人　工　知　能　74－1
（１９９１．　１．　１６）

制約論理型プログラミング言語における変数管理方式

碓崎　賢一
九州工業大学　情報工学部

近年，制約に基づく問題解決手法が注目され，制約論理型プログラミング言語の研究が活発に行われるようになってきている．制約は高い表現能力を持つが，その特長が広く利用されるようになるためには処理速度も向上する必要があり，制約評価機構の効率的な実現方式が要求される．本報告では，変数セルに付加した制約情報を用いた，論理型プログラミング言語における効率的な制約管理手法を提案する．この方式を用いた処理系を12-Queenによって性能評価した結果，従来の生成検査法による実行と比較して，7800倍の処理速度を実現できることを確認した．

Variable Management Metod for Constraint Logic Programming Languages

Ken'ichi KAKIZAKI
Faculty of Computer Science and Systems Engineering
Kyushu Institute of Technology

In recent years, problem solving technique based on constraints have been noticed, and constraint logic programming languages are researched actively. For high expression abilities of constraints to be used widely, the processing speed should be imploved. Therefore, an efficient achievement method for constraints is demanded. This paper proposes an efficient constraints management method for logic programming languages which uses constraints information added to variable cells. The system with this metod is evaluated in the performance by using 12-Queen problem. As the result, the executing speed of 12-queen problem using constraints is 7800 times faster than that of one using conventional generate-test method.

1．はじめに

近年，制約に基づく問題解決手法が注目され，特に，述語論理と制約との整合性が高いために，制約論理プログラミング言語の開発が活発に行われるようになってきている．制約を用いた問題解決法では，変数が取り得る値を制約条件として宣言的に記述し，制約評価機構を用いることにより，一連の制約を満たすような解を求める処理を行う．制約と述語論理はともに宣言的であるという特徴を持っており，両者を結合することによって問題解決法をより宣言的に記述できるようになるという利点がある．

制約評価機構はその機能に応じて，制約をどのような形式で表現し，制約の評価をいつ起動するかを定める制約管理機構と，制約が満たされているかどうか判断し，あるいは制約を満たす解が特定できるならばそれを求める制約解消機構に分けて考えることができる．制約に関する多くの研究は，制約の表現能力に着目し，後者の制約解消機構についての研究が進められている．しかしながら，制約の特長が広く利用されるようになるためには，表現能力が高いだけでなく処理速度も向上する必要があり，制約管理機構の効率的な実現方式が要求される．本報告では，このような要求を満たす方式として，変数セルに制約情報を付加する制約管理機構を提案し，その評価を示す．

2．受動的制約

制約はその解決能力によって，受動的制約と能動的制約に分類される．受動的制約評価機構は変数が具体化された場合に，その値が変数に付加された一連の制約を満たすかどうかを調べる機能を持ち，能動的制約評価機構は変数に付加された一連の制約を用いて，その変数が取り得る値を求める機能を持つ．例えば，X =< 5, X >= 5 という制約が付加されている場合には，受動的制約評価機構では，X = 5 に具体化する単一化のみ成功し，それ以外の値への具体化を試みる単一化は制約評価機構によって失敗させられる．一方，能動的制約評価機構では，制約評価機構によってX = 5 が自動的に求められる．このように，受動的制約と能動的制約では，能動的制約の方が高い問題解決能力を持つといえるが，本報告では制約管理機構に着目しているため，制約解消機能の実現手法に関しては，受動的な制約に限定して述べる．

受動的制約の効果の１つとして，PROLOG の基本特性である深さ優先の探索手法を必要に応じて変更し，解を効率よく求めることができるようになることがあげられる．従来の PROLOG 処理系では，深さ優先の探索手法しか利用できないために，基本的な問題解決手法の一つである生成検査型のプログラムの実行効率が悪いという問題がある．このため，生成器の中に検査器を埋め込むなどの本質的に不必要な調整を行ない，仮説の生成数を抑える必要があった．

制約評価機構を組み込んだ制約論理プログラミング言語では，検査器を制約で置き換えることができる．従来の検査器とは異なり，制約は生成器の実行の前に付加することができるために，ゴールの出現順序に依存しない，より宣言的な記述ができるようになっている．また，付加された制約の評価は，生成器によって値が具体化されるまで遅延され，変数が具体化された時点で即座に検査される．この動作は，生成器の中の適切な位置に検査器を埋め込み理想的に調整された問題解決器の動作に等しいものである．受動的制約を用いると，一般化された生成器をブラックボックスと見なして手を加える必要がなくなるとともに，生成器は不適切な仮説枝を生成せず最初から制約を満たす解を生成することになるために，大幅な性能の向上が実現される．

