# 制約論理プログラミングシステム
## – CAL, GDCC とその制約評価系 –

相場 亮　長谷川 隆三
(財) 新世代コンピュータ技術開発機構
研究所 第4研究室
108 東京都港区三田 1 丁目 4-28
{aiba, hasegawa}@icot.or.jp

## 概要

本論文においては、ICOT で開発された 2 つの制約論理プログラミング言語 CAL (*Contrainte Avec Logique*) および GDCC (*Guarded Definite Clauses with Constraints*) について述べる。

CAL は代数制約評価系、ブール制約評価系、線形制約評価系を持つ逐次制約論理プログラミング言語であり、一方 GDCC は代数制約評価系、ブール制約評価系、整数線形制約評価系を持つ並列制約論理プログラミング言語である。

代数制約評価系においては制約の評価アルゴリズムとしてブフバーガ・アルゴリズムを採用しているため、解の個数が有限の場合、解制約には 1 変数の高次方程式が含まれる。CAL および GDCC の代数制約評価系においては各変数のとりうる値を求めるため、このような 1 変数方程式からその変数の実根の近似値を求める機能を付加した。この機能を付加した結果、ある変数が 2 つ以上の値をとる可能性が生じ、これを扱うために CAL においてはコンテキストを、また GDCC においてはブロックを導入した。

制約論理プログラミング言語の有用性を示すため、いくつかの応用プログラムを GDCC を用いて試作した。

## 1　はじめに

第五世代コンピュータ (FGCS) プロジェクトは知識処理を指向した並列コンピュータのための新しい技術の研究開発を目的として 1982 年に始められた日本の国家プロジェクトのひとつである。

FGCS プロトタイプシステムはプロトタイプ・ハードウェア、基本ソフトウェア・システム、および知識プログラミングの 3 階層からなり、さらにその上に並列応用システムが構築される。制約論理プログラミングシステムは知識ベース構築モジュールや並列プログラミング環境とともに知識プログラミングモジュールを構成する要素のひとつである。本論文においては ICOT における制約論理プログラミングシステムの研究開発成果の概要について述べる。

制約論理プログラミング (*CLP*) とは、A. Colmerauer [Colmerauer 1987] や J. Jaffar、J-L. Lassez [Jaffar and Lassez 1987] 等によって提唱された論理プログラミングをその計算領域を拡張することによって得られた新しいプログラミング・パラダイムである。Jaffar と Lassez 等は CLP の論理的意味論と操作的意味論とが、論理プログラミングの場合 [van Emden and Kowalski 1976] と同様、互いに一致することを示した。

我々は並列推論マシン上の効率的で強力な問題解決向きの高水準言語を目指して、CLP 言語の研究開発を 1986 年に開始した。

CLP の記述能力は制約を扱うサブシステム、制約評価系に負うところが大きい。これは制約評価系がその CLP 言語で扱うことのできる問題領域を決定するからである。Prolog III [Colmerauer 1987] や CLP(ℜ) のような殆どの既存の CLP 言語は線形方程式および線形不等式を対象とする制約評価系が用意されているが、我々はハンドリング・ロボットのような非線形方程式によって記述される問題を取り扱うため非線形代数方程式のための制約評価系に着目した。そのため我々はブフバーガ・アルゴリズムを制約評価のアルゴリズムとして採用した。

非線形代数方程式以外にも我々はブール制約、集合に関する制約、線形方程式・不等式制約、および階層制約にも着目した。ブール制約のためにはまずブフバーガ・アルゴリズムを、ブール制約を扱うことができるように改良し、ついでブール単一化に基づく新しいアルゴリズムを開発した。集合制約のためにブフバーガ・アルゴリズムに基づくブール制約のためのアルゴリズムをさらに拡張した。また他の CLP 言語と同様に線形方程式・不等式制約のためにシンプレックス法を実現した。この制約評価系を用いて整数線形制約のための制約評価系の開発も行った。さらに我々の枠組の中で階層制約を扱うことを試みた。

我々は 2 つの CLP 言語処理系の研究開発を行った。我々はまず逐次推論マシン PSI 上の逐次制約論理プログラミング言語 CAL (*Contrainte Avec Logique*) の処理系を試作した。ついで CAL 言語処理系を拡張し、さまざまな機

能付加を行い、さらにその結果に基づいて並列制約論理プログラミング言語GDCC (*Guarded Definite Clauses with Constraints*) の処理系の試作を行った。

この論文の構成は以下の通りである。まず第2節においてCLPと制約評価系について簡単に述べる。ついで第3節においてCALについて述べる。さらに第4節ではGDCCについて、また第5節ではさまざまな制約評価系とそれらの並列化について述べる。第6節においては我々の言語を用いた応用プログラムを紹介する。

## 2 CLP と制約評価系の役割

CLPの概念はより容易なプログラミングに対する要求から発生したものである。実際、[Jaffar and Lassez 1987, Sakai and Aiba 1989] においても述べられているように、CLPには次のような重要な特質がある。

- 自然な宣言的意味論を持つ。
- その宣言的意味論と一致する、明白な操作的意味論を持つ。

したがって、このパラダイムを利用することにより、ユーザは宣言的な(それゆえ容易な)プログラミングが可能となる。一方、実行機構の面から見ると、このユーザの与えた宣言的なプログラムと一致するような操作を行うような実行機構が存在する。

たとえばPrologは最も典型的なCLP言語であるが、ユーザはプログラムを"... if ... and ..."のように宣言的に読み書きすることができるとともにそのプログラムをユニフィケーションを基本機構として実行することができる。

ほとんどのCLP言語は互いに良く似たプログラミングのスタイルとPrologにおけるユニフィケーション機構と同様の働きをする制約評価系と呼ばれる機構とを持っている。それゆえCLPプログラムの実行はこの制約評価系に大きく依存している。

通常CLP言語は特定の問題領域を指向しており、その制約評価系はその領域の問題を解くにあたって必要な知識を持っている。Prologの場合にはこの問題とは項間の構文的等式、すなわち単一化である。またCALあるいはGDCCにおいてはこの問題とは

- 代数等式
- ブール等式
- 集合とその要素に関する関係
- 線形等式・不等式

などである。これらの関係のことを、制約と呼ぶ。

CLPのパラダイムにおいては問題をその問題を構成する対象間の制約の形で記述する。したがってCLPには「問題の仕様を与えるだけで、それをどう解くかについては触れる必要がない」という良く知られた利点がある。いいかえるとCLPでは対象間の制約のみを記述すればよく、制約を満たすような対象の値を見つける必要はないということである。

CLPの持つ利点はこれだけではないが、この点はCLPの持つ最も重要な利点のひとつである。方程式をたてることはそれを解くことより容易であるのと同様に対象間の関係を書き下すことはその関係を満たすような対象の値を求める方法を知らなくても可能だからである。

理想的なCLPシステムを想定すると、そのシステムにおいては任意の制約からなる任意の論理式を扱えなければならない。論理プログラミングのパラダイムを利用することによって制約に関するさまざまな論理式を扱うことができるが、しかしなおそれぞれの制約を扱うために強力で柔軟な制約評価系が必要となる。制約評価系の持つ機能の理論的考察についてはCLPの宣言的意味論によって次のような条件が与えられている [Sakai and Aiba 1989]。ただし以下において制約系は各制約の論理積の形で与えられるものとする。

(1) その制約評価系は制約の充足可能性を判定できること。
(2) 充足可能な制約系が与えられたとき、制約評価系はそのすべての解を単純化された形式で表現できること。

Prologの制約評価系、すなわちユニフィケーション・アルゴリズムはこれらの条件を満たしている。またCALやGDCCの制約評価系もそうである。実際にはこれらの制約評価系は次のようなより強い要求をほとんど満たしている。

(3) 制約系が与えられたとき、制約評価系はある意味でその系の最も簡単な表現(その制約系の標準形と呼ぶ)へと変換することができること。

しかしながらこれらの条件は応用面からみた場合にはかならずしも十分なものではなく、たとえば次のような条件を付加することが考えられる。

(4) 充足可能な制約系に対して制約評価系は制約系を満たすような変数値の組を少なくとも1通りは見つけ出すことができる。

このような解を見つけだすことはそれ以前の条件とは独立であり、理論的にはおそらく不可能である。したがって近似値を見つけだすことによりこの条件を部分的に満たすことが必要になる。次節以降においてCALやGDCCのこのような機能について論じる。

制約評価系のもうひとつ重要な性質は「漸増性」という性質である。漸増的な制約評価系は制約を次々に与えられた場合新たに与えられた制約をそれまでに得られている制約系を利用して出来るだけ簡単な形へと変換する。したがって漸増的な制約評価系は「充足不能」である場合、それを出来る限り早い時点でみつけだすことが可能である。またこれによってPrologのようなバックトラックを効率良く行うことも可能となる。幸いCALやGDCCの制約評価系はユニフィケーションと同様、漸増的である。

## 3 CAL - 逐次制約論理プログラミング言語

この節ではCALの構文を中心に述べる。詳細に関してはCALの利用手引き書 [CAL Manual] を参照されたい。

### 3.1 CAL 言語

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問い合わせが容易となる。

以下に代数制約を用いたCALの簡単なプログラム例を示す。このプログラムは三角形に関する3つの既知な関係から新しい関係、三角形の三辺と面積との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" は三角形の高さHと底辺の長さLとからその三角形の面積Sを求める求積公式である。また次の節は直角三角形に対するいわゆるピタゴラスの定理である。最後の節は「任意の三角形は2つの直角三角形に分割可能である」という性質を述べたものである(図1参照)。

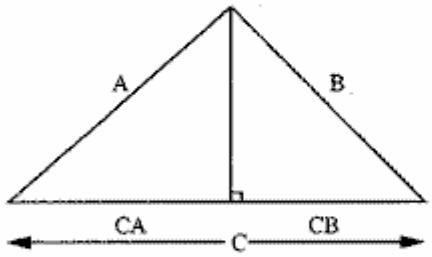


図 1: 任意の三角形は2つの直角三角形に分割可能である

以下の問い合わせにおいて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。

最初の問い合わせ

```
?- 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。

この問い合わせ中、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ッバーガ・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、この問い合わせに対する解は、変数sに対して解かれる。

この問い合わせに対してシステムは次を出力する*:

