有限領域上の等式制約コンパイルのための一最適化手法とその実装

兼子聡介[†] 周能法[††] 長澤勲[†††]

近年の重要なプログラミングパラダイムの 1 つに，制約論理プログラミング（以下，CLP）がある．CLP は高い宣言性を有し，プログラムの開発効率や保守性を大幅に向上させるものとして期待されている．この中で CLP（FD）は，離散的な有限領域上の制約を扱う CLP の一種で，様々な探索問題に適応できることが知られ，近年，高性能な処理系の実現が重要な研究課題になってきた．ところが，これらの探索問題は，問題ごとにその特性が異なるため，一般的な解法により，そのすべてを効率良く処理できる処理系の実現は困難である．この問題に対処するため，筆者の 1 人周は遅延節を利用する方法を提案し，これを用いて問題の特性を用いた種々のアルゴリズムを実現できることを示した．しかし周が実装したアルゴリズムでは探索空間の大きな問題に対しては，十分な性能は得られなかった．そこで本論文では，部分ルックアヘッド・アルゴリズムと完全ルックアヘッド・アルゴリズムを組み合わせて制約を処理することにより，この問題を解決する方法を提案する．この方法では，制約内に変数が 3 つ以上含まれるときは部分ルックアヘッド・アルゴリズムを用い，制約内の変数が 2 つになると完全ルックアヘッド・アルゴリズムを用いる．実験結果から，この手法が一般的に優れていることが確認できた．

An Optimization Method for Compiling Equality Constraints over Finite-Domains and Its Implementation

Sosuke Kaneko,[†] Neng-Fa Zhou[††] and Isao Nagasawa[†††]

Constraint propagation is one of the key operations employed in CLP (FD). There are several constraint propagation algorithms available. It is a trade-off between the cost of reducing domains and that of backtracking to decide which algorithm is better. *Delay clause* is a good high-level intermediate language for implementing constraint-propagation based CLP (FD) systems. With delay clauses, we can easily implement various constraint propagation algorithms. Actually, Zhou implemented a high performance constraint system over B-Prolog by using delay clauses. The current constraint system employs the *partial look-ahead* algorithm that checks the interval consistency of constraints. The cost for reducing domains in this algorithm is low, but the cost of backtracking may be astonishingly high for big problems. To deal with this problem, we introduce new primitives on domain variables to make it possible to implement the *full look-ahead* algorithm and propose a hybrid algorithm for compiling equality constraints. The experimental results show our algorithm is better than either of the partial look-ahead and the full look-ahead algorithm in general.

1. はじめに

近年の重要なプログラミングパラダイムの 1 つに，制約論理プログラミング（Constraint Logic Programming；以下，CLP）がある．CLP は高い宣言性を有し，プログラムの開発効率や保守性を大幅に向上させるものとして期待されている．この中で CLP（FD）は，離散的な有限領域上の制約☆を扱う CLP の一種で，スケジューリング，画像解釈，時間推論等の様々な探索問題に適応できることが知られ[7]，近年，高性能な処理系の実現が重要な研究課題になってきた．ところが，これらの探索問題は，問題ごとにその特性が異なるため，一般的な解法により，そのすべてを効率良

† 九州工業大学大学院情報工学研究科
Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology
†† ニューヨーク市立大学ブルックリン校情報科学部
Department of Computer and Information Science, Brooklyn College, The City University of New York
††† 九州工業大学情報工学部
Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology

☆ 本論文では，整数上の線形等式制約，線形不等式制約，線形非等式制約のみを扱う．

く処理する処理系の実現は困難である．そこで筆者らは，抽象度の高い中間言語を用いて，個々の問題に適した様々な制約処理系を実現する方法を研究している．

CLP（FD）処理系の実装法としては次の2つが代表的である．1つは，CHIP[1]のように，WAM（Warren Abstract Machine）と呼ばれるProlog用抽象マシンを拡張し，制約処理機構を抽象マシン内に完全に組み込む方法である．もう1つは，GNU Prolog[2]のように，ある要素的な制約を中間言語として利用し，一般の制約をこれに還元する方法である．前者の処理系において上述の問題に対処するには，抽象マシンをさらに拡張して，様々な制約伝播アルゴリズム☆1を実装する必要がある．しかし，この方法は抽象マシンが無限に複雑化するため現実的ではない．後者の処理系の場合，中間言語の記述力が問題となる．GNU-Prologの中間言語の記述力は，様々な制約伝播アルゴリズムを記述するには不十分である．