3．言語仕様

提案する実現方式を示すために，その前提となる制約論理型言語の言語仕様の概要を示す．言語仕様は，基本的に DEC-10 PROLOG のそれと同様であるが，制約を付加するための表現が加えられていることと，制約が付加されている変数が具体化される際に，制約の評価が行なわれるという点が異なっている．ここで示す制約は受動的なもので，制約式は評価するために必要十分な変数の具体化が行われた場合に評価され，制約評価を起動することになった単一化処理は，制約条件が満たされれば成功し，そうでなければ失敗する．

制約を変数に付加する式は，以下に示すように #{Constraints} の形式で表現する．

```
#{X < 5}
```

この例は，変数 X が 5 未満の値しか取れないという制約を表わしており，たとえば，次に示すような実行結果が得られる．

```
:- #{X < 5}, member(X, [1,2,3,4,5,6,7]),
        write(X),fail.
1234
```

変数に制約を付加する表現 #{Constraints} は宣言ではなく述語として実現されており，変数に付加された制約は #{Constraints} をゴールとする呼び出しよりも後のゴールの実行のみに影響を与え，それ以前のゴールの実行には影響しない．また，後戻りで制約が付加された以前に戻ると，制約は解除される．

本論文では，制約として表現できる制約式は以下の4種類とする．

1）データ型

integerp(X), floatp(X), numberp(X), symbolp(X), listp(X), structurep(X)

2）算術式比較

X =:= Y, X =¥= Y,
X < Y, X =< Y, X > Y, X >= Y,
X := Y

3）項順序比較

X @= Y, X @¥= Y,
X @< Y, X @=< Y, X @> Y, X @>= Y,
X == Y, X ¥== Y

4）要素

X + [...], X - [...]

データ型を表現する制約式は，指定されたデータ型であるかどうかが調べられる1つの変数を引数として持つ．変数が具体化された時点で，具体化された値が指定されたデータ型かどうかが調べられる．データ型が異なる場合には，制約式の評価を起動することになった変数の単一化が失敗する．

算術式比較を表現する制約式は，比較される算術式を表わす2つの引数を持つ．この制約が付加された場合には，2つの引数に含まれる変数がすべて具体化された時点で算術式として評価され，その結果得られた数値の比較が行われる．条件が満たされなかった場合には，制約式の評価を起動することになった変数の単一化が失敗する．

項順序比較を表現する制約式は，比較される項を表わす2つの引数を持つ．この制約が付加された場合には，2つの引数が具体化された時点で，言語で規定された項順序によって両者の比較が行われる．条件が満たされなかった場合には，制約式の評価を起動することになった変数の単一化が失敗する．

要素を表現する制約式は，項の集合を表わすリストと，その要素であるかどうかが調べられる変数の2つの引数を持つ．+ の式は具体化された項がリストの要素に含まれる場合に真となり，- の式はその逆の意味を持つ．変数が具体化された時点で，具体化された項が集合を表わすリストに含まれるかどうかが調べられる．条件が満たされなかった場合には，制約式の評価を起動することになった変数の単一化が失敗する．

以上に，使用することができる制約式を示したが，これらの制約は，1つの変数に対して複数付加することができる．また，制約が付加された変数どうしが単一化された場合には，単一化された変数は，それぞれの変数に付加されていたすべての制約を持つことになる．単一化処理によって変数の具体化が試みられる場合には，その変数に付加された一連の制約が満たされるかどうかが調べられる．具体化される値がすべての制約を満たす場合には，その単一化処理は成功し，そうでなければ失敗する．

制約を表す式は，プログラムの中に静的に記述されている必要はなく，次に示す例のように，実行時に動的に制約式を指定することができる．

```
test (C,X) :-
        #{C},
        execute(X).
:- test(X<5, X).
```

この例では，変数 X に付加された制約 X < 5 を満たす解を求める処理が execute/1 により実行される．

4．制約実現手法

4．1　記述言語

制約論理型プログラミング言語の実現方式として，PROLOG によってメタインタプリタ[1]を記述する方法が多用されている．この方式は，処理系の核部分を拡張するのと比較して作業量が少なく，PROLOG の高い記述性を利用して，さまざまな制約解消機構を実装しやすいという利点がある．しかしながら，PROLOG によって記述された制約評価機構は性能が低く，性能を改善するためには，プログラムを変更するたびにコンパイルしなおす必要がある．また，プログラムをコンパイルしても性能上の問題が残る．さらに，制約をまったく