```
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。

また問い合わせ

```
?- 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、もしも解の個数が有限個であれば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。

次のような問い合わせ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

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。

上の問い合わせの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。

この問い合わせの中心部分は述語 alg:get_result/4 の呼びだしと 述語 alg:find/2 の呼びだしである。5行目の alg:get_result によってグレブナ基底の中からある条件を満たすような等式(群)が選ばれる。この場合には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、選択された等式のリストが変数R に与えられる。

このR 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。

ついで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。

## 3.2 CAL システムの構成

CAL 言語処理系はトランスレータ、推論エンジン、制約評価系の3つからなる。これらのシステムは図2に示すように接続されている。

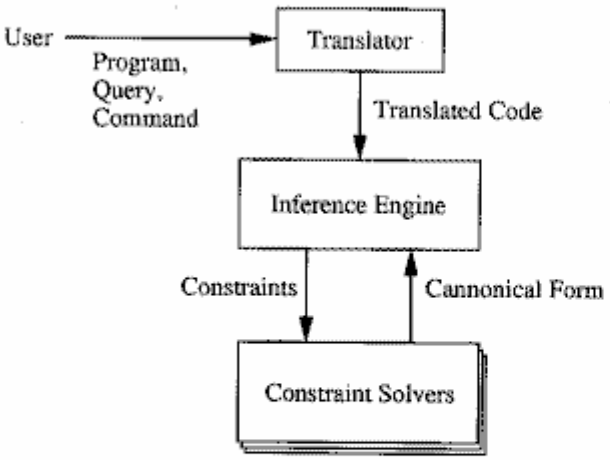


図2: CAL 言語処理系の構成

トランスレータはユーザからの入力を受けとり、それを対応するESP コードへと変換する。これにより、CAL のソースプログラムはESP プログラムへと変換され、推論エンジンによって実行される。またプログラムの実行中に推論エンジンは制約に出会う度に制約の評価を行うために適当な制約評価系を呼び出す。

制約評価系は新たに得られた制約をそれまでの制約の標準形に付加し、新たな標準形を求める。

現在CAL には第1節で述べたような5つの制約評価系が備えられている。

## 3.3 コンテキスト

上述の例のようにある変数が2つ以上の値をとるような状況を扱うために我々はコンテキスト木という概念を導入した。

コンテキストとは制約集合のことである。制約集合が変更される度に新しいコンテキストが生成される。このようなコンテキストを節に持つような木がコンテキスト木である。このコンテキスト木のルートのことをルート・コンテキストと呼び、その時点でユーザの処理の対象となっているコンテキストのことをカレント・コンテキストと呼ぶ。

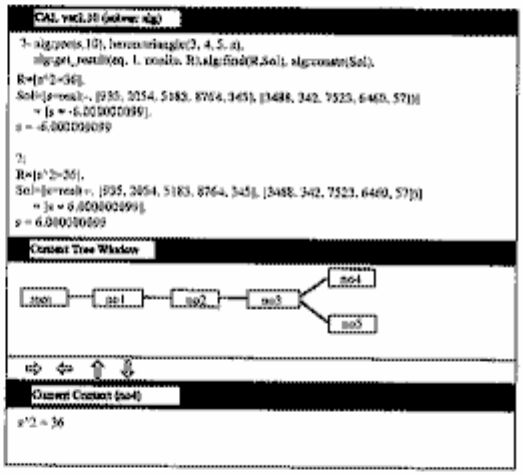


図3: CAL のシステム・ウィンドウ

コンテキスト木は次のようなときに変更される。

1. ゴールの実行:
   新しいコンテキストがカレント・コンテキストの子ノードとして生成される。
2. 別解要求による新しい制約集合の生成:
   新しいコンテキストがカレント・コンテキストの兄弟ノードとして生成される。
3. 優先順位の変更:
   新しいコンテキストがカレント・コンテキストの子ノードとして生成される。

上のすべての場合、新しく生成されたコンテキストが新しいカレント・コンテキストとなる。

CAL処理系にはあるコンテキストの内容の表示、カレント・コンテキストの設定、あるコンテキスト以下のコンテキスト木の部分木の削除といったコンテキスト木を扱うためのコマンドがいくつか用意されている。

図3にCAL処理系ウィンドウの例を示す。

## 4 GDCC - 並列制約論理プログラミング言語

CLPシステムの並列化に関しては次の2つのレベルを考えることができる。ひとつは推論エンジンと制約評価系との並列動作であり、もうひとつは制約評価系の並列化である。CLPシステムの並列化についての研究は、いくつかのものが報告されている。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 1987]、PEPSysへ制約を導入した実験の報告 [Van Hentenryck 1989]、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 1989] などがある。

cc言語のパラダイムでは、計算は図4に示すように「ストア」を中心に複数のエージェントが「問い合わせ」と「アサート」のためのメッセージを交換しながら協調動作を行うような図式にモデル化される。

図4においては *Ask* は問い合わせを、*Tell* はアサートを意味している。

このパラダイムはガード付きリダクション・システム、あるいは条件付きリダクション・システムと考えることができる。リダクションのコントロールは、ガードがストアの内容によって真となる(entailされるという)とそのガードを持つリダクションの規則が発火するといったメカニズムによって実現される。このようにこのパラダイムはKL1 [Ueda and Chikayama 1990] と高い親和性を持っている。

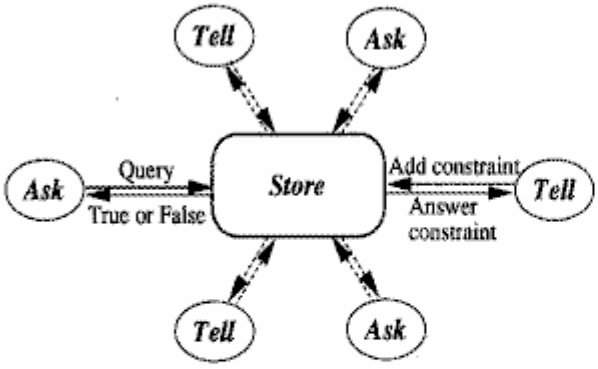


図4: cc言語

GDCC (Guarded Definite Clauses with Constraints) はccに基づいて上で述べた2種類の並列化を行った並列制約論理プログラミング言語であり、KL1を用いてMulti-PSI上に試作された。GDCCではKL1の組み込み述語のほとんどを利用することが可能である。これはKL1の組み込み述語およびユニフィケーションがHERBRANDという特定の領域における制約であると考えられる [Saraswat 1989] からである。

GDCCには制約の標準形を格納するためのデータベースとして「ストア」が概念的に存在する。GDCCシステムは *Ask* あるいは *Tell* を適当な制約評価系へと送る。*Ask* 制約はいわゆる「受動的制約」でありストアの内容を変更することなしに評価される。一方 *Tell* 制約はストアを変更する可能性を持っている。Kl1プログラムのガード部において能動的なユニフィケーションが禁止されているのと同様にGDCCプログラムのガード部には、*Ask* 制約しか書くことが出来ないようになっている。

また、GDCCにはユーザが問題領域に応じて使い分けることができるように複数の制約評価系をプラグインすることが可能になっている。

この節ではGDCCの言語仕様とその計算モデル、システムについて簡単に述べる。さらに詳しい情報については [Terasaki *et al.* 1992] を参照されたい。

### 4.1 GDCC 言語

GDCCの節は次のような構文に従う。

$$Head \text{ :- } Ask \mid Tell, Goal.$$

ここで *Head* は節頭部であり、"|" がコミット・オペレータである。また *Goal* は述語呼びだしの列である。*Ask* はAsk制約とKL1組み込み述語の呼びだしをあらわし、*Tell* はTell制約を表す。

ある節の *Ask* 制約が *true* へと書き換えられるとき、かつそのときに限りその節は *entail* されたという。またある節の *Ask* 制約が *ture* へも *false* へも書き換えられないときこの節は *suspend* されたという。節の本体部、すなわちコミット・オペレータの右側はその節の *Ask* 制約が *entail* されたとき、かつそのときに限り評価される。*Ask* 制約が *true* へと書き換えられる節のことを候補節と呼ぶ。GDCCのプログラムはすべての候補節がrejectされるか、あるいは *Tell* か *Goals* が評価中に失敗したときに失敗する。

次はGDCCで記述したpony_and_manのためのプログラムである。

```
pony_and_man(Heads,Legs,Ponies,Men) :- true |
    alg# Heads= Ponies + Men,
    alg# Legs=  4*Ponies + 2*Men.
```

ここで、trueは常に *true* へと書き換えられるような *Ask* 制約である。本体部のalg#から始まる等式は *Tell* 制約である。ここでalg#はこれらが代数制約評価系において解かれるべき制約であることを示している。また、本体部には *Tell* 制約だけでなくKL1の述語呼びだしも書くことができる。なお、制約論理プログラミングの重要な特性のひとつである「双方向性」はこの制限を加えることによって失われることはない。CALと同様たとえば問い合わせ