上述の問題に対処するため，筆者の1人周は遅延節を利用する方法を提案した[10]．遅延節はMeierが提案し[5]周が拡張したもので，論理式の評価順序を制御する機能を持つ．**図1**は周が実装したCLP（FD）処理系の概観である．周はまず遅延節および領域変数☆2を処理できるように，Prolog処理系[8]（以下，B-Prolog）を拡張した．次にこのB-Prolog上に制約変換系を実装した．この制約変換系では，入力されたCLP（FD）プログラムを遅延節とPrologプログラムからなる中間コードに変換する．制約変換系はPrologで記述されているため改良が容易であり，上述の問題に容易に対処できる．周が実装したCLP（FD）処理系は，現在最も高性能な処理系の1つであるGNU Prologに匹敵する性能を示した[10]．しかし探索空間が大きな問題に対しては十分な性能は得られなかった．これは制約伝播アルゴリズムとして部分ルックアヘッド・アルゴリズム☆3を用いたためで，このような問題に対しては解探索時のバックトラックに要する計算量が大きくなることがあるからである．

そこで本論文では，部分ルックアヘッド・アルゴリズムと完全ルックアヘッド・アルゴリズム☆3を組み合わせ，探索空間の大きな問題も効率的に処理できる方法（以下，ハイブリッド・アルゴリズム）を新たに提案する．

筆者らはまず領域変数を拡張し，完全ルックアヘッド・

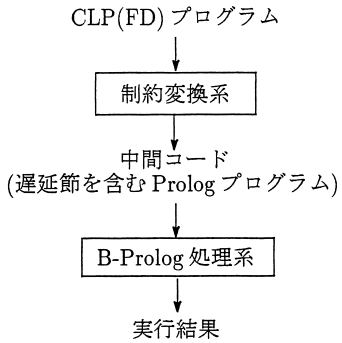

図1 CLP（FD）処理系の概観
Fig. 1 Overview of the CLP (FD) system.

アルゴリズムの1つであるAC-5アルゴリズム[4],☆4を実装できるようにB-Prologを拡張した．次にハイブリッド・アルゴリズムを実行する制約伝播器☆5を生成できるように制約変換系を改良した．このアルゴリズムでは，制約内に変数が3つ以上含まれる場合は部分ルックアヘッド・アルゴリズムを用い，制約内の変数が2つになるとAC-5アルゴリズムを用いる．実験結果から，ハイブリッド・アルゴリズムは探索空間の大きな問題に対して優れた性能を示すことが確認できた．

本論文の構成は以下のようである．2章では，準備として，本研究の基礎となる概念について，3章では領域変数および遅延節について述べる．4章ではAC-5アルゴリズムを実装するための領域変数の拡張について述べる．5章では2変数等式制約のためのAC-5アルゴリズムを適用した制約伝播器について述べる．6章ではハイブリッド・アルゴリズムを適用したn変数等式制約の制約伝播器について述べる．7章では実験結果を示し，ハイブリッド・アルゴリズムの評価を行う．

2. 準　　備

2.1 CLP（FD）

簡単のため，次のCLP（FD）プログラムを用いて説明する．

```
test(X,Y):-          ……(1)
    [X,Y] in 1..5,   ……(2)
    X #= Y+1,        ……(3)
    labeling([X,Y]). ……(4)
```

この簡単なプログラムを実行するだけで次に示す解を得ることができる．

☆1 CLP（FD）における制約充足アルゴリズムは，一般に汎用的かつ効率的なことから制約伝播が主に利用されている．
☆2 3章で詳述する．
☆3 2章で詳述する．
☆4 最も計算量の少ないアルゴリズムとして知られている．2章で詳述する．
☆5 中間コードのことであるが，制約伝播を実行するためのものであることを明確にするためにこの用語を用いる．

$(X, Y) = \{(2,1),(3,2),(4,3),(5,4)\}$

一般に CLP（FD）プログラムは，制約，ラベリング，および Prolog のゴールから構成される．制約にはドメイン制約と算術制約がある☆．ドメイン制約では変数の領域を指定する．たとえば上のプログラムの (2) は変数 X，Y の領域を 1 から 5 の範囲（以下，1..5）に指定している．算術制約には等式制約，不等式制約，非等式制約があり，関係演算子の前に # を付けて制約であることを示す．たとえば上の (3) は等式制約を表す．ラベリングは変数をその領域の要素に順次具象化し，解を探索するためのものである（(4)）．

2.2 制約伝播アルゴリズム

制約伝播には，いつ，どの程度の領域縮小を行うかにより，様々なアルゴリズムが存在する．ここでは，本研究と関係の深い，部分ルックアヘッド・アルゴリズムおよび完全ルックアヘッド・アルゴリズムについて述べる．