用いないプログラムの実行も，処理速度の遅いメタインタプリタで実行されてしまうという問題がある．

このような問題が生じないようにするために，本報告で提案する方式は，CなどでPROLOG処理系の核部分を拡張することを想定している．この方式では，PROLOG処理系の核部分を変更する必要があるが，実用的な処理速度が得られるという利点がある．また，実際にシステムを構築した結果，変数に制約を付加する機構と制約解消を行なう機構を新たに追加するほかは，従来のシステムの単一化処理を行なうルーチンを部分的に変更するだけですみ，開発も容易に行なえることが明らかになっている．

4．2　制約集合の管理

制約論理型プログラミング言語の一般的なメタインタプリタ[2]を図1に示す．3番目の節に含まれるclause/3は，従来の2引数のほかに第3引数を持ち，節に付加されている制約を取得するために用いられる．constraints_solve/3は，第1引数には現在までの実行過程で得られた制約集合を，第2引数には新たに得られた制約集合を取り，これらの制約を評価して得られた新たな制約集合（制約の標準形）を第3引数に返す．制約が満たされない場合には，constraints_solve/3は失敗し後戻りが起きる．

```
solve([], C, C).
solve([Goal| Goals], PrevC, NewC) :-
        solve(Goal, PrevC, TempC),
        solve(Goals, TempC, NewC).
solve(Goal, PrevC, NewC) :-
        clause(Goal, Body, CurC),
        constraints_solve(PrevC, CurC, TempC),
        solve(Body, TempC, NewC).
```

図1　制約論理型言語のメタインタプリタ

このメタインタプリタでは，プログラムの実行過程で得られたすべての変数に関する制約を制約集合として一括管理する方式を取っている．この方式では，新たなゴールが呼び出されるたびに得られる新たな制約がそれ以前の制約集合に加えられ，新たに得られた制約集合が評価される．制約を評価する場合には，制約集合の中から評価可能な制約集合を選出する処理が必要になるために，制約集合の増加にしたがって処理速度が低下するという問題がある．また，制約集合に変化が生じたかどうかに関わらず，ゴールが呼び出されるたびに制約評価が行なわれるために，制約を使用しないプログラムにおけるオーバーヘッドが非常に大きくなるという問題がある．

本報告で提案する方式は，すべての変数に関する制約を一括して保持する制約集合を管理する方式ではなく，各変数に着目し，制約が付加された変数セルに対して個別に制約情報を記録していくものである．この方式では，ある変数が単一化された場合に，その変数に関係する制約情報だけを簡単に取り出すことができるために，制約評価を効率良く行なうことができる．また，制約と関係のない変数は，制約情報が付加されていないために簡単に判別することができる．このような情報を利用して，制約が付加されていない変数の単一化時には，制約評価機構を起動しないようにできるために，制約が使用されていないプログラムでは，制約評価機構のオーバーヘッドがほとんど無視できる程度に押さえられるという利点もある．

単一化処理に着目した場合，図1のメタインタプリタを利用する方式では，実行する必要のない単一化処理が多数実行され，性能を低下させる要因になるという問題がある．これは，clause/3が呼ばれた時点で，制約を考慮することなくゴールのすべての引数の単一化処理が行なわれ，その後に制約評価機構constraints_solve/3が呼び出され，単一化処理の結果が制約を満たしているかどうかが検査されるためである．このため，制約を満たさない単一化処理とその後の制約評価機構が呼び出されるまでの一連の単一化処理は共に無駄であるにもかかわらず実行されてしまう．変数セルに制約情報を付加する方式では，単一化処理を行なう時点で制約が満たされているかどうかを効率良く検査し，制約を満たさない単一化を即座に失敗させることができるために，無駄な単一化処理が全く行なわれず効率がよいという利点がある．