```
?- pony_and_man(5,14,Ponies,Men).
```
に対しては Ponies=2、 Men=3 という解が得られ、
```
?- pony_and_man(Heads,Legs,2,3).
```
に対しては Heads=5、 Legs=14 という解が得られる。

## 4.2 GDCC システム

GDCC システムは、コンパイラ、シェル、インタフェースおよび制約評価系からなる。コンパイラでは GDCC ソースプログラムを KL1 コードへと変換する。シェルでは問い合わせの変換と、通常の KL1 のトレーサ、スパイ機能、制約評価系におけるログ情報の収集という簡単なデバッグ環境を提供する。またシェルはインクリメンタルな問い合わせのための環境も提供している。インタフェースは GDCC プログラム (オブジェクトコード) と通信して本体部の制約を制約評価系に送ったり、制約評価系の結果を使ってガード部の制約のチェックを行う。

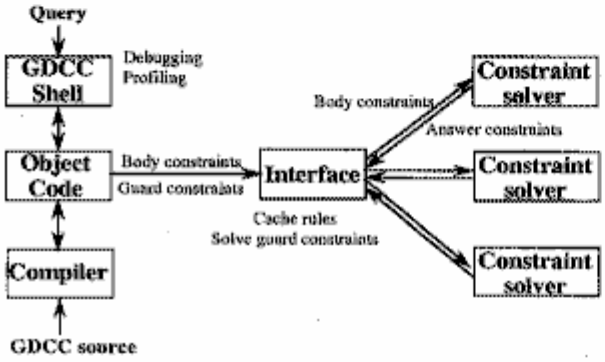


図 5: GDCC システムの構成

GDCC システムの構成を図 5に示す。図中の各要素は並列プロセスであり、特に GDCC プログラムと制約評価系とは並列に実行される。これらの間の同期はプログラム中のガード制約においてのみ取られる。すなわちプログラムの実行は節の選択により行われ、それらの節のガード制約は並列に評価される。もしあるガードが成功すると他のガード評価は中止させられ、選択された節の本体部の評価が開始可能となる。推論エンジンによる本体部のゴールの評価と並行して本体部にある制約は推論エンジンによる評価につれて制約評価系へと送られる。このような協調のスタイルは大変に緩い同期によるものであり、これによって GDCC は逐次制約論理プログラミング言語以上に宣言的なものとなる。

## 4.3 ブロック

GDCC をハンドリングロボット設計のような問題に適用 [Sato and Aiba 1991] するためには次の 2 点について考慮しなければならない。第 1 点は多重環境の取り扱いであり、第 2 点は推論エンジンと制約評価系の同期である。たとえば代数制約評価系から $X^2 = 2$ という解が得られたとき、1 変数の等式から実根の近似値を求める機能があればより詳しく結果を求めることができる。この場合には 2 つの制約集合が得られる。そのひとつには $X = \sqrt{2}$ が含まれ、もうひとつには $X = -\sqrt{2}$ が含まれる。CAL においてはこのような複数の制約集合からシステムはまずひとつの制約集合を選択し、バックトラックによって次々と他の制約集合を求めることができる。しかし GDCC はコミッテッド・チョイスに基づく言語であるため、このような多重環境を扱うためにバックトラックを利用することはできない。一方、同期の問題はたとえば与えられた目的関数の最大値を求めるような制約集合に対するメタな操作を行おうとするときに生じる。メタ操作を行う前に対象となるすべての制約は制約評価系に送られ評価されていなければならない。CAL のような逐次制約論理プログラミング言語においてはこのような操作はプログラム中に書かれた位置によって制御することが可能であるが、GDCC においては節の順番や節中のゴールの順番は実行順序とは無関係になるため同期のポイントを指定する何か他の方法を考えなくてはならない。

これらの問題は制約の局所集合を導入することによって解決することができる。多重環境については制約の局所集合をコンテキストであると考えればよく、推論エンジンと制約評価系の同期の問題も局所集合の評価が終了した時点に設定することで解決することができる。

このため我々は *block* と呼ばれる機構を制約集合のスコープを記述するために導入した。これはあるゴール列をあるブロック中の局所制約集合に対して適用するような機能を持っている。さらにブロック中の失敗を局所化するため PIMOS における荘園機能 [Chikayama *et al.* 1988] を用いている。

# 5 制約評価系とその並列化

この節では CAL と GDCC の制約評価系を簡単に紹介する。まず CAL と GDCC 両方のための代数制約評価系について述べ、次いで 2 つのブール制約評価系について述べる。ひとつはブフバーガ・アルゴリズムを修正したものを利用したものであり、もうひとつはインクリメンタル・ブーリアン・エリミネーション・アルゴリズムを利用したものである。前者は CAL、GDCC の両者で使われており、後者は CAL にのみ使われている。次に GDCC のための整数線形制約評価系について述べ、さらに CAL と GDCC における階層制約評価系について述べる。最後に代数制約評価系において制約の依存性解析を利用して制約評価の効率改善をはかる方法について簡単に述べる。

CAL の制約評価系はすべて ESP で記述されており、一方 GDCC の制約評価系はすべて KL1 で記述されている。

## 5.1 代数制約評価系

代数制約評価系における制約の領域は多変数(非線形)代数方程式である。ブフバーガ・アルゴリズム [Buchberger 1985] はこのような(非線形)代数方程式を解くための方法であり、近年数式処理の分野で広く使われだしている。

ブフバーガ・アルゴリズムの並列化に関しては最近いくつかの試みがなされているが、共有メモリ型のマシンにおける実装 [Vidal 1990, Clarke *et al.* 1990] を除いてはあまりよい結果が得られていない [Ponder 1990, Senechaud 1990, Siegl 1990]。我々はブフバーガ・アルゴリズムを台数効果よりも絶対性能と漸増性の両面においてよい結果が得られることを目標に並列化を行った。我々はいくつかの実装を行い現在もまだなお改良中である。

この節ではブフバーガ・アルゴリズムの逐次版と並列版の両方の実装について概観する。

### 5.1.1 グレブナ基底 と ブフバーガ・アルゴリズム

一般性を失うことなしにすべての等式が $p = 0$ という形をしていると仮定することができる。ここで $E = \{p_1 = 0, \ldots, p_n = 0\}$ を等式の系とする。このとき Buchberger はグレブナ基底の概念を導入し、与えられた等式系からこの基底を求めるアルゴリズムを与えた。このアルゴリズムの概要は以下の通りである(アルゴリズムなどの詳細に関しては、[Buchberger 1985] を参照のこと)。

等式系と単項間のある順序が与えられたとする。このとき等式は最大の単項をそれ以外の単項からなる多項式へと書き換えるような書き換え規則であると考えることができる。たとえば単項間の順序を $Z > X > B > A$ とするとき等式 $Z - X + B = A$ は $Z \rightarrow X - B + A$ という書き換え規則であると考えられる。書き換え規則の対 $L_1 \rightarrow R_1$ と $L_2 \rightarrow R_2$ とに関し、$L_1$ と $L_2$ とが互いに素ではない場合にこの書き換え規則の対を「要対」と呼ぶ。またこれらの規則から生成される「S-多項式」を次で定義する。

$$\text{S-poly}(L_1, L_2) = R_1 \frac{lcm(L_1, L_2)}{L_2} - R_2 \frac{lcm(L_1, L_2)}{L_1}$$

ここで $lcm(L_1, L_2)$ は $L_1$ と $L_2$ の最小公倍数である。

S-多項式が書き換え規則の集合によって0へと書き換えられないとき、この要対を「発散する」といい、その書き換えの結果の等式を等式系に付加する。このような手続きを発散するような要対がなくなるまで繰り返すと合流性のある書き換え規則系が得られる。この合流性のある書き換え規則系、すなわち等式系のことを与えられた等式系のグレブナ基底と呼ぶ。

グレブナ基底において書き換え規則が互いに他を書き換えることが無いようなときこのグレブナ基底は「既約」であるという。既約なグレブナ基底は与えられた等式系の与えられた単項間の順序に関する標準形であると考えることができる。もし等式 $f = 0$ のすべての解が等式系 $E$ の解集合に含まれているならば $f$ は $E$ のグレブナ基底によって0へと書き換えられる。またもし等式系 $E$ が解を持たないならば $E$ のグレブナ基底は "1" を含む。したがってこのアルゴリズムは与えられた等式系の充足可能性を検査するための性質を持っていることになる。

### 5.1.2 並列アルゴリズム

ブフバーガ・アルゴリズムの分散メモリマシンに適合するような粗粒度の並列化としては多項式の集合の並列書き換えが考えられる。しかしブフバーガ・アルゴリズムの計算時間は書き換え規則化される等式の選択の順序に大きく依存する。実際出来るだけ小さな等式を書き換え規則化することが計算効率を上げることになるが、そのため並列実装においても小さな多項式をできるだけ早い時点で書き換え規則化するような選択を行うことが重要である。我々は3つの異なったアーキテクチャに基づいた実装を行った。これらは「パイプライン方式」、「分散方式」、「マスタ・スレーブ方式」の3種類である。ここでは比較的良い結果の得られた、マスタ・スレーブ方式について概説する。

図6にこの方式におけるアーキテクチャを示す。

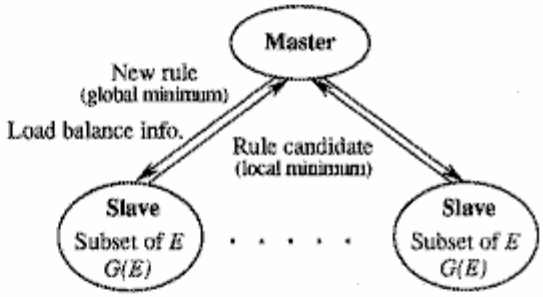


図 6: マスタ・スレーブ方式におけるアーキテクチャ

この方式において多項式集合 $E$ は各スレーブに分割されて保持されている。書き換え規則の初期状態 $G(E)$ はこれとは異なり、すべてのスレーブにおいて同じ書き換え規則集合が使えるようにコピーがすべてのスレーブに置かれている。新しく入力された多項式はマスタによってスレーブに分配される。このようなアーキテクチャのもとで、グレブナ基底はおよそ次のようにして求められる。

各スレーブはマスタからそのスレーブに分配された多項式の最大単項を $G(E)$ を用いて書き換え、そのスレーブにおける最小の最大単項を持つ多項式を選択し、その最大単項をマスタへ書き換え規則化の候補として送る。マスタはすべてのスレーブから書き換え規則化の候補が出揃うのを待って、それらの中から最小のものを選択する。マスタが最小の多項式を判定する際に「最小ではない」候補は、最小である候補よりも早く認識できるので、最小ではない候補を送ってきたスレーブには *non-minimum* なるメッセージを送る。するとこのメッセージを受けとったスレーブは

新しい書き換え規則がマスタから送られて来るまでの間古い書き換え規則の集合を使って多項式をさらに書き換えることができる。一方最小の候補を送ってきたスレーブには、マスタから *minimum* メッセージが送られるが、これを受けとったスレーブはその多項式を書き換え規則へと変換してマスタへ送る。もしこのような方法では最小の候補を決定できない場合にはそれぞれのスレーブで書き換え規則化を行い、それらをマスタへと送りマスタが最小のものを決定する。

表 1 にベンチマーク・テストの結果を示す。これらの問題は [Boege *et al.* 1986, Backelin and Fröberg 1991] からとったものである。なおこの実装の詳細については [Terasaki *et al.* 1992] を参照されたい。

表 1: マスタ・スレーブ方式における実行時間と台数効果

| Problems | Processors | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 16 |
| Katsura-4 (秒) | 8.90 | 7.00 | 5.83 | 6.53 | 9.26 |
| | 1 | 1.27 | 1.53 | 1.36 | 0.96 |
| Katsura-5 (秒) | 86.74 | 57.81 | 39.88 | 31.89 | 36.00 |
| | 1 | 1.50 | 2.18 | 2.72 | 2.41 |
| Cyc.5-roots (秒) | 27.58 | 21.08 | 19.27 | 19.16 | 25.20 |
| | 1 | 1.31 | 1.43 | 1.44 | 1.10 |
| Cyc.6-roots (秒) | 1430.18 | 863.62 | 433.73 | 333.25 | 323.38 |
| | 1 | 1.66 | 3.30 | 4.29 | 4.42 |