2.2.1 部分ルックアヘッド・アルゴリズム

ある制約 $c + a_1X_1 + ... + a_nX_n\ R\ 0$（ここで c および a_i $(i = 1, ..., n)$ は任意の整数，X_i $(i = 1, ..., n)$ は領域変数，R は関係演算子）が次の条件 (1)〜(3) のいずれかを満足するとき，その制約を区間無矛盾性制約と呼ぶ．

(1) R が #= のとき，左辺の式の最小値および最大値がともに 0 と等しい．
(2) R が #>（#>=）のとき，左辺の式の最小値が 0 より大きい（0 以上である）．
(3) R が #<（#=<）のとき，左辺の式の最大値が 0 より小さい（0 以下である）．

部分ルックアヘッド・アルゴリズムは区間無矛盾性を維持するためのものである．たとえば制約 X#=Y+1（X ∈ 2..5，Y ∈ 1..4）において X の最大値が 3 に更新された場合，Y の最大値が 2 に更新される．ただし境界値以外の要素が除去されても何も起こらない．

非等式制約は，一般に制約内の変数が 1 つを除いてすべて具象化されるまで評価が遅延される．

2.2.2 完全ルックアヘッド・アルゴリズム

ある等式制約が次の条件を満足するとき，その制約をアーク無矛盾性制約と呼ぶ．

(1) 制約内の各変数の領域の各要素に対し，残りのすべての変数の領域内にその制約を満足する要素が 1 つ存在する．

完全ルックアヘッド・アルゴリズムはアーク無矛盾性を維持するためのものである．たとえば制約 X#=Y+1（X ∈ 2..5，Y ∈ 1..4）において X の領域から 3 が除去された場合，Y の領域から 2 が除去される．

完全ルックアヘッド・アルゴリズムは変数の領域の全要素を参照する必要があるため，変数を 3 つ以上含む制約に対しては組合せの爆発を引き起こす可能性が非常に大きい．したがって，一般にこのアルゴリズムは 2 変数等式制約に対してのみ適用される．

不等式制約および非等式制約の処理は，部分ルックアヘッド・アルゴリズムと同じである．

2.3 AC-5 アルゴリズム

AC-5 アルゴリズムは Hentenryck らが提案した[4]，完全ルックアヘッド・アルゴリズムの 1 つである．

このアルゴリズムは 2 変数等式制約に対して次の 2 つのステップを実行する．ステップ 1) 制約内の 2 つの変数の領域の全要素の中から，もう一方の変数の領域内に，それと対になって制約を満たす要素を持たないものを除去し，制約をアーク無矛盾性制約にする．ステップ 2) 領域からある要素が除去されると，その要素が支持していた要素をもう一方の変数の領域から除去し，アーク無矛盾性を維持する．ステップ 1 は制約の生成時に 1 度だけ実行され，ステップ 2 は領域から要素が除去されるたびに繰り返し実行される．**表 1** は 2 変数等式制約 X#=Y+1（X, Y ∈ 1..5）において，あるイベントが発生したときの X および Y の領域の状態変化を表す．まず制約の生成時にステップ 1 が実行され，X および Y の領域が縮小される（状態 A）．そして X の領域から 3 が除去されると，ステップ 2 が実行され，それが支持していた Y の要素 2 が Y の領域から除去される（状態 B）．同様に Y の領域から 4 が除去されると，ステップ 2 が再び実行され，X の領域から 5 が除去される（状態 C）．

表 1 X#=Y+1（X, Y ∈ 1..5）の状態変化
Table 1 Changes in condition of X#=Y+1 (X, Y ∈ 1..5).

状態	イベント	AC-5	X の領域	Y の領域
初期			1..5	1..5
A	生成時	ステップ 1	2..5	1..4
B	$X \neq 3$	ステップ 2	2, 4, 5	1, 3, 4
C	$Y \neq 4$	ステップ 2	2, 4	1, 3

3. 領域変数および遅延節

この章では，B-Prolog の遅延機構を構成する領域変数および遅延節について述べる．

3.1 領域変数

領域変数は有限領域を持つ変数である．領域変数に関する情報として，領域の最小値（min），最大値（max），領域の全要素を格納するためのビットベクト

☆ 処理系によっては他に最適化のための記号制約等がある．

ル（elms）等があり，各領域変数がこれらの情報を格納するフィールドを持つ．通常，領域は最小値と最大値の組，すなわち min..max，として表現される．しかし 1 度領域内に “穴” が発生し領域が不連続になると，領域の表現は elms に切り替わる．