4．3　制約情報セル

本報告で提案する方式は，制約情報を含むセルを変数に付加することによって制約の管理を行な

う．制約情報を含むセルは，以下の３種類に分類され，これらを総称して制約情報セルと呼ぶことにする．

１）制約式セル
２）制約リンクセル
３）制約変数セル

これらのセルは，ヒープスタック上に作成される．なお，以下の説明はWAMアーキテクチャ[3]を想定している．

制約式セルは変数に付加される制約式と，その関連情報を記録するために使用される．制約リンクセルは，１つの変数に対して複数の制約が付加された場合に，それらの制約を記録した制約式セルを連結し，一連の制約をまとめるために使用される．制約リンクセルは，link と expression の２つのセルを持ち，link には，複数の制約がある場合に次の制約リンクセルを指すポインタが格納され，expression には，その制約リンクセルが参照する制約式セルを指すポインタが格納される．制約式セルと制約リンクセルが分離されているのは，制約式セルがその制約式に含まれる複数の変数で共有される可能性があるためである．制約変数セルは，変数セルから制約リンクセルを参照できるようにするために使用される．制約リンクセルは，value と link の２つのセルを持ち，value には制約が付加された変数の変数値が格納され，link にはこの変数に付加された制約リンクセルを指すポインタが格納される．ただし，value セルに格納されるポインタには例外があり，制約が付加された変数どうしを単一化した場合には，他の制約変数セルを指すポインタが格納される．３種類の制約情報セルの関係の概略を図２に示す．

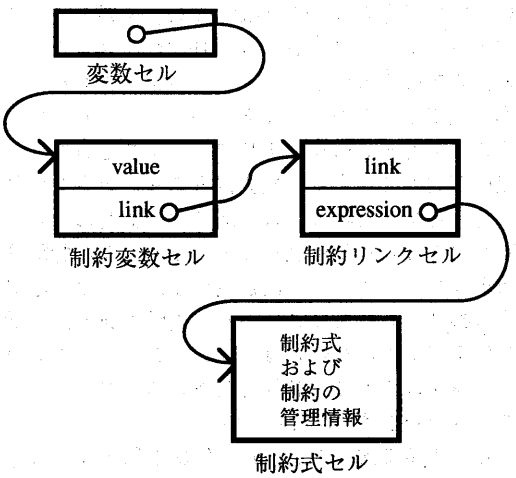

図２　制約情報セル

４．４　制約の付加

変数への制約の付加は，述語として実現されている #{Constraints} の形式で行なう．制約を付加する時点ですでに変数が具体化されており評価可能な場合には，制約式として与えられた式が通常の述語として呼び出された場合の処理と同様な処理が行なわれ，#{Constraints} は単に成功するか失敗する．

変数に対して制約を付加する最初の処理として，制約式セルの生成と制約リンクセルの生成が行なわれる．制約式に含まれる変数が１つの場合には，２種類の制約情報セルは１つずつ生成され，制約リンクセルの expression セルには制約式セルを指すポインタが設定される．制約式に含まれる変数が複数ある場合には，図３に示すように，制約式セルは１つだけ生成されるが，制約リンクセルは制約式に含まれる変数の種類と同じ個数だけ生成され，各制約リンクセルの expression セルには１つだけ生成された制約式セルを指すポインタが設定される．

制約式セルには，制約式の構造が記録され，制約解消機構によって評価できるようになっている．制約は，単一化処理が行なわれるたびに頻繁に評価されるために，できるだけ評価のコストが小さくなるように考慮する必要がある．このため，制約式セルを生成する際に，部分評価法を用いた最

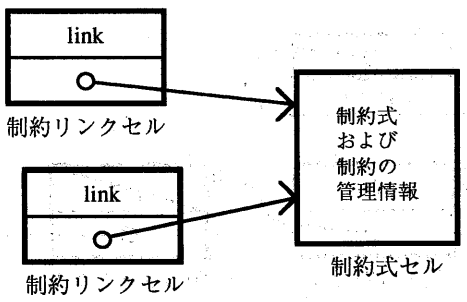

図３　制約リンクセルと制約式セル

適化を行なう．3章に示したように，制約式として算術式を使用することができるが，式の中で評価可能な部分がある場合には，その部分を評価して得られた式を制約式セルに記録する．例えば，次の1に示すような制約があり，この制約を付加する時点でA＝10に具体化されていた場合には，2のような制約式が制約式セルに記録される．

1）#{X =:= Y + A * 5}

2）X =:= Y : 50

制約式セルと制約リンクセルの生成が終了すると，次に，これらのセルを変数に関係付ける処理が行なわれる．この処理は，変数が初めて制約を付加される場合と，すでに制約を付加されている場合で異なる．