## 5.2 ブール制約評価系

ブール制約を解くにはいくつかの方法がある。しかし制約の標準形を求めることができて、漸増性を持ち、さらに解の表現にパラメータを使わずにすむものというと多くはない。これらの基準は第 2 節において述べたようにそのアルゴリズムを制約評価系において利用する際に重要なものとなる。ブール制約評価系として我々は CAL システムにおいてブーリアン・ブフバーガ・アルゴリズム [Sato and Sakai 1988] の試作を行い、これを GDCC システムのために並列化した。このアルゴリズムは上の基準を満たすものである。次いでこれもやはり上の基準を満たすアルゴリズムであるインクリメンタル・ブーリアン・エリミネーション・アルゴリズムを開発し、CAL システムのための試作を行った。

### 5.2.1 Buchberger アルゴリズムによる制約評価系

我々はまずブーリアン・ブフバーガ・アルゴリズムと呼ばれるブフバーガ・アルゴリズムに修正を加えたアルゴリズムに基づき、ブール制約評価系の試作を行った [Sato and Sakai 1988, Aiba *et al.* 1988]。ブフバーガ・アルゴリズムとは異なり、このアルゴリズムは複素数体上ではなくブール環上で動作し、ブーリアン・グレブナ基底と呼ばれるブール制約の標準形を求めるものである。この制約評価系においてはまずアルゴリズムを適用する前に「論理和」($\vee$) や「否定」($\neg$) といったブール演算子を含むような式をブール環の式へと変換する。

我々はこのブーリアン・ブフバーガ・アルゴリズムを KL1 を用いて次のようにして並列化した。まず CAL 上のブーリアン・ブフバーガ・アルゴリズムの動作をいくつかの例題について解析を行い、書き換え規則の集合による式の書き換えを行う部分が並列化を行うに値する大きな部分であることを見出した。またアルゴリズム自身の解析により、アルゴリズム中の並列化可能な部分を見つけ出した。これらの考察の結果、我々は並列実行モデルとしてマスタ・スレーブ方式を採用した。

マスタ・スレーブ方式においては、ひとつのプロセッサがマスタとなりコントローラの働きをし、のこりのプロセッサがスレーブとなり書き換えを担当する。コントローラはブール式を保持し、その時点のグレブナ基底を更新し、S-多項式および自己要対を生成し、さらに等式を各スレーブに分配する。各スレーブはグレブナ基底のコピーを持っており、マスタから送られてくる等式をそのグレブナ基底を用いて書き換える。書き換えの結果が 0 以外の式になった場合にはその結果の式をマスタへと送り返す。等式の分配後にマスタがアイドル状態になるとマスタはその回の書き換えの間スレーブのひとつとして動作する。

6-Queen 問題においては並列化される部分は全体の 77.7% であるので最大 4.48 倍の速度向上がありうるが、実際に 16 プロセッサを利用した場合の台数効果は 2.96 であった。この差はタスク分配のオーバーヘッド、各スレーブにおけるグレブナ基底の更新の手間、分散されたタスクの大きさの差などによるものと考えられる。

ついで我々は実行中に冗長な要対を生成しないような改良を試みた。この改良の結果は改良前と比較して絶対性能は向上したが、台数効果についてはたとえば 16 プロセッサの場合に 2.28 という具合に前よりは低下した。

この並列アルゴリズム、および実験結果の詳細に関しては [Terasaki *et al.* 1992] を参照されたい。

### 5.2.2 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制約評価系

ブーリアン・ユニフィケーションと SL- 導出はブーリアン・ブフバーガ・アルゴリズム以外に良く知られているブール制約の評価アルゴリズムである。ブーリアン・ユニフィケーションは CHIP [Dincbas *et al.* 1988] で用いられているアルゴリズムであり、SL- 導出は Prolog III [Colmerauer 1987] で用いられているアルゴリズムである。ブーリアン・ユニフィケーションはそれ自体効率的な方法であり、binary decision diagram (BDD) をブール式を表すデータ構造として使うことでさらに効率良く実行することができる。しかしこの方法を用いた場合、解には実行中に導入されるパラメータが含まれるため、制約評価系にとって重要な標準形というものが存在しなくなる。これを回避するた

め我々は新しいアルゴリズム「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」を開発した。ブーリアン・ユニフィケーションと同様このアルゴリズムは「ブールの除去」に基づいているが実行中にパラメータを導入しないため、与えられたブール制約集合の標準形が存在するようになっている。

以下においてブール変数を $x, y, z, \ldots$ で、またブール式を $A, B, C, \ldots$ で表すことにする。またブール式を論理積 $(\times)$ と排他的論理和 $(+)$ を用いて表すことにする。たとえばブール式 $F \wedge G$、$F \vee G$、$\neg F$ はそれぞれ $F \times G$、$F \times G + F + G$、$F + 1$ で表すことができる。以降においてブール式 $F$ 中の変数 $x$ のすべての出現にブール式 $G$ を代入して得られるブール式を $F_{x:=G}$ と表すことにする。また混乱のないかぎり記号 $\times$ は書かないものとする。また変数上の全順序を仮定する。

「正規」なブール式を以下のように再帰的に定義する。

1. 2つの定数 0 および 1 は「正規」である。
2. もし正規なブール式 $A$ と $B$ が $x$ よりも小さな変数のみから成っている場合、$Ax + B$ は正規でありこれを $Ax \oplus B$ と表す。このとき $A$ を $x$ の「係数」と呼ぶ。

もし変数 $x$ が式 $F$ 中で最大の変数であるならば、$F$ を $(F_{x:=0} + F_{x:=1})x \oplus F_{x:=0}$ という正規なブール式へと変換することができる。したがってすべてのブール式は正規であると仮定することが可能である。

ブールの除去によればもしあるブール式 $F$ が 0 であるならば $F_{x:=0} \times F_{x:=1}$ $(= G)$ もまた 0 になる。ここで $G$ は $x$ を含んでいないので $G$ の変数の数は $F$ よりも少なくなる。このようにブールの除去によってすこしづつ変数の少ない式を得ることができる。

ブーリアン・ユニフィケーションでは式 $F$ から変数 $x$ を除去した後に $x$ と $(F_{x:=0} + F_{x:=1} + 1)u + F_{x:=0}$ とをユニファイする。ただし $u$ はパラメータである。このユニフィケーションは変数 $x$ に関する新しい制約が与えられたときに、変数 $x$ に式 $(F_{x:=0} + F_{x:=1} + 1)u + F_{x:=0}$ を代入することを意味している。この代入の結果、得られる式は変数 $x$ のかわりにパラメータ $u$ を含むことになる。したがって次は $u$ が他のパラメータを含むような式とユニファイされることになる。

インクリメンタル・ブーリアン・エリミネーション・アルゴリズムにおいては $F = 0$ を $x = (F_{x:=0} + F_{x:=1} + 1)u + F_{x:=0}$ へと変換し $x$ を $(F_{x:=0} + F_{x:=1} + 1)u + F_{x:=0}$ とユニファイするかわりに次のようなリダクションを行う。このことがインクリメンタル・ブーリアン・エリミネーション・アルゴリズムにおいてパラメータを導入せずにすむ理由となっている。

**リダクション**　式 $Cx$ (ただし $C \neq 1$) は以下に示すように式 $Ax \oplus B = 0$ によってリダクションされる。このリダクションは $x$ の係数をもし可能であれば 1 にし、さもなければ可能な限り小さな式にしようとする。

$$
\begin{array}{ll}
Cx \rightarrow x + BC + B & (AC + A + C \equiv 1) \\
Cx \rightarrow (A+1)Cx + BC & (\text{otherwise})
\end{array}
$$

新しいブール制約が与えられた場合には次のマージ操作が実行される。これはインクリメンタル・ブーリアン・エリミネーション・アルゴリズムがユニフィケーションを行わないために必要となる。

**マージ操作**　$Cx \oplus D = 0$ を新しい制約とする。また $Ax \oplus B = 0$ という制約をすでに持っているとする。このときこれらをマージして得られる新しい制約 $(AC + A + C)x \oplus (BD + B + D) = 0$ が新しい解である。もし $ACD + BC + CD + D$ の既約形が 0 でなければ、この式のマージ操作を繰り返し適用する。

この操作はブールの除去の拡張になっている。もし制約を持っていなければ $A$ も $B$ も 0 であると考えられる。この場合にはマージ操作はブールの除去に一致する。

**例**　5つの変数 $a$、$b$、$c$、$d$、$e$ $(a < b < c < d < e)$ のうち、ただひとつのみが 1 である。

$$
\begin{array}{l}
a \wedge b = 0,\ a \wedge c = 0,\ a \wedge d = 0,\ a \wedge e = 0,\ b \wedge c = 0, \\
b \wedge d = 0,\ b \wedge e = 0,\ c \wedge d = 0,\ c \wedge e = 0,\ d \wedge e = 0, \\
a \vee b \vee c \vee d \vee e = 1
\end{array}
$$

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と次のような標準形が得られる。

$$
\begin{aligned}
e &= d + c + b + a + 1 \\
(c + b + a) \times d &= 0 \\
(b + a) \times c &= 0 \\
a \times b &= 0
\end{aligned}
$$

この解は次のように解釈される。この解には $A \times a = B$ という形の式が含まれていないので変数 $a$ は自由である。また $a \times b = 0$ であるのでもし $a = 1$ であるならば変数 $b$ は 0 であり、さもなければ $b$ は自由である。変数 $c$、$d$ に関しても同様であり、最後に $e = d + c + b + a + 1$ であるのでもし $a$、$b$、$c$、$d$ がすべて 0 であるならば変数 $e$ は 1 であり、さもなければ $e$ は 0 である。

インクリメンタル・ブーリアン・エリミネーション・アルゴリズムによって得られた結果に含まれる全ての変数に小さい方から 0 か 1 かを割り当てることによって、与えられた制約を満足するような任意の割り当てを求めることができる。したがって変数に適当な順序を導入することにより制約を満たす好みの変数の割り当てを、容易に得ることができる。

## 5.3 整数線形制約評価系

整数領域における線形制約(これを整数線形制約と呼ぶ)のための制約評価系は有理係数を持った等式と不等式の無矛盾性を判定し、さらにこれらの制約条件のもとである線形の目的関数の最大値、あるいは最小値を求める機能を持つ。本制約評価系の目的は探索プロセスの並列処理を導入することによって整数領域における最適化問題を効率良く解くための道具立てを用意することである。

並列整数線形制約評価系は、その解を整数に限定せず有理数の範囲で求める際に線形制約評価系を用いており、この線形制約評価系自身はシンプレックス法を採用し、KL1を用いて実装されている。

### 5.3.1 整数線形計画法と分枝限定法

以下においては整数線形制約評価系において用いられる並列探索法について述べる。この問題は混合整数線形計画問題、すなわち整数線形制約のもとで与えられた線形関数の最大値あるいは最小値を求める問題である。

すなわち線形制約

$$\sum_{i=1}^{n} a_{ij}\, x_i + \sum_{i=1}^{m} b_{ij}\, y_i \geq e_j,\ \text{for } j = 1, \ldots, l.$$