領域に含まれる要素が 1 つになると，その領域変数はその値に具象化される．領域が空になると失敗となる．

領域変数に対する基礎述語として次のものがある．

- `dvar(+X)`☆：X が領域変数であるかどうかを検証する．
- `fd_min_max(+X,-Min,-Max)`：領域変数 X の最小値（Min）および最大値（Max）を取り出す．
- `exclude(+X,+Elm)`：領域変数 X の領域から要素 Elm を削除する．

その他の基礎述語に関しては文献 8) を参照されたい．

3.2 遅 延 節

遅延節は一般に次のような記述形式をとる．

```
delay ヘッド : -
        条件部 : {トリガ部} アクション部.
```

ヘッドが示す述語の呼び出しに対して，その呼び出しがヘッドと一致し，条件部が満足される場合，その呼び出しはアクション部☆☆を実行した後に遅延される．ヘッドと一致しない，あるいは条件部が満たされない場合は次の節が試される．アクション部で失敗が生じると元の呼び出しも失敗となる．トリガ部は遅延された呼び出しの再実行を引き起こすトリガを指定する．トリガには次の 4 種類がある．`ins(X)` は X が具象化されたとき，`min(X)` は X の領域の最小値が更新されたとき，`max(X)` は最大値が更新されたとき，そして，`dom(X)` は領域の中間の要素が削除されたとき，に発火し再実行を引き起こす．

遅延節はイベント駆動方式で実行される．処理系は各述語呼び出しの出入り口において発火したトリガがないかを調べる．もしあった場合，現在の実行は中断され，発火したトリガと関連する遅延された呼び出しの再実行を行う．

例として次のプログラムを考える．

```
delay freeze(X,Goal):- ……(1)
    var(X) :           ……(2)
    {ins(X)}.          ……(3)
freeze(X,Goal):-       ……(4)
    true : call(Goal). ……(5)
```

`freeze(X, Goal)` は意味的には `Goal` と同じであるが，その評価は X が具象化されるまで遅延される．この述語への呼び出しは X が変数のとき遅延される（(2)）．X が具象化すると，トリガ `ins(X)`（(3)）により遅延された呼び出しの再実行が起こる．そして `var(X)`（(2)）が失敗した後に 2 番目の節が実行され，`Goal` が呼び出される．2 番目の節は照合節と呼ばれ，実行のための条件は遅延節と同じであるが，アクション部を実行した後に遅延されることはない．

4. 領域変数の拡張

AC-5 アルゴリズムのステップ 2 を実行するためには，どの要素が領域から削除されたかを知る必要がある．そこで筆者らは，領域変数に新たなフィールドを追加し，その領域から削除された要素のリスト（以下，被削除要素リスト）をそのフィールドに格納するように領域変数を拡張した．ただし境界値の更新により削除された要素は含まない．これは，効率化のために，アーク無矛盾性の検査の前に区間無矛盾性を検査するからである．領域の要素数が大きく異なる場合，たとえば制約 `X#=Y+1`（$X \in 6..10, Y \in 1..20$）において，先に区間無矛盾性を検査すると Y の領域が 5..9 となり，全体の計算量が少なくなる☆☆☆．

新たなフィールドの導入にともない，次の 2 つの基礎述語を導入した．

- `fd_delta(+X,-DeltaX)`：処理系に X の被削除要素の記録の開始を告げるための述語である．X は領域変数，DeltaX は X の被削除要素リストを参照するための識別子である．
- `fd_delta_elms(+DeltaX,-Elms)`：DeltaX が指し示す被削除要素リスト Elms を取り出すための述語である．被削除要素リストはこの述語で参照されるたびに，空に初期化される．

例として以下のプログラムを考える．

```
testDelta:-                    ……(1)
    X in 1..10,                ……(2)
    fd_delta(X,DeltaX),        ……(3)
    X #\= 4,                   ……(4)
    fd_delta_elms(DeltaX,A),   ……(5)
    X #\= 5,                   ……(6)
    X #=< 8,                   ……(7)
    fd_delta_elms(DeltaX,B),   ……(8)
    write(A),write(B).         ……(9)
```

☆ + は入力用の引数を，- は出力用の引数を表す．

☆☆ ない場合は省略できる．

☆☆☆ アーク無矛盾性検査のみの場合は要素の総参照回数が 20 であるのに対し，先に区間無矛盾性を検査する場合は，区間無矛盾性検査時の参照回数も含めて，総参照回数は 12 である．