初めて制約が付加される変数の場合には，図4に示すように，制約リンクセルを変数に関係付けるセルとして制約変数セルが生成される．制約変数セルのlinkセルには，制約リンクセルを指すポインタが設定され，制約が付加される変数セルには，制約変数セルを参照するようにポインタが設定される．また，変数セルのアドレスは，後戻りに備えるためにトレイルスタックに記録される．さらに，制約変数セルのvalueセルは変数セルとして初期化され，以後の処理で変数セルとして利用される．なお，制約変数セルを参照するポインタにはタグが付加されており，このタグは変数のデリファレンスに利用される．

制約が付加されている変数に新たな制約を付加する場合には，制約変数セルはすでに存在しているので新たに生成されることはない．この場合には，すでに存在している制約変数セルのlinkセルを参照して，すでに付加されている制約を連結している制約リンクセルをたどる処理が行なわれる．

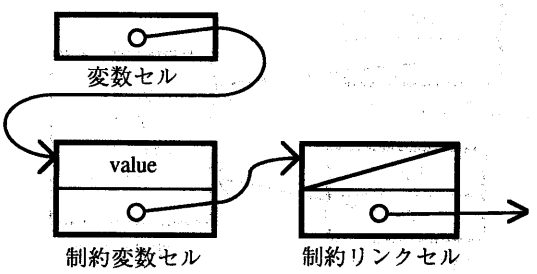

図4　最初の制約の付加

制約変数セルからたどることができる最後の制約リンクセルに到達したならば，図5の破線で示すように，そのセルのlinkセルには新たな制約に関係付けられた制約リンクセルを指すポインタが設定される．また，linkセルのアドレスは，後戻り処理に備えてトレイルスタックに保存される．図5の例では，この操作の結果，変数に2つの制約が付加されている．

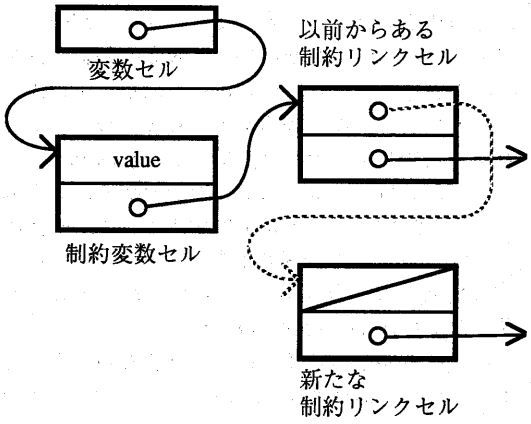

図5　2つめ以降の制約の付加

4．5　制約変数セルのデリファレンス

単一化処理で必要とされるデリファレンス処理について説明する．PROLOGでは，変数の値を参照する際にデリファレンスと呼ばれる処理を行ない，変数セルのポインタをたどる処理を行なう．本方式では，このような従来のデリファレンスと共に，制約変数セルのデリファレンスを行なう．後ほど改めて示すが，制約付きの変数どうしの単一化を行なった場合には，図7の制約変数セルAに示すように制約変数セルのvalueセルに制約変数セルへのポインタが設定される．このような設定は繰り返し行なわれている可能性があり，変数の値を取得するためには，従来の変数セルのデリファレンスの他に，制約変数セルのデリファレンスを行なわなければならない．

4．4節で述べたように，制約変数セルへのポインタにはタグが付加されており，他のセルへのポインタと区別できるようになっている．デリファレンスを行なう場合には，まず従来の変数のデ

リファレンスを行ない，次に制約変数セルのデリファレンスを行なう．

4．6　制約付き変数の単一化処理

制約が付加されている変数の単一化処理は，以下の3種類に分類される．

1）制約付きの変数と通常の変数
2）制約付きの変数と制約付きの変数
3）制約付きの変数と定数

これらに分類されない通常の単一化処理は，従来の方法で行なわれる．

4．6．1　通常の変数との単一化処理

通常の変数と制約付きの変数の単一化処理では，図6の破線で示すように，制約付きの変数に付加されている制約変数セルを指すポインタが通常の変数のセルに設定される．また，変数セルのアドレスは，後戻りに備えるためにトレイルスタックに記録される．この操作によって，制約変数セルに格納されている変数値と制約が共有されることになる．

