，制約は成立するか否かである．す

なわち，制約の集合に含まれるすべての制約を満たすような解が存在しなければ，全体の計算が失敗に終わり，要求された解は存在しないという結果に終わる．

ところが，問題によっては，制約を満たすにあたって，優先順位がつけられるようなものがある．たとえば，ある制約は必ず成立しなければならないし，別の制約は，出来れば成立してほしいという程度であるとする．これは，そのようにプログラミングを行なえば，制約論理プログラミング言語でも書くことは出来るが，そのためには，その言語の操作モデルがどのようなものであるかを知っていなければならない．

このような問題を制約論理プログラミング言語のときと同様に，宣言的に記述して解くことが，制約階層を導入する目的である．

このような，制約を満たすことに関する優先順位のみならず，制約が満たされない場合には，その「満たさない具合」を最小にするなどということも，この制約階層の考え方によって扱うことが出来る．

このような制約階層を導入した制約論理プログラミング言語を階層制約論理型言語と呼ぶ．これについては，A. Borning 等の HCLP [Borning et al. 89]，ICOT の佐藤等の CHAL [Satoh 90] などがある．

いずれにせよ，制約論理プログラミング言語は生まれてから高々5年程度の言語であり，多くの問題が未解決のまま残されているといっても過言ではあるまい．むしろ，これからの研究の発展が，これらの言語の妥当性を判断するための材料を提供することになるであろう．現状においては(たとえば FORTRAN のような意味での)実用性というよりも，その将来性において期待をかけられている言語であるということが出来ると思うのである．

参考文献

[Aiba et al. 88] A. Aiba, K. Sakai, Y. Sato, D. J. Hawley, and R. Hasegawa. Constraint Logic Programming Language CAL. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, November 1988.

[Borning et al. 89] A. Borning, M. Maher, A. Martindale, and M. Wilson. Constraint Hierarchies and Logic Programming. In *Proceedings of the International Conference on Logic Programming*, 1989.

[Cohen 90] J. Cohen. Constraint logic programming languages. *Communications of the ACM*, 33(7), July 1990.

[Colmerauer 87] A. Colmerauer. Opening the Prolog III Universe: A new generation of Prolog promises some powerful capabilities. *BYTE*, pages 177–182, August 1987.

[Dincbas et al. 88] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The Constraint Logic Programming Language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, November 1988.

[Jaffar and Lazzez 86] J. Jaffar and J-L. Lassez. Constraint Logic Programming. Technical report, IBM Thomas J. Watson Research Center, 1986.

[Jaffar and Lassez 87] J. Jaffar and J-L. Lassez. Constraint Logic Programming. In *4th IEEE Symposium on Logic Programming*, 1987.

[Jaffar and Michaylov 87] J. Jaffar and S. Michaylov. Methodology and Implementation of a CLP System. In *Proceedings of the Fourth International Conferenceon Logic Programming*, pages 196–218, Melbourne, May 1987.

[Maher 87] M. J. Maher. Logic Semantics for a Class of Committed-choice Programs. In *Proceedings of the Fourth International Conference on Logic Programming*, pages 858–876, Melbourne, May 1987.

[Sakai and Aiba 89] K. Sakai and A. Aiba. CAL: A Theoretical Background of Constraint Logic Programming and its Application. *Journal of Symbolic Computation*, 8:589–603, 1989.

[Sakai 89] 坂井 公. 制約解消系. In 溝口 文雄, 古川 康一, and J-L. Lassez, editors, 制約論理プログラミング, 知識情報処理シリーズ 別巻 2, chapter 2. 共立出版株式会社, 1989.

[Saraswat 89] V. Saraswat. *Concurrent Constraint Programming Languages*. PhD thesis, Carnegie-Mellon University, Computer Science Department, January 1989.

[Satoh 90] K. Satoh. Formalizing Soft Constraints by Interpretation Ordering. In *Proceedings of 9th European Conference on Artificial Intelligence*, pages 585–590, 1990.

[Van Hentenryck 89] P. Van Hentenryck. Parallel constraint satisfaction in logic programming: Preliminiary results of chip with pepsys. In *6th International Conference on Logic Programming*, pages 165–180, 1989.