X は領域変数でその領域は 1..10 である（(2)）．基礎述語 fd_delta/2 により処理系は X の被削除要素の記録を開始する（(3)）．4 が X の領域から除去されると（(4)），基礎述語 fd_delta_elms/2 により獲得される被削除要素リストは [4] となる（(5)）．次に 5，9，10 が削除されるが（(6)，(7)），このときの被削除要素リストは [5] となる（(8)）．9 および 10 は領域の最大値の更新により削除された要素なのでリストには含まれない．

5. 2 変数等式制約の制約伝播器

2 変数等式制約の制約伝播器は**表 2** に示す 6 種類である．係数を正に限定しているのは区間無矛盾正検査の効率化のためであり，係数の有無により 3 つに分類しているのは最適化のためである．この章では，まず AC-5 アルゴリズムを適用した 2 変数等式制約 aX#=bY+c のための制約伝播器について述べ，次に表 2 で 2 類，3 類に分類される伝播器に対する最適化について述べる．

5.1 aX#=bY+c の制約伝播器

図 2 に制約 aX#=bY+c の伝播器を示す．まず AC-5 アルゴリズムのステップ 1 を実行し，制約をアーク無矛盾性制約にする（(2)）．その後に X および Y の被削除要素の記録を開始する（(3)，(4)）．述語 'aX=bY+c_propagate'/6 は AC-5 アルゴリズムのステップ 2 を実行する．(5) は X が更新されたときに Y の領域を縮小し，(7) は Y が更新されたときに X の領域を縮小する．

図 3☆に述語 'aX=bY+c_propagate'/6 の定義を示す．最初の節が遅延節で残りは照合節である．この述語への呼び出しは X および Y がともに領域変数の場合に遅延され（(2)），X に対して何かしらの更新が行われると宣言されたトリガにより再実行が引き起こされる（(3)）．アクション部では，まずドメイン制約 in/3 により制約を区間無矛盾性制約にする（(5)）．そして fd_delta_elms/2 により X の被削除要素リストを取り出し（(6)），そのリストを用いて制約をアーク無矛盾性制約にする（(7)）．(13)－(17) がそのための述語の定義である．X あるいは Y が具象化すると，この制約は代入となる（(8)－(11)）．

5.2 他の制約伝播器に対する最適化

表 2 で 3 類に分類される制約伝播器において，X（Y）の領域に穴がなければ，図 2 の (2) の呼び出しで Y（X）の領域の縮小には区間無矛盾性のみを検査すればよい．領域に穴があるかどうかは，次の基礎述語を用いれば分かる．

- dvar_bv(+X)：X の領域の表現がビットベクトルであるかを検査する．

同様に，2 類に分類される制約伝播器において，X の領域に穴がなければ Y の領域の縮小には区間無矛盾性のみを検査すればよい．

表 2 2 変数等式制約の制約伝播器
Table 2 Propagators for binary equality constraints.

1 類	2 類	3 類
'aX=bY+c'	'X=bY+c'	'X=Y+c'
'aX+bY=c'	'X+bY=c'	'X+Y=c'

X，Y：領域変数
a，b，c：任意の整数（ただし，a > 0, b > 0）

6. n 変数等式制約の制約伝播器

この章では，まず周による n 変数等式制約の制約伝播器について述べ，次にハイブリッドアルゴリズムを適用した制約伝播器についてのべる．ただし，制約はすべて標準形の形式をしているものとする．

- 標準形：$c + a_1X_1 + a_2X_2 + ... + a_nX_n \ \# = \ 0$. ここで c および a_i $(i = 1, ..., n)$ は任意の整数，X_i $(i = 1, ..., n)$ は領域変数である．

6.1 周による制約伝播器

周が実装した CLP（FD）処理系では n 変数等式制約は**図 4** に示す制約伝播器に変換される[10]．インラインテスト no_vars_gt(n,m) は，引数の後ろ n 個中に変数が m+1 個以上あれば成功する．したがって，この述語への呼び出しはその引数中に領域変数が 1 つでも含まれていれば遅延される（(2)）．そしてすべての領域変数が具象化されるまで区間無矛盾性を検査し，領域を縮小する（(4)）．領域の縮小には変数の数によらず同一のコードを使用する．したがって領域縮小の効率は良くないが，制御の切替えのオーバヘッドが少なく済む．

6.2 ハイブリッド・アルゴリズムを適用した制約伝播器

上述のように，周による制約伝播器では制約中の変数がすべて具象化されるまで区間無矛盾性検査を用いて領域を縮小する．一方，ハイブリッド・アルゴリズムでは，制約内に変数が 3 つ以上含まれる場合は区間無矛盾性を検査し，制約内の変数が 2 つになると，区間無矛盾性検査だけでなく，アーク無矛盾性も検査する．ハイブリッド・アルゴリズムを適用した n 変数等式制約の伝播器は**図 5** に示すようになる．制約内の変数が