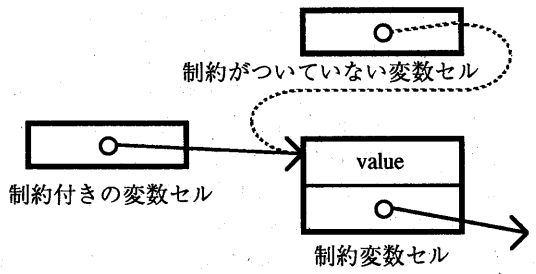

図6　制約のない変数との単一化

4．6．2　制約付き変数との単一化処理

制約付きの変数どうしの単一化処理では，まず，変数値を共有するための処理が行なわれる．2つの変数は，それぞれの制約変数セルのvalueセルが変数として利用されており，図7のポインタ1で示されるように，変数値の共有は一方のvalueセルに他方の制約変数セルを指すポインタを設定することにより行なう．また，ポインタが設定されたvalueセル（制約変数セルAのvalueセル）のアドレスは，後戻りに備えるためにトレイルスタックに記録される．4．5節で述べた制約変数セルのデリファレンスは，このようにして設定された制約変数セルへのポインタをデリファレンスし，変数値を求めるために行なわれる．

次に，2つの変数それぞれに付加されていた制約を結合し，単一化された変数の制約として参照されるようにする必要がある．この処理では，まず，変数を格納するセルとして利用される制約変数セル（制約変数セルB）から制約リンクセルをたどり，最後の制約リンクセル（制約リンクセルB）を取得する．次に，図7のポインタ2で示されるように，この制約リンクセルのlinkセルに，もう一方の制約変数セル（制約変数セルA）に付加されている制約リンクセル（制約変数セルA）を指すポインタを設定する．この場合も，後戻りに備えるために，linkセル（制約リンクセルB）のアドレスがトレイルスタックに記録される．

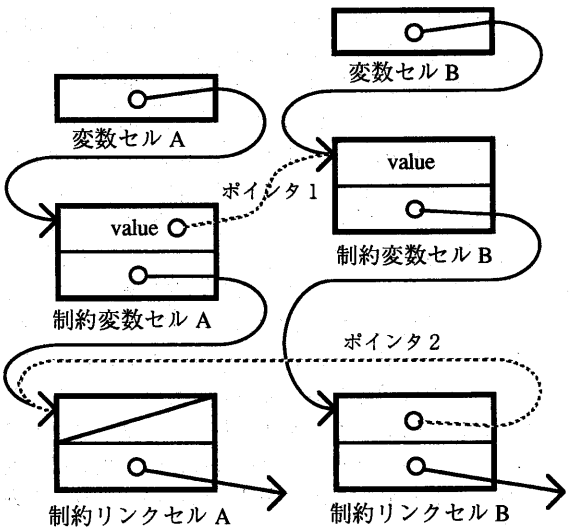

図7　制約付きの変数との単一化

このような操作の結果，制約変数セルBから制約リンクセルをたどることによって，2つの変数に付加されていたすべての制約を参照することができるようになる．一方，変数セルAから制約変数セルをたどった場合，制約変数セルAと変数制約セルBの2つのセルから制約リンクセルを取得することができる．しかしながら，制約変数セル

A から取得できる制約リンクセルは制約変数セル B から取得できるそれに含まれており，無視しなければ効率が低下することになる．本方式では，変数値が格納される制約変数セル（制約変数セル B）から参照される制約以外は無視し利用されないようにしている．

4．6．3　定数との単一化処理

制約付きの変数と定数の単一化処理では，まず，定数値を制約変数セルの value セルに設定する．この場合も，後戻りに備えるために value セルのアドレスをトレイルスタックに記録する．これらの処理が終了すると，制約解消機構を呼び出す．

4．7　制約解消

制約が付加されている変数が具体化された場合には，制約解消機構が呼び出される．本報告では，制約解消機構の機能を受動的制約に限定しているので，具体化した値が制約を満たすかどうかを検査する処理が行なわれる．制約の評価は，制約変数セルから制約リンクをたどって得られる順番で行なわれる．制約は，後から追加されたものが制約リンクの最後に付け加えられるために，先に付加された制約から評価される．制約を付加する際には，重要な制約ほど早期に付加されると考えられるので，この評価順序は適切なものであると考えられる．

3章で示した制約式のほとんどは組み込み述語として用意されているものと同じ機能を持つため，制約解消機構は，これらの組み込み述語の処理ルーチンを直接呼び出して検査を行なう．この方法では，制約式の機能を述語として呼び出す処理のオーバーヘッドが生じないため，制約の評価が頻繁に行なわれても効率が低下しないという利点がある．