$$\sum_{i=1}^{n} c_{ij}\, x_i + \sum_{i=1}^{m} d_{ij}\, y_i = f_j,\ \text{for } j = 1, \ldots, k.$$

ただし

$$x_i \in \boldsymbol{R},\ \text{かつ} x_i \geq 0,\ \text{for } i = 1, \ldots, n$$

$$y_i \in Z,\ \text{ただし} l_i \leq y_i \leq u_i,$$

$$l_j, u_j \in Z,\ \text{for } i = 1, \ldots, m$$

ここで

$$a_{ij},\ b_{ij},\ c_{ij},\ d_{ij},\ e_i,\ f_i \text{ は実定数である。}$$

のもとで実数値をとる変数 $x_j$ と整数値をとる変数 $y_j$ 上の目的関数

$$z = \sum_{i=1}^{n} p_i\, x_i + \sum_{i=1}^{m} q_i\, y_i$$

の最小値を求める問題である。

我々はこの問題を解くにあたって分枝限定法を用いる。この方法ではまず変数 $y_i$ の値を整数に限定せずに上の問題を解く。この問題のことを「連続緩和問題」と呼ぶ。もしこの連続緩和問題が整数解を持たなければ、整数解を持つようになるまでもとの問題を木状の探索空間を作りながら2つの部分問題へと分割しつづける。

連続緩和問題はシンプレックス法によって解くことができ、その結果もしもとの問題で整数値をとるべき変数が整数値をとればそれがもとの問題の解でもある。さもなければまず連続緩和問題の解から非整数値 $\bar{y}_s$ をとる整数変数 $y_s$ を選びこれをもとに $l_s \leq y_s \leq \lfloor \bar{y}_s \rfloor$ と $\lfloor \bar{y}_s \rfloor + 1 \leq y_s \leq u_s$ の2つの区間制約を生成し、これらを制約集合に別々に付加することにより2つの部分問題に分割する(図7参照)。この分枝操作を繰り返し、探索空間をより制約の多い部分問題へと分割していく。すると整数値を解に持つような部分問題が得られる。それらの中で最良の解を選択すればそれがもとの問題の解となる。

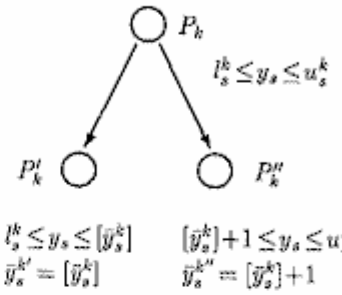


図 7: ノードの分枝

上の分枝の過程では整数解を数え上げているだけなので、もしある部分問題が目的関数の値としてそれまでに得られている最適の整数解よりも良い整数解を持ち得ないということが分かれば、その部分問題を解く必要がなくなる。連続緩和問題の目的関数は、もとの整数問題における目的関数の最適値よりも常に悪くない最適値を取るためこの決定を行うための指標を与えてくれる。すなわちある部分問題において、連続緩和問題がすでに得られている整数解よりも良いような最適解を持たない場合、その部分問題を解く必要がなくなる(これを「限定手続き」と呼ぶ)。

このようにして限定手続きによって得られた部分問題のことを「探索ノード」と呼ぶ。

以下に逐次の探索プロセスにおける探索順序決定のための重要な指針を示す。

1. どのノードに分枝手続きを適用するかを決定する、部分問題(ノード)の優先順位。
2. 探索空間を分割するための、整数変数の選択。

このような選択は全体の探索空間が最も小さくなるように行われなければならない。本研究における並列アルゴリズムにおいてはオペレーションズ・リサーチの分野で有効とされるヒューリスティクスのひとつを採用した([Benichou *et al.* 1971])。

### 5.3.2 分枝限定法の並列化

分枝限定法を並列化する際に我々は分枝手続きで生成される探索の部分木を各プロセッサに分散させ、それぞれのプロセッサに対応する部分問題を逐次で解かせる方法をとった。それぞれの逐次探索プロセスは互いに最新の解と部分

木の枝刈りに関する情報を通信しあうようになっている。我々は逐次探索における最良のヒューリスティクスのひとつを採用した。このヒューリスティクスは部分木の探索順序を制御するものであって、これを利用することで最終結果を得るまでにたどらなければならない節の数を減らすことが出来る。したがって探索アルゴリズムの並列版を設計する際にも各プロセッサ間の負荷を均等化し、枝刈り可能な部分木を出来るだけ早く刈るための通信を行うためにも、このヒューリスティクスは重要な役割を果たす。

上述のような逐次アルゴリズムに基づいて我々は並列アルゴリズムの設計を行った。

我々の並列アルゴリズムでは限定手続きによって生成される部分問題の独立性を利用し、これらを異なったプロセッサに分配している (図8参照)。これらの部分問題間のスケジューリングは、KL1 言語で提供される優先順位制御機能を用いて行っている ([Oki *et al.* 1989] 参照)。暫定的な解をプロセッサ間で通信し、各プロセッサの暫定解を更新する。

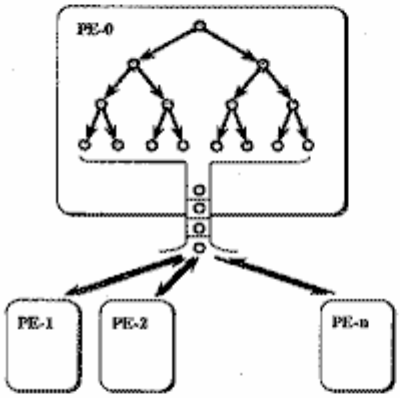


図 8: 並列プロセスの生成

### 5.3.3 実験結果

我々は上述の並列アルゴリズムを KL1 を用いて実装し、混合整数問題の例としてジョブショップ・スケジューリング問題について実験を行った。以下に「4 ジョブ 3 機械問題」について適用した場合の台数効果と解を得るために探索されたノードの数を示す。

表 2: 整数線形制約評価系の台数効果

| プロセッサ数 | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| 台数効果 | 1.0 | 1.5 | 1.9 | 2.3 |
| ノードの数 | 242 | 248 | 395 | 490 |

上の表によればプロセッサ数が増えるにつれて探索されたノードの数も増加していることがわかる。これはひとつにはこのようなタイプの並列アルゴリズムにはつきもののスペキュラティブな計算のためである。またもうひとつにはプロセッサ間の通信の遅れのためである。すなわち、あるプロセッサから他のプロセッサへ暫定解を遅れなしに通信出来れば、不要な枝をただちに刈ることが出来るが、実際には通信には遅れが生じるため、枝刈りが遅れ、その結果不要な計算を行うこともある。

## 5.4 階層制約評価系

### 5.4.1 柔らかい制約と制約階層

我々は [Satoh 1990] において解釈間の順序を表現するようなメタ言語を用いた柔らかい制約の論理的な意味付けを行った。柔らかい制約は、およそ以下のようにして形式的意味付けを行うことが出来る。まず固い制約は 1 階の論理式で表されるものとする。このときこれらの 1 階の論理式をすべて満たすような解釈とは、解になりうるものと考えられる。このとき柔らかい制約はこれらの解釈上の順序であるとみなすことができる。これは最も好ましい解を選択するために解となりうるものに対して適用する選択基準を柔らかい制約が表しているからである。我々はこのような好ましさを直接表現するようなメタ言語を用いた。このメタ言語は最も好ましい解を構文的に定義するため 2 階の論理式へと変換される。

この枠組は厳密で宣言的なものではあるが 2 階の論理式によって定義されるために一般には計算不能である。したがってこのような制約が計算可能であるように記述可能な制約のクラスを限定する必要がある。

この枠組を計算可能にするため我々は以下のような制限を加えた。

1. 領域依存の関係の解釈を固定するため制約の領域を固定した。
2. 柔らかい制約、固い制約共に領域依存の関係のみからなる。

このような制限を導入することで、柔らかい制約は 1 階の論理式で表される。さらに [Satoh and Aiba 1990a] で示したようにこのような制限された柔らかい制約のクラスと階層制約論理型言語 (HCLP 言語)[Borning *et al.* 1989, Satoh and Aiba 1990b] との間には次のような対応関係がある。

HCLP 言語はラベル付き制約を導入することによって拡張された CLP 言語である。HCLP プログラムは次のようなルールからなる。

$$h \ :- \ b_1,...,b_n$$

ここで $h$ は述語、$b_1,...,b_n$ は述語呼びだし、制約、あるいはラベル付き制約である。ラベル付き制約とは次のようなものである。

`label` $C$

ここで $C$ は領域依存の関数記号のみを関数記号に持つような制約であり `label` が制約 $C$ の強さを表すラベルで

ある。

HCLP 言語では [Satoh and Aiba 1990a] で示したように制約階層を用いて、最も好ましい解を計算することが出来る。このような対応に基づいて我々は次のような特徴を持つ制約階層の評価アルゴリズムを PSI マシン上に試作した。

1. 計算をボトム・アップに行うので制約の各組み合わせに対して制約評価系は 1 回だけ呼び出される。
2. 制約評価系の呼びだしによって矛盾する制約集合が見つかった場合、制約のその組み合わせは nogood ということで登録されそれ以降の計算において矛盾する制約集合の発見に用いられる。したがってこれらの制約集合に他の制約を付け加えた制約の組み合わせについては制約評価系を呼び出さない。
3. ある制約集合が nogood に登録されている制約集合を含む場合、制約評価系の呼びだしをせずに矛盾が認識される。

Borning 等は [Borning *et al.* 1989] において制約階層を処理するアルゴリズムを与えている。しかしこのアルゴリズムは別解を求めるのにバックトラックを用いており、制約の同じ組み合わせに関して制約評価系が何度も呼び出されることになる。

我々は上述の特徴を持つこのアルゴリズムに基づき CAL の拡張であるような CHAL(Contrainte Hierarchiques avec Logique) [Satoh and Aiba 1990b] と呼ばれる言語の試作を行った。

#### 5.4.2 制約階層のための並列評価系

我々が PSI 上に試作したアルゴリズムには次のような並列性が見出すことが出来る。

1. 無矛盾な制約集合はボトム・アップに構築されるのでそれぞれ独立な制約集合の無矛盾性のチェックは並列に実行可能である。
2. それぞれ独立な制約集合に関してそれらが nogood に含まれているか否かの判定は並列に実行可能である。
3. 領域依存の制約評価系内部に並列性が見出せる。
4. 解の冗長性のチェックは並列に実行しうる。

これらの並列性のうち最初のものは最も疎粒度であり、Multi-PSI マシン上での実装に最も適している。したがって我々はこの並列性を利用することとした。並列アルゴリズムの特徴は以下の通りである。

1. 各プロセッサにおいては与えられた制約集合から無矛盾な最大の制約集合をボトム・アップに求める。これはプロセッサ毎に並列に行われる。しかし、一旦制約集合が与えられると分散されるようなタスクはもう存在しなくなる。したがってもし暇なプロセッサが忙しいプロセッサにタスクの要求を出し、忙しいプロセッサが自分のタスクを分割して与えることが出来ればそのタスクは暇なプロセッサへ送られる。

表 3: 並列階層制約評価系の効率

| problems | Processors | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 16 |
| Tele4 (秒) | 43 | 32 | 32 | 32 | 29 |
| | 1 | 1.34 | 1.34 | 1.34 | 1.48 |
| 5queen (秒) | 69 | 39 | 26 | 21 | 19 |
| | 1 | 1.77 | 2.65 | 3.29 | 3.63 |
| 6queen (秒) | 517 | 264 | 136 | 77 | 50 |
| | 1 | 1.96 | 3.80 | 6.71 | 10.34 |

2. 我々が並列アルゴリズムを前もって評価したところ、nogood に含まれるかどうかのチェックと解の冗長性のチェックには非常に大きなオーバヘッドがあることが分かった。そこでこれらのチェックは実行の最後の段階で行うこととした。

表 3に 3 つの例題についての台数効果を示す。Tele4 は自然言語におけるあいまい性を解く問題であり、5queen と 6queen はそれぞれ 5 Queen 問題と 6 Queen 問題である。これらの問題はブール制約によって表されており、これらを解くためにブーリアン・ブフバーガ・アルゴリズム [Sato and Sakai 1988, Sakai and Aiba 1989] を用いた。