☆ 図中，/>，/<はそれぞれ切り上げ割り算，切り捨て割り算を表す．

```
'aX=bY+c'(A,X,B,Y,C):-                                    ……(1)
      'aX=bY+c_arc_consistent'(A,X,B,Y,C),                ……(2)
      fd_delta(X,DeltaX),                                 ……(3)
      fd_delta(Y,DeltaY),                                 ……(4)
      'aX=bY+c_propagate'(A,X,B,Y,C,DeltaX),              ……(5)
      NewC is -C,                                         ……(6)
      'aX=bY+c_propagate'(B,Y,A,X,NewC,DeltaY).           ……(7)
```

図 2 'aX=bY+c' の制約伝播器
Fig. 2 The propagator for aX=bY+c.

```
delay 'aX=bY+c_propagate'(A,X,B,Y,C,DeltaX):-             ……(1)
      dvar(X),dvar(Y) :                                   ……(2)
      {ins(X),min(X),max(X),dom(X)},                      ……(3)
      fd_min_max(X,MinX,MaxX),                            ……(4)
      Y in (A*MinX-C)/>B..(A*MaxX-C)/<B,                  ……(5)
      fd_delta_elms(DeltaX,ElmsX),                        ……(6)
      'aX=bY+c_maintain_ac'(A,X,B,Y,C,ElmsX).             ……(7)
'aX=bY+c_propagate'(A,X,B,Y,C,DeltaX):-                   ……(8)
      dvar(X) : X is (B*Y+C)//A.                          ……(9)
'aX=bY+c_propagate'(A,X,B,Y,C,DeltaX):-                   ……(10)
      true :  Y is (A*X-C)//B.                            ……(11)
                                                          ……(12)
'aX=bY+c_maintain_ac'(A,X,B,Y,C,[]).                      ……(13)
'aX=bY+c_maintain_ac'(A,X,B,Y,C,[E|Elms]):-               ……(14)
      Y1 is (A*E-C)//B,                                   ……(15)
      exclude(Y,Y1),                                      ……(16)
      'aX=bY+c_maintain_ac'(A,X,B,Y,C,Elms).              ……(17)
```

図 3 述語 'aX=bY+c_propagate'/6 の定義
Fig. 3 The definition of the predicate 'aX=bY+c_propagate'/6.

```
delay c(C,A1,A2,...,An,X1,X2,..,Xn):-                     ……(1)
       no_vars_gt(n,0) :                                  ……(2)
       {ins(X1),min(X1),max(X1),ins(X2),...},             ……(3)
       領域 X1,X2,..,Xn の区間無矛盾性検査.                ……(4)
c(C,A1,A2,...,An,X1,X2,..,Xn):-                           ……(5)
       true : 制約のテスト.                               ……(6)
```

図 4 周による n 変数等式制約の制約伝播器
Fig. 4 Constraint propagator for a n-ary equality constraint implemented by Zhou.

3 つ以上のときは（(2)），区間無矛盾性検査により領域を縮小する（(4)）．制約内の変数が 2 つ以下になると 2 番目の節（(5)−(8)）を実行する．組み込み述語 nary_to_binary/6 は，ヘッドの引数を走査し，制約を 2 変数等式制約の形式（NewC+B1*Y1+B2*Y2=0）に変換する（(7)）．そして call_bc_propagator/5 でその 2 変数等式制約に適した制約伝播器（表 2 参照）を呼び出す（(8)）．

7. 性能評価

7.1 2 つ CLP（FD）処理系の比較

表 3 はいくつかの伝統的なベンチマーク・プログラムを実行するのに必要な CPU 時間ならびにバックトラックの回数を，2 つの CLP（FD）処理系で計測した

```
delay c(C,A1,A2,...,An,X1,X2,..,Xn):-                    ……(1)
      no_vars_gt(n,2) :                                  ……(2)
      {ins(X1),min(X1),max(X1),ins(X2),...},             ……(3)
      領域 X1,X2,..,Xn の区間無矛盾性検査.                ……(4)
c(C,A1,A2,...,An,X1,X2,..,Xn):-                          ……(5)
      true :                                             ……(6)
      nary_to_binary(n,NewC,B1,Y1,B2,Y2),                ……(7)
      call_bc_propagator(NewC,B1,Y1,B2,Y2).              ……(8)
```

図 5 ハイブリッド・アルゴリズムを適用した n 変数等式制約の制約伝播器
Fig. 5 Constraint propagator for a n-ary equality constraint with the hybrid algorithm.