本報告では詳細を示さないが，能動的な制約解消機構を実現する場合には，制約付きの変数どうしの単一化を行なった場合にも，制約を評価し新たな標準形を求める制約解消機構を呼び出すようにする．また，制約の評価によって制約の標準形を求める操作を行なうために，制約集合を書き換える必要がある．この処理では，以下の2つの操作が必要になる．

1）新たな制約（標準形）の付加
2）不必要な制約の消去

1の操作は，新たな制約を制約リンクセルの最後に追加することで実現できる．また，2の操作は，制約式セルの中に，その制約式が有効かどうかを示すフラグを設けておき，不必要になった制約式のフラグを操作することによって実現できる．不必要な制約式セルを制約リンクセルと共に削除する方法も考えられるが，この方法は後戻りができなくなるので使用できない．

5．評価

制約論理型プログラミング言語の歴史はまだ浅く研究段階の処理系がほとんどであり，広く利用できる制約論理型プログラミング言語を入手することが困難である．このため，図1に示したメタインタプリタ型の処理系との比較ではなく，本方式を用いた処理系上での，制約を用いたプログラムと生成検査法を用いたプログラムの処理時間を比較した評価を示す．

4章で示した実現手法を Quasar Prolog インタプリタ[4]に組み込み，N-Queen を利用した性能評価を Sun-3/260 上で行なった．N-Queen のプログラムは，生成検査法によるものと制約を利用したものを用いた．また，参考のために Quintus Prolog コンパイラによる生成検査法の実行時間も測定した．

生成検査法のプログラムは図8，1に示すように，place/2 によって仮説を生成する部分と，safe_gt/1 によって制約条件を検査する部分からなり，制約を用いたプログラムは図8，2に示すように，safe_cg/1 によって制約を付加する部分と，place/2 によって制約を満たす解を生成する部分からなる．なお，生成器となる place/2 は2つのプログラムで共通である．図8のプログラムは，次のような形式で呼び出す．

```
:- queens_generate_test(
        [X6, X5, X4, X3, X2, X1],
        [6, 5, 4, 3, 2, 1]).
```

この例では，生成検査法を用いて 6-Queen の解を求めている．

N-Queen を同一の PROLOG インタプリタにより2種類の方法で実行させた実行時間を表1に，生成検査法をコンパイルドコードで実行させた実行時間を表3に示す．測定した時間は，最初の解を求めるのに要した CPU 時間である．

表1に示されるように，盤が大きくなって生成される仮説枝の数が多くなるにつれて，制約を用

```
queens_generate_test(X, N) :-
        place(X, N),
        safe_gt(X).

safe_gt([]).
safe_gt([X| L]) :-
        check_gt(X, L, 1),
        safe_gt(L).

check_gt(_, [], _).
check_gt(X, [Y| L], N) :-
        X =\= Y + N,
        X =\= Y - N,
        NN := N + 1,
        check_gt(X, L, NN).

place([], []).
place([X| Y], NL) :-
        select_one(NL, X, NLL),
        place(Y, NLL).

select_one([A| L], A, L).
select_one([A| L], X, [A| L1]) :-
        select_one(L, X, L1).
```

1）生成検査型

```
queens_constraint_generate(X, N) :-
        safe_cg(X),
        place(X, N).

safe_cg([]).
safe_cg([X| L]) :-
        check_cg(X, L, 1),
        safe_cg(L).

check_cg(_, [], _).
check_cg(X, [Y| L], N) :-
        #{X =\= Y + N},
        #{X =\= Y - N},
        NN := N + 1,
        check_cg(X, L, NN).
```

2）制約充足型

図8　N-Queen プログラム

いたプログラムと生成検査法を用いたプログラムの実行速度の比が大きくなっている．例えば，盤の大きさが１２の場合には，制約を用いたプログラムでは1.6秒ほどで解が求められるのに対して，生成検査法を用いたプログラムでは12000秒，３時間以上かけなければ解を求めることができない．この問題では，制約を用いたプログラムは生成検査法のプログラムに対して7800倍ほど性能が改善されていることが示されている．さらに，盤の大きさが１４の場合には，制約を用いたプログラムでは14秒ほどで解が求められるのに対し，生成検査法を用いたプログラムでは時間がかかり過ぎて，現実的には解を求められない状態に達している．この評価では，同じ性能を持つ諸理系（この場合には，同一の処理系）であっても制約評価機構を利用できるかどうかによって，処理時間が全く異なると共に，解くことができる問題の大きさが大きく異なることが示されている．