表 3によれば Tele4 の台数効果は 1.34 であり、5queen、6queen についてはそれぞれ 3.63 と 10.34 であった。ブーリアン・ブフバーガ・アルゴリズムにとって 6queen 問題は比較的大きな問題であり最も大きな台数効果が得られているが、この台数効果はおよそ 16 プロセッサのあたりで頭打ちになっている。このことは負荷が良く分散されていないことを示しており、より良い負荷分散方式を見つけ出す必要がある。

### 5.5 集合制約評価系

集合制約評価系は次のような述語の論理積で表される制約を扱うことができる。

$$F_1(\bar{x}, \bar{X})$$
$$\vdots$$
$$F_n(\bar{x}, \bar{X})$$

ここで各述語 $F_i(\bar{x}, \bar{X})$ は述語記号 $\in$、$\subseteq$、$\neq$、$=$ と関数記号 $\cap$、$\cup$、$\sim$、集合の要素を表す変数 (以降「要素変数」と呼ぶ) $\bar{x}$、集合を表す変数 (以降「集合変数」と呼ぶ) $\bar{X}$、および要素の定数記号から構成される述語である。

上のような制約に対して制約評価系は次の形の解を返す。

$$f_1(\bar{x}, \bar{X}) = 0$$
$$\vdots$$
$$f_l(\bar{x}, \bar{X}) = 0$$
$$h_1(\bar{x}) = 0$$
$$\vdots$$
$$h_m(\bar{x}) = 0$$

ここで $h_1(\bar{x}) = 0, \ldots, h_m(\bar{x}) = 0$ は制約を満たすための必要充分条件を与えるものである。このとき要素変数に関する方程式系 $h_1, h_2, \ldots, h_m$ の解を $f_1, f_2, \ldots, f_l$ に代入した結果はもとの方程式系の解となる。

まず次の例について考える。

$$\begin{aligned} A^{\sim} \cap C^{\sim} \cap E^{\sim} &= \emptyset \\ C \cup E &\supseteq B \\ C \cup E &\supseteq D \\ D \cap B^{\sim} &\supseteq A \\ A^{\sim} \cap B &\subseteq D \\ A \cup B &\supseteq D \end{aligned}$$

ここで $A^{\sim}$ は $A$ の補集合を表すものとする。

集合のクラスがブール代数をなすことからこれらの制約をブール制約とみなすことができる。したがってこの問題はブーリアン・グレブナ基底を求めることで次のように解くことができる。

$$\begin{aligned} D &= A + B \\ E * C &= E + C + 1 \\ A * B &= 0 \end{aligned}$$

ここでこれらには要素変数も集合変数も含まれていないことに注意されたい。すなわちこれらは変数 $A$、$B$、$C$、$D$、$E$ からなるブール式である。しかしこのことはあらゆる集合制約について成り立つことではない。

たとえば次のような上の例にさらに 3 つの述語を加えた例について考えてみる。

$$\begin{aligned} A^{\sim} \cap C^{\sim} \cap E^{\sim} &= \emptyset \\ C \cup E &\supseteq B \\ C \cup E &\supseteq D \\ D \cap B^{\sim} &\supseteq A \\ A^{\sim} \cap B &\subseteq D \\ A \cup B &\supseteq D \\ (C \cap \{x\}) \cup (E \cap \{p\}) &= D \cap \{x, p\} \\ x &\notin A \\ p &\notin B \end{aligned}$$

ここで $x$ は要素変数であり $p$ は要素を表す定数記号とする。

この問題は上の例とは異なりブール式で表すことはできない。例えば最後の式は $\{p\}$ を係数として $\{p\} * B = 0$ と表される。このような一般的なブール式を扱うために我々はブーリアン・グレブナ基底の概念を拡張した [Sato *et al.* 1991]。これにより集合制約評価系の実現が可能となったのである。

上の制約系に対して集合制約評価系は次のような解を返す。

$$\begin{aligned} D &= A + B \\ E * C &= E + C + 1 \\ A * B &= 0 \\ \{x\} * E * B &= \{x\} * E + \{x\} * B + \{x\} \\ \{p\} * C * A &= \{p\} * C + \{p\} * A + \{p\} \\ \{p\} * E &= \{p\} * A \\ \{x\} * C &= \{x\} * B \\ \{x\} * A &= 0 \\ \{p\} * B &= 0 \\ \{p\} * \{x\} &= 0 \end{aligned}$$

この例では $\{p\} * \{x\} = 0$ が制約の充足条件である。この条件は $x \neq p$ のとき、かつそのときに限り成立する。この場合もとの制約系を満たすような $A$、$B$、$C$、$D$ は常に存在する。次がその標準形である。

$$\begin{aligned} D &= A + B \\ E * C &= E + C + 1 \\ A * B &= 0 \\ \{x\} * E * B &= \{x\} * E + \{x\} * B + \{x\} \\ \{p\} * C * A &= \{p\} * C + \{p\} * A + \{p\} \\ \{p\} * E &= \{p\} * A \\ \{x\} * C &= \{x\} * B \\ \{x\} * A &= 0 \\ \{p\} * B &= 0 \end{aligned}$$

なお本制約評価系の詳細に関しては [Sato *et al.* 1991] を参照されたい。

### 5.6 制約集合の依存性解析

応用プログラム記述の実験を通じて我々は CLP 言語の強力な記述力はプログラミングの際の大きな助けとなると結論付けることができた。これはユーザが記述しなければならないことが問題の本質的な性質だけで、その問題を解く方法については記述する必要がないということによるものである。

一方制約評価系の持つ一般性と強力さが逆に欠点となることもある。すなわちある場合、特に CAL や GDCC の代数制約評価系のように非常に強力な制約評価系においてはその一般性ゆえに効率の良い実装が難しい。言語処理系のサブシステムのひとつとして制約評価系の効率はこのような言語の実装における重要な問題点のひとつである [Marriott and Sondergaard 1990, Cohen 1990]。

一般的にある制約集合に関して制約評価の効率は制約評価系に入力される制約の順番に依存する。しかし CAL のような逐次 CLP においては制約評価系は推論エンジンが SLD-導出によって集めた制約を解くため、この順序はプログラム中の位置によって決定される。一方 GDCC のような並列 CLP においては制約評価系への制約の入力順序は逐次の場合以上に重要となる。これは並列 CLP においては推論エンジンと制約評価系とが並列に動作するために

この順序がプログラム中の位置によっては決定されないためである。

CAL および GDCC においてグレブナ基底の計算は時間を要するものでありブフバーガ・アルゴリズムは最悪の場合の計算量は2重の冪になる [Hofmann 1989]。我々は代数制約評価系の効率を制約の順序の変更によって向上させることを試みた。

我々はまず依存性解析に基づいて決定される制約の順序に関する考察を行った [Nagai 1991, Nagai and Hasegawa 1991]。この解析はデータ・フロー解析、制約集合の収集、制約集合の依存性解析、ゴールの順序と変数の優先順位の決定からなる。

データ・フローの解析のため我々は SLD- 導出に基づくトップ・ダウン解析を用いた。与えられたゴールとプログラムとに対してゴールから述語の呼びだしを制約評価系の呼びだしを行わずにたどり、変数の束縛情報と制約とを収集する。

この解析において制約集合は二部グラフと呼ばれるグラフによって表現され、その代数的構造は DM 分解 [Duimage and Mendelsohn 1963] によって抽出される。これは制約集合に対応する行列の要素の並び換えによりブロック三角化行列を求めるものである。

解析の結果、制約集合は比較的独立性の高い制約の部分集合に分割されることになる。これらの分割された部分集合は部分集合間で共有される変数ができる限り少なくなるように求められる。この分割と共に部分集合間の共有変数、および同じ部分集合内の制約間の共有変数が求められる。これらの結果に基づいてゴールの順序と変数の優先順位とが求められる。

2つの幾何学の定理の証明問題 [Kapur and Mundy 1988, Kutzler 1988] についてこの方法を適用した際の結果を示す。これらの定理はひとつは三角形に関する三垂線の定理であり8変数に関する5つの制約からなり、もうひとつは九点円の定理と呼ばれ12変数に関する9つの制約からなるものである。我々の方法を適用した結果前者では3.2倍、後者では276倍の効率向上が得られた。

## 6 CAL と GDCC による応用システム

CAL および GDCC の有用性を示すためにいくつかの応用システムを試作した。この節ではこれらの中から2つ、ハンドリング・ロボット設計支援システムと、ボロノイ図作成プログラムについて述べる。

### 6.1 ハンドリング・ロボット設計支援システム

ハンドリング・ロボットの設計過程は、基本構造設計と内部構造設計の2つのフェーズに分けることが出来る [Takano 1986]。基本構造設計とはロボットの自由度、関節数、アーム長といった骨組みに関する部分の設計であり、一方内部構造設計とは各関節のモータのトルクといったような内部の詳細に関する設計である。ここで述べるハンドリング・ロボット設計支援システムは主に基本構造設計の支援を目的とするものである。

現在ハンドリング・ロボットの設計は次のように行われている。

1. まず直交座標型、円柱座標型、多関節型といったロボットの構造の形式をそのロボットに対する要求に基づいて決定する。
2. 次いでエンド・エフェクタと各関節との関係を表す方程式系を導き出す。そしてこの方程式系を必要な形へと変形する。
3. さらにこの変形された方程式系に基づいて FORTRAN や C といったプログラミング言語を用いて、設計を行っているロボットの解析のためのプログラムを作成する。
4. このプログラムを実行し設計の評価を行う。もしその結果が満足なものであれば設計過程は終了するが、そうでなければ満足な結果が得られるまで全体の過程を繰り返す。

ハンドリング・ロボットの設計過程に CLP のパラダイムを適用するとプログラムを作成するには上の2で述べたような関係を記述するだけで良く、しかも方程式系の変形はそのプログラムの実行によって行われる。したがって上の2と3の過程が計算機によって支援されることとなる。

#### 6.1.1 制約に基づく機構学と静力学のプログラム

ロボットの機構学とはエンド・エフェクタの位置および姿勢と、各アームの長さおよび各関節の回転角度との関係を表すものである。これらのうちエンド・エフェクタの位置と姿勢をハンド空間パラメータと呼び、各アームの長さおよび各関節の回転角度を関節空間パラメータと呼ぶ。ロボットの静力学は関節空間パラメータと、エンド・エフェクタに作用する力、各関節に作用するトルクとの関係を表すものである [Tohyama 1989]。

ハンドリング・ロボットを扱うプログラムを作成するにあたって、我々はこのプログラムをロボットの基本構造とは独立なものとなるようにした。すなわち機構学、静力学のプログラムはこれらのプログラムに対する問い合わせの変更を行うだけで任意の構造のロボットを扱えるようにした。

実際、設計中のロボットの構造を表す行列は問い合わせの中でリストのリストの形で表される。この引数の構造を変更することで単一のプログラムで任意構造のロボットを扱うことができるのである。

たとえば次に示すのは機構学プログラムへの3関節3アームのロボットのための問い合わせである。