表 3 2 つの CLP（FD）処理系の比較
Table 3 Comparison of two CLP (FD) systems.

	CPU 時間（SPARC-10，ミリ秒）			バックトラックの回数		
	bp_interval	bp_hybrid	$\frac{\text{bp\_interval}}{\text{bp\_hybrid}}$	bp_interval	bp_hybrid	$\frac{\text{bp\_interval}}{\text{bp\_hybrid}}$
alpha	14966	8183	1.8	8440	4605	1.8
alpha_ff	67	66	1.0	44	44	1.0
crypta	67	67	1.0	52	52	1.0
eq20	167	166	1.0	49	49	1.0
magic4	17	17	1.0	18	18	1.0
8_queens	17	17	1.0	18	23	0.8
12_queens	67	50	1.3	91	39	2.3
24_queens	467	50	9.3	403	7	57.6
36_queens	425450	117	3636.2	407815	21	19419.8
64_queens	-	267	-	-	3	-
100_queens	-	60433	-	-	22293	-

bp_interval：周による CLP（FD）処理系
bp_hybrid：本論文による CLP（FD）処理系

結果である．bp_interval は周が実装した CLP（FD）処理系で，制約伝播に部分ルックアヘッド・アルゴリズムを適用している．bp_hybrid は，本論文で実装した CLP（FD）処理系で，制約伝播にハイブリッド・アルゴリズムを適用している．alpha では bp_hybrid は bp_interval より約 2 倍速い．線形空間 N クィーンプログラム[6]（N_queens）では格段に速く，探索空間が大きくなるに従いその差は広がっている．残りのプログラムでは bp_hybrid は bp_interval とほぼ同等の性能を示している．この結果から完全ルックアヘッド・アルゴリズムの計算量が非常に少ないことが分かる☆．したがって，一般的に bp_hybrid は bp_interval より優れているといえる．

表 4 は DJ プログラムにおける 2 つの CLP（FD）処理系の比較の結果である．DJ[9),11)] は有限領域上の制約プログラミングを可能にした Java の拡張言語で，制約の処理を B-Prolog 上の制約変換系に依存している．実験に用いたプログラムの説明を以下に示す．

- Circles1，Circles2：幾何学模様を描画するプログラム．
- Japan，UK，USA：各国の国旗を描画するプログラム．
- Marriage：安定結婚問題を解き，結果をグラフィカルに表示するプログラム．
- SendMory：SendMoreMoney 問題を解き，結果をグラフィカルに表示するプログラム．

表 4 DJ プログラムを用いた 2 つの CLP（FD）処理系の比較（SPARC-10，ミリ秒）
Table 4 Comparison of two CLP (FD) systems for DJ programs (SPARC-10, milliseconds).

	bp_interval	bp_hybrid	$\frac{\text{bp\_interval}}{\text{bp\_hybrid}}$
Circles1	15616	3083	5.1
Circles2	2717	333	8.2
Japan	317	133	2.4
UK	185984	24850	7.5
USA	14983	6767	2.2
Marriage	23667	1300	18.2
SendMory	44133	1450	30.4

bp_interval：周による CLP（FD）処理系
bp_hybrid：本論文による CLP（FD）処理系

☆ これは次の 2 つの要因によるものである．1）制約が 2 変数制約になってから被削除要素を登録し始める．2）領域の境界値の更新により削除された要素は登録せず，区間無矛盾性の検査を併用する．

表 5 2 つの処理系の比較（SPARC-10，ミリ秒）
Table 5 Comparison of two systems (SPARC-10, milliseconds).

	bp_hybrid	gprolog	$\frac{\text{gprolog}}{\text{bp\_hybrid}}$
alpha	8183	4710	0.6
crypta	67	50	0.7
eq10	50	80	1.6
eq20	166	140	0.8
magic4	17	0	-
8_queens	17	30	1.8
12_queens	50	310	6.2

bp_hybrid：本論文による CLP（FD）処理系
gprolog：GNU Prolog の CLP（FD）処理系

これらのプログラムにはグラフィカル・コンポーネントを配置するための制約が多く含まれており，探索空間の大きな問題である．これらのプログラムに対して，bp_hybrid は bp_interval より非常に優れた性能を示している．特に SendMory では 30 倍ほども速い．