表１　制約と生成検査法の比較
（単位は秒）

	制約	生成検査法	比率
6	0.07	0.62	8.9
8	0.38	13.1	34.5
10	0.48	185	385
12	1.55	12170	7852
14	13.7	-	-
16	90.6	-	-
18	425	-	-
20	2408	-	-

盤が大きくなり，生成される仮説枝が多くなるにつれて，実行時間の増加率がどのようになるかを表２に示す．この表は，制約，生成検査法のそれぞれで，盤の大きさが６のときの実行時間を１とした実行時間の比率を示したものである．例えば，盤の大きさが１２のときには，制約を用いたプログラムでは，２２倍程度の時間で解が得られるのに対して，生成検査法を用いたプログラムでは，20000倍近い時間を必要としている．この表では，制約を用いたプログラムでは，盤が大きくなっても実行時間の増加が比較的緩やかであるのに対して，一般的な生成検査法を用いたプログラム

では実行時間が急激に増大し，現実的に解を求めることができなくなる限界にすぐに到達してしまうことが示されている．

表２　実行時間の増加率
（単位は秒）

	制約		生成検査法	
	実行時間	比率	実行時間	比率
6	0.07	1	0.62	1
8	0.38	5.4	13.1	21.0
10	0.48	6.9	185	298
12	1.55	22.1	12170	19629
14	13.7	196	-	-
16	90.6	1294	-	-
18	425	6071	-	-
20	2408	34400	-	-

表３に示されるように，盤の大きさが小さい問題から，生成検査法を用いたプログラムをコンパイルドコードで実行させよりも，制約を用いたプログラムをインタプリタで実行させるほうが性能が高いことが明らかになっている．プログラムの性能を向上させる一般的な手段は，プログラムをコンパイルすることであるが，この方法では高々２０倍程度の向上しか見込めないという限界がある．表２では，コンパイラによる性能向上は，比較的小さな問題には効果があるが，仮説数が膨大になる大きな問題になればなるほど，制御戦略の変更を実現する制約評価機構の効果が大きいことが示されている．

以上の評価では，制約の効果が大きいことと共に，本報告で提案した制約管理機構の効率が高く，制約の利点を十分に引き出せることが示されている．なお，本方式では，制約評価機構を付加しても，制約が使用されないプログラムの実行時間はほとんど変わらず，性能の低下は５％程度に留まっている．

表３　コンパイラとの比較
（単位は秒）

	制約 インタプリタ	生成検査法 コンパイラ	比率
6	0.07	0.08	1.1
8	0.38	1.26	3.3
10	0.48	19.0	39.6
12	1.55	1281	826

６．まとめ

本報告では，各変数セルに制約情報を付加することによって効率のよい制約管理を行なう方式を提案し，その有効性を示した．制約解消機構に関しては受動的制約の解消機構について示すにとどめたが，能動的な制約評価を行なうための制約解消機構についても研究を進めたいと考えている．

制約の応用分野としては，知的な設計支援機能を持つ CAD などが考えられる．設計問題では，多くの仮説枝の中から制約を満たす解を求める処理が重要な位置を占める．本報告で評価を行なった N-Queen は，駒の位置を検査する処理以外は単一化と後戻りだけの簡単なプログラムであり，それぞれの仮説枝上の処理に消費される時間は比較的少ない．しかしながら設計問題では，各仮説枝上で浮動小数演算を伴う多くの演算処理が行なわれるために，各仮説枝上で消費される時間が多くなるものと考えられる．したがって，不必要な仮説枝を早期に刈る機能を持つ制約の効果が非常に大きくなると考えられため，現実的で複雑な設計問題などを対象として評価を行ないたいと考えている．

参考文献

[1] 坂井，相場：制約ロジックプログラミング言語 CAL，情報処理学会，記号処理研究会資料，SYM-47-1，(1988).

[2] Cohen J.: Constraint Logic Programming Languages, CACM, Vol.33, No.7, pp.52-68, (1990).

[3] Warren, D.H.D.: An Abstract Prolog Instruction Set, SRI International, Tech. Note 309, (1983).

[4] 砲崎：Quasar Prolog の言語仕様，人工知能学会，第４回全国大会論文集，pp.245-248，(1990).