```
robot([[cos3, sin3, 0,  0, z3, 0, 0, 1],
       [cos2, sin2, x2, 0, 0,  1, 0, 0],
       [cos1, sin1, 0,  0, z1, 0, 0, 1]],
       5, 0, 0, 1, 0, 0, 0, 1, 0,
```

```
px, py, pz, ax, ay, az, cx, cy, cz).
```

ここで最初の引数は上で述べたようなハンドリング・ロボットの構造を表し、px、py、pzはエンド・エフェクタの位置を、ax、ay、az、cx、cy、czは、エンド・エフェクタの姿勢を表すための互いに直交する単位ベクトルの成分である。sinとcosは各関節の回転角度を表し、z3、x2、z1は各アームの長さを表す。この問い合わせに対してこの機構学プログラムは次のような解を返す。

```
cos1^2 = 1-sin1^2
cos2^2 = 1-sin2^2
cos3^2 = 1-sin3^2
    px = -5*cos2*sin3*sin1+z_3*sin2*sin1
         +5*cos3*cos1+x_2*cos1
    py = 5*cos3*sin1+x_2*sin1
         +5*cos1*cos2*sin3-z_3*cos1*sin2
    pz = 5*sin3*sin2+z_1+z_3*cos2
    ax = -1*cos2*sin3*sin1+cos3*cos1
    ay = cos3*sin1+cos1*cos2*sin3
    az = sin3*sin2
    cx = -1*cos1*sin3-cos3*cos2*sin1
    cy = -1*sin3*sin1+cos3*cos1*cos2
    cz = cos3*sin2
```

すなわちエンド・エフェクタの位置、姿勢を表すパラメータが、各アーム長と各関節の回転角度の関数の形で得られる。

ハンド空間パラメータから関節空間パラメータを求める問題を「順機構学」と呼び、その逆を「逆機構学」と呼ぶがこのプログラムはCLP言語によるプログラムの特徴により、その両方を扱うことができる。

またこのプログラムはユーザが定義した基本構造を持つ任意のハンドリング・ロボットを扱えるようなプログラムの生成系であると考えることもできる。

静力学プログラムもこの機構学プログラムと同様の特徴を持っており、単一のプログラムで問い合わせを変更するだけで任意の構造のハンドリング・ロボットを扱うことができる。

#### 6.1.2 設計支援システムの構築

ハンドリング・ロボット設計支援システムは次のような機能を持っているべきであると考える。

1. 任意構造のロボットについて機構学および静力学を表現する制約が生成できること。
2. 順機構学、逆機構学の両方が解けること。
3. 各関節に作用するトルクの大きさが計算できること。
4. 操作性を評価できること。

上の要求を満たすためにハンドリング・ロボット設計支援システムを次のような3つのGDCCプログラムから構成する。

**Kinematics** 機構学プログラム
**Statics** 静力学プログラム
**Determinant** 行列式を計算するプログラム

これらのうち**Kinematics**と**Statics**については、すでに上で述べた。またハンドリング・ロボットの操作性を評価する行列をヤコビアンと呼ぶが、これは**Statics**から得られる。さらに**Determinant**はヤコビアンの行列式を計算するために用いられる。この行列式は「可操作度」と呼ばれ、ハンドリング・ロボットの操作性を評価するために用いられる[Yoshikawa 1984]。

各変数の値を求めるためには、システムはGDCCの実根の近似値を求める機能を利用する。

### 6.2 ボロノイ図の作成

我々はまたボロノイ図作成プログラムをGDCCを用いて試作した。

制約を用いることで、プログラムを複雑なアルゴリズムを記述することなしに作ることが可能となる。したがって、制約を用いるとボロノイ図はこの図の性質、あるいは定義を記述するだけで作成することができる。このプログラムは各点毎のボロノイ多角形を、並列に求めるものである。

#### 6.2.1 ボロノイ図の定義

与えられた平面上の点の有限集合$S$に対して、ボロノイ図とはその平面を$S$のそれぞれの点に最も近い部分に分割して得られる図形である[Preparata and Shamos 1985]。

最も単純な場合この2点間の距離はユークリッドの距離によって定義される。

ある平面上の$N$点の集合$S$が与えられたとき、$S$中の各点$P_i$に対してボロノイ多角形とは次によって定義される$V(P_i)$である。

$$V(P_i) = \{P \mid d(P,P_i) < d(P,P_j), \forall j \neq i\}$$

ここで$d(P,P_i)$は点$P$と点$P_i$の距離である。

ボロノイ図とはそれぞれの部分がボロノイ多角形であるような分割のことをいう(図9参照)。ボロノイ図の各頂点のことをボロノイ点といい、各辺をボロノイ辺という。

ボロノイ図は物理学や都市工学といった広い分野で利用されている。

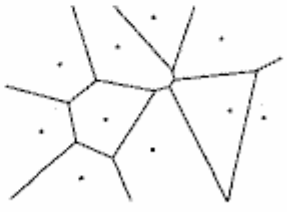

図9: ボロノイ図の例

### 6.2.2 詳細

ボロノイ図を作成する方法は次の2つに大別される。

1. 逐次添加法 ([Green and Sibson 1978])
2. 分割統治法 ([Shamos and Hoey 1975]).

しかしながらボロノイ図を作成する最も簡単な方法はボロノイ多角形をひとつづつ作っていく方法である。

与えられた2点 $P_i$ と $P_j$ に対して $P_j$ よりも $P_i$ に近い点の集合とは線分 $\overline{P_iP_j}$ の垂直二等分線により分割される半平面のうち $P_i$ を含む方である。この垂直二等分線を $H(P_i, P_j)$ で表すことにする。

点 $P_i$ のボロノイ多角形は次のような式によって得られる。

$$V(P_i) = \cap_{j \neq i} H(P_i, P_j) .$$

GDCC の線形制約評価系を用いることで、ボロノイ多角形は上述のボロノイ多角形を直接作成する方法を表した以下のアルゴリズムによって作成される。