7.2 本論文による CLP（FD）処理系と GNU Prolog の CLP（FD）処理系との比較

表 5 は本論文による CLP（FD）処理系と GNU Prolog（バージョン 1.0）[2),3)] の CLP（FD）処理系（gprolog）との比較の結果である．2 つの処理系は大きく異なり，B-Prolog がエミュレータベースであるのに対し，GNU Prolog はプログラムをネイティブコードにコンパイルする．また GNU Prolog の CLP（FD）処理系では，制約伝播に部分ルックアヘッド・アルゴリズムを適用している．

alpha では gprolog は bp_hybrid より 2 倍近く速いが，線形空間 N クィーンプログラムでは逆に bp_hybrid が gprolog より格段に速い．個々のプログラムでばらつきはあるものの，全体としてはほぼ同等の性能を示している．2 つの処理系の実装法の違いを考慮すれば，この結果は良いものであるといえる．

8. おわりに

本論文では，探索空間の大きな問題に対処するためのハイブリッド・アルゴリズムを提案し，その有効性を示した．従来の実装法と違い，記述力の高い中間言語を用いたため，この問題に容易に対処できた．同様の手法により，今後，様々な対象領域の制約処理系を実現できる可能性もある．

また，この手法は工学的な問題に対しても適用できると考えられ，機械設計や建築の間取問題等への応用を検討中である．

参考文献

1) Aggoun, A. and Beldiceanu, N.: Overview of the CHIP Compiler System, *Proc. 8th International Conference on Logic Programming*, pp.775–789, MIT Press (1991).
2) Codognet, P. and Diaz, D.: Compiling Constraints in clp (FD), *Journal of Logic Programming*, Vol.27, No.3, pp.185–226 (1996).
3) Diaz, D.: GNU Prolog Manual (Version 1.0) (1999).
4) Hentenryck, P.V., Deville, Y. and Teng, C.-M.: A Generic Arc-Consistency Algorithm and its Specializations, *Artificial Intelligence*, Vol.57, No.2-3, pp.291–321 (1992).
5) Meier, M.: Better Late Than Never, *Implementations of Logic Programming Systems* Tick, E. and Succi, G. (Eds.), Kluwer Academic Publishers (1994).
6) Puget, J.-F. and Leconte, M.: Beyond the Glass Box: Constraints as Objects, *Proc. International Symposium on Logic Programming*, pp.513–527, MIT Press (1995).
7) Tsang, E.: *FOUNDATIONS of CONSTRAINT SATISFACTION*, ACADEMIC PRESS (1993).
8) Zhou, N.-F.: B-Prolog User's Manual (Version 3.1) (1998). http://www.sci.brooklyn.cuny.edu/~zhou/bprolog.html.
9) Zhou, N.-F.: DJ User's Manual (Version 0.5) (1998). http://www.sci.brooklyn.cuny.edu/~zhou/dj.html.
10) Zhou, N.-F.: A High-Level Intermediate Language and Algorithm for Compiling Finite-Domain Constraints, *Proc. Joint International Conference and Symposium on Logic Programming*, pp.70–84, MIT Press (1998).
11) Zhou, N.-F., Kaneko, S. and Yamauchi, K.: DJ:A Java-based Constraint Language and System, 日本ソフトウェア科学会第 15 回大会論文集，pp.97–100 (1998).

(平成 12 年 1 月 21 日受付)
(平成 13 年 1 月 11 日採録)

兼子 聡介（学生会員）

1973 年生．1997 年九州工業大学情報工学部卒業．1999 年同大学院情報工学研究科博士前期課程修了．現在，九州工業大学大学院情報工学研究科博士後期課程在学中．制約論理型プログラミング言語処理系の研究開発に従事．

周　能法

1984 年南京大学計算機科学系卒業．1991 年九州大学情報工学研究科博士課程修了．1991 年九州工業大学情報工学部講師（機械システム工学科）．1993 年同学部助教授．現在，ニューヨーク市立大学バークレー校情報科学部助教授．情報工学博士．制約論理型プログラミング言語処理系の研究開発に従事．ACM，IEEE，ALP，人工知能学会，日本ソフトウェア科学会各会員．

長澤　勲（正会員）

1944 年生．1967 年九州大学工学部電子工学科卒業．1972 年同大学院工学研究科博士課程単位取得退学．1972 年九州大学中央計数施設講師．現在，九州工業大学情報工学部教授（機械システム工学科）．工学博士．知識情報処理の立場から CAD/CAM，ロボット，医療システム等の研究開発に従事．人工知能学会，日本建築学会，精密工学会，電子情報通信学会，日本機械学会，日本設計工学会，日本ロボット学会各会員．