```
E_s ← {x ≥ 0, y ≥ 0, x ≤ x_Max, y ≤ y_Max}
{loop a}
for i = 1 to n
    CF_0 ← linear_constraint_solver(E_s)
    for j = 1 to n
        if(j ≠ i) then
          E_j ← y ≤ (P_jx − P_ix)/(P_jy − P_iy) · x
                  +(P_jy^2 + P_ix^2 − P_iy^2 − P_jx^2)/2 · (P_jy − P_iy)†
          CF_j ← linear_constraint_solver(E_j ∪ CF_{j−1})
          {eq_1, eq_2, ..., eq_k}(0 ≤ k ≤ n) を CF_j の不等号を
          等号に換えて得られる等式の集合とする。
    {loop b}
    for l = 1 to k
        vertices := {}
        m := 1
        while (m =< k &
                vertices の要素数 ≠ 2)
                pp ← intersection(eq_l, eq_m)
                if pp が制約集合 CF_i を満たす
                  then vertices := {pp} ∪ vertices
                m := m + 1
        vertices の間の線分をボロノイ辺に追加する。
end.
```

このアルゴリズムの前半はボロノイ多角形を求めるために各点 $P_i$ とそれ以外のすべての点を結ぶ線分の垂直二等分線を求めて冗長なものを除去している。後半ではボロノイ辺の計算を行う。

上のアルゴリズムを並列マシン上で実装するためにループ a の中の各 $i$ に対する手続きをプロセスのグループに割り当てた。すなわち全体では $n$ 個のグループが必要となる。またループ b の中の各 $l$ に対する手続きを同じプロセスグループの中のプロセスに割り当てた。すなわちそれぞれのプロセスグループは $k$ 個のプロセスからなることになる。結局、これらの $n \times k$ 個のプロセスを複数のプロセッサに割り当てることになる。

†この不等式は線分 $\overline{(P_i, P_j)}$ の垂直二等分線によって分割される半平面を表す。

表 4: 実行時間とリダクション数

| Points | Processors 1 | 2 | 4 | 8 | 15 | Reductions ( ×1000) |
|---|---|---|---|---|---|---|
| 10 | 130 | 67 | 33 | 17 | 16 | 5804 |
| | 1 | 1.936 | 3.944 | 7.377 | 7.844 | |
| 20 | 890 | 447 | 241 | 123 | 88 | 42460 |
| | 1 | 1.990 | 3.685 | 7.218 | 10.077 | |
| 50 | 4391 | 2187 | 1102 | 566 | 336 | 210490 |
| | 1 | 2.007 | 3.981 | 7.749 | 13.065 | |
| 100 | 17287 | 8578 | 4305 | 2191 | 1263 | 830500 |
| | 1 | 2.015 | 4.014 | 7.887 | 13.679 | |
| 200 | 52360 | 26095 | 13028 | 6506 | 3500 | 2458420 |
| | 1 | 2.006 | 4.018 | 8.047 | 14.959 | |
| 400 | 220794 | 110208 | 54543 | 27316 | 14819 | 10161530 |
| | 1 | 2.003 | 4.048 | 8.082 | 14.899 | |

### 6.2.3 結果

表4に点の数を 10 から 400 までの場合を 1 から 15 プロセッサで処理した場合の処理時間と台数効果を示す。

この結果によると点の数が充分大きい場合にはプロセッサ数に比例する台数効果が得られていることがわかる。

このアルゴリズムを用いることによりボロノイ図作成の問題を、絶対性能はともかく、非常に容易に解決することができる。実際プログラムの大きさで比較するとこのアルゴリズムは逐次添加法にくらべておよそ 3 分の 1 程度で記述可能である。

## 7 まとめ

FGCS プロジェクトにおいて我々は 2 種類の CLP 言語、逐次制約論理プログラミング言語 CAL と並列制約論理プログラミング言語 GDCC の処理系の試作を知識プログラミングモジュールの要素技術の確立のために行い、いくつかの応用プログラムを開発した。我々の研究の目的は知識処理に適した強力で高水準な言語を構築することである。制約は知識表現と知識処理の両方において非常に重要な役割を果たすことは広く知られているところであり、我々はこの分野における高水準言語として CLP 言語が有望な候補であると考えている。

我々の試作した CAL および GDCC は、CLP(𝑹) や Prolog III、あるいは CHIP といった他の CLP 言語と比較して次のような特徴を持っている。

- CAL や GDCC では非線形代数方程式制約の処理ができる。
- 代数制約評価系は制約集合が有限個の解を持つとき実根の近似値を求める機能を持つ。

- CALやGDCCは多重環境操作系を持つ。すなわち1つ以上の解制約集合がある場合でもユーザはそれらを柔軟に取り扱うことができる。
- ユーザは複数の制約評価系を使うことができる。さらに自分自身のための制約評価系を実装することができる。

このようにCALやGDCCを使うことによって複素数上の非線形代数方程式、真偽値上の関係式、集合とその要素に関する関係式、実数上の線形方程式/線形不等式などを制約として記述することができる。

ハンドリング・ロボットの分野で代数制約評価系の応用プログラムを書き始めて以来、我々は1変数方程式の実根の近似値を求める機能が必要であると考えてきた。我々は実根の近似値を求める機能をCALに付加し、ブフバーガ・アルゴリズムを近似値を扱えるように修正することによりこの機能を実現した。

このような拡張の結果、我々はある変数が2つ以上の値を持ち得るという状況に直面した。このような状況を取り扱うため、我々はCALにはコンテキスト木を導入し、一方GDCCにはブロックという機能を言語仕様に追加した。GDCCにおけるブロックは、単に複数の値を扱うためだけではなく制約評価系の失敗の局所化および推論エンジンと制約評価系の同期の問題をも扱うことができる。

CALについての将来の課題としては次がある。

1. メタ機能:
   現在ユーザはプログラムからコンテキスト木を処理することができない。すなわち現在のCALには解制約集合を処理するようなメタ機能が不足している。
2. CALプログラムの部分評価:
   依存性解析による制約集合の解析は試みたものの、この手法は部分評価や抽象解釈と組み合わせてより効力を発揮するものと考える。
3. より多くの応用プログラム:
   CALの応用プログラムはまだ依然として少ない。様々な領域で多くの応用プログラムを書くことにより、より強力なCLP言語を実現するためのアイデアが得られるものと考える。このため我々は現在CAL処理系をCommon ESPと呼ばれるUNIX上で動作するESPを用いて実装している。これによりCALは様々なマシン上で稼働できるようになる。

一方GDCCについての将来の課題としては次がある。

1. 多重環境の取り扱い:
   現在のGDCCには多重環境を扱う機能が備わっているが、ユーザはすべてを陽に記述しなければならない。したがってGDCC言語の仕様に多重環境を扱うためのより高水準のツールが必要であると考えている。
2. より効率の良い制約評価系:
   様々な制約評価系の絶対性能と台数効果の両面における改良が必要である。
3. より多くの応用プログラム:
   並列CLPは非常に新しい言語であり、多くの応用プログラムを書くことによってより効率の良い強力な言語が得られるものと考える。

CALとGDCCにおける経験と上に述べた将来の課題とに基づき、我々はGDCCの仕様とその実装方式に関する改良を行う予定である。

我々はこれらの改良や様々な領域におけるさらなる応用プログラム記述実験を通じて、高水準で充分に効率の良い制約論理プログラミング言語に必要な事項が明確になるものと考えている。

## 謝辞

本論文で述べた制約論理プログラミング言語に関する研究はICOT第4研究室の研究員と再委託メーカ、およびCLPワーキンググループとの緊密な協力体制のもとで行われたものである。

我々は特に、絶え間ない援助と有益な助言を賜ったICOT研究所の渕所長、古川次長、ワーキンググループの主査である九州大学の雨宮先生に謝意を表したい。

またICOT外部の研究者の方々や、メーカの方々、ワーキンググループのメンバの方々にその有益で熱意ある議論や助言、御協力に対して感謝したい。

最後に第4研究室のすべての研究員、坂井テーマリーダ、川岸主任研究員、佐藤(健)主任研究員、佐藤(晋)主任研究員、佐藤(洋)主任研究員、寺崎主任研究員、岩山研究員、沢田研究員、毛受研究員、また現在コンピューフレックスジャパン(株)勤務で、GDCC処理系の最初の設計、実装を行ったD. J. Hawley氏に感謝する。

## 参考文献

[Aiba *et al.* 1988] 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.

[Backelin and Fröberg 1991] J. Backelin and R. Fröberg. How we proved that there are exactly 924 cyclic 7-roots. In S. M. Watt, editor, *Proceedings of ISSAC'91*. ACM, July 1991.

[Benichou *et al.* 1971] M. Benichou, L. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, (1), 1971.

[Boege *et al.* 1986] W. Boege, R. Gebauer, and H. Kredel. Some Examples for Solving Systems of Algebraic Equations by Calculating Gröbner Bases. *Journal of Symbolic Computation*, 2(1):83–98, 1986.

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

[Buchberger 1985] B. Buchberger. Gröbner bases: An Algorithmic Method in Polynomial Ideal Theory. In N. Bose, editor, *Multidimensional Systems Theory*, pages 184–232. D. Reidel Publ. Comp., Dordrecht, 1985.

[CAL Manual] Institute for New Generation Computer Technology. *Contrante Avec Logique version 2.12 User's manual.* in preparation.

[Chikayama 1984] T. Chikayama. Unique Features of ESP. In *Proceedings of FGCS'84*, pages 292–298, 1984.

[Chikayama *et al.* 1988] T. Chikayama, H. Sato, and T. Miyazaki. Overview of Parallel Inference Machine Operationg System (PIMOS). In *International Conference on Fifth Generation Computer Systems*, pages 230–251, 1988.

[Clarke *et al.* 1990] E. M. Clarke, D. E. Long, S. Michaylov, S. A. Schwab, J. P. Vidal, and S. Kimura. Parallel Symbolic Computation Algorithms. Technical Report CMU-CS-90-182, Computer Science Department, Carnegie Mellon University, October 1990.

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

[Colmerauer 1987] 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.* 1988] 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.

[Dulmage and Mendelsohn 1963] A. L. Dulmage and N. S. Mendelsohn. Two algorithms for bipartite graphs. *Journal of SIAM*, 11(1), March 1963.

[Green and Sibson 1978] P. J. Green and R. Sibson. Computing Dirichlet Tessellation in the Plane. *The Computer Journal*, 21, 1978.

[Hofmann 1989] C. M. Hoffmann. *Gröbner Bases Techniques*, chapter 7. Morgan Kaufmann Publishers, Inc., 1989.

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

[Kapur and Mundy 1988] K. Kapur and J. L. Mundy. Special volume on geometric reasoning. *Artificial Intelligence*, 37(1-3), December 1988.

[Kutzler 1988] B. Kutzler. *Algebraic Approaches to Automated Geometry Theorem Proving.* PhD thesis, Research Institute for Symbolic Computation , Johannes Kepler University, 1988.

[Lloyd 1984] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1984.

[Maher 1987] 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.

[Marriott and Sondergaard 1990] K. Marriott and H. Sondergaard. Analysis of constraint logic programs. In *Proc. of NACLP '90*, 1990.

[Menju *et al.* 1991] S. Menju, K. Sakai, Y. Satoh, and A. Aiba. A Study on Boolean Constraint Solvers. Technical Report TM 1008, Institute for New Generation Computer Technology, February 1991.

[Nagai 1991] Y. Nagai. Improvement of geometric theorem proving using dependency analysis of algebraic constraint (in Japanese). In *Proceedings of the 42nd Annual Conference of Information Processing Society of Japan*, 1991.

[Nagai and Hasegawa 1991] Y. Nagai and R. Hasegawa. Structural analysis of the set of constraints for constraint logic programs. Technical report TR-701, ICOT, Tokyo, Japan, 1991.

[Oki *et al.* 1989] H. Oki, K. Taki, S. Sei, and S. Furuichi. Implementation and evaluation of parallel Tsumego program on the Multi-PSI (in Japanese). In *Proceedings of the Joint Parallel Processing Symposium (JSPP'89)*, 1989.

[Ponder 1990] C. G. Ponder. Evaluation of 'Performance Enhancements' in algebraic manipulation systems. In J. Della Dora and J. Fitch, editors, *Computer Algebra and Parallelism*, pages 51–74. Academic Press, 1990.

[Preparata and Shamos 1985] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer-Verlag, 1985.

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

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

[Sato and Aiba 1991] S. Sato and A. Aiba. An Application of CAL to Robotics. Technical Report TM 1032, Institute for New Generation Computer Technology, February 1991.

[Sato and Sakai 1988] Y. Sato and K. Sakai. Boolean Gröbner Base, February 1988. LA-Symposium in winter, RIMS, Kyoto University.

[Sato *et al.* 1991] Y. Sato, K. Sakai, and S. Menju. Solving constraints over sets by Boolean Gröbner bases (in Japanese). In *Proceedings of The Logic Programming Conference'91*, September 1991.

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

[Satoh and Aiba 1990a] K. Satoh and A. Aiba. Computing Soft Constraints by Hierarchical Constraint Logic Programming. Technical Report TR-610, ICOT, Tokyo, Japan, 1990.

[Satoh and Aiba 1990b] K. Satoh and A. Aiba. Hierarchical Constraint Logic Language: CHAL. Technical Report TR-592, ICOT, Tokyo, Japan, 1990.

[Senechaud 1990] P. Senechaud. Implementation of a parallel algorithm to compute a Gröbner basis on Boolean polynomials. In J. Della Dora and J. Fitch, editors, *Computer Algebra and Parallelism*, pages 159–166. Academic Press, 1990.

[Shamos and Hoey 1975] M. I. Shamos and D. Hoey. Closest-point problems. In *Sixteenth Annual IEEE Symposium on Foundations of Computer Sceience*, 1975.

[Siegl 1990] K. Siegl. Gröbner Bases Computation in STRAND: A Case Study for Concurrent Symbolic Computation in Logic Programming Languages. Master's thesis, CAMP-LINZ, November 1990.

[Takano 1986] M. Takano. Design of robot structure (in Japanese). *Journal of Japan Robot Society*, 14(4), 1986.

[Terasaki *et al.* 1992] S. Terasaki, D. J. Hawley, H. Sawada, K. Satoh, S. Menju, T. Kawagishi, N. Iwayama, and A. Aiba. Parallel Constraint Logic Programming Language GDCC and its Parallel Constarint Solvers. In *International Conference on Fifth Generation Computer Systems*, 1992.

[Tohyama 1989] S. Tohyama. *Robotics for Machine Engineer (in Japanese)*. Sougou Denshi Publishing Corporation, 1989.

[Ueda and Chikayama 1990] K. Ueda and T. Chikayama. Design of the Kernel Language for the Parallel Inference Machine. *The Computer Journal*, 33(6), December 1990.

[Vidal 1990] J. P. Vidal. The Computation of Gröbner bases on a shared memory multi-processor. Technical Report CMU-CS-90-163, Computer Science Department, Carnegie Mellon University, August 1990.

[van Emden and Kowalski 1976] M. H. van Emden and R. A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4), October 1976.

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

[Yoshikawa 1984] T. Yoshikawa. Measure of manipulatability of robot arm (in Japanese). *Journal of Japan Robot Society*, 12(1), 1984.