5M−1

Quasar Prologの制約評価機構

砥崎　賢一
九州工業大学　情報工学部

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，制約論理型プログラミング言語[1]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．本報告では，Quasar Prolog[2]に実装された受動的制約評価機構の概要とその評価を示す．

2．受動的制約

Quasar Prologに実装された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受動的制約評価機能を持つ．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．制約評価機構を組み込んだ制約論理型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．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り，不適切な仮説枝の生成が押さえられるため，処理速度の大幅な向上が実現される．

3．言語仕様

制約を変数に付加する式は，以下に示すように#{Constraints}[3]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いる．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．例えば，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，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．

Quasar Prologで利用できる制約式は，以下の4種類である．

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-[...]
```

1から3に示す制約式は，遅延評価機能が付加されていることを除き，基本的にはそれぞれの組み込み述語と同様な機能を持つ．4に示した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．

制約は，1つの変数に対して複数を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

Constraint Evaluator in Quasar Prolog
Ken'ichi KAKIZAKI
Kyushu Institute of Technology

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．

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

Quasar Prolog では処理速度を重視しており，PROLOG によってメタインタプリタ[1]を記述する方式ではなくC言語を用いて制約評価機構を追加する方式を採っている．また，制約は，制約集合を一括管理する方法ではなく，制約情報を含むセルを変数に付加する[3]ことにより管理している．この方式では，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．

5．評価

生成検査法と制約を用いた2種類の N-Queen[3]を利用した性能評価を Sun-3/260 上で行なった．また，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．N-Queen を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2に示す．測定時間は，最初の解を求めるのに要した CPU 時間である．

表1では，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ることが示されている．例えば，盤の大きさが 12 の場合には，生成検査法を用いたプログラムでは解を求めるのに3時間以上かかるのに対して，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，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 以上の場合には，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，現実的には解が得られない状態に達している．

盤が大きくなるにつれて，実行時間がどのように増加するかを見ると，盤の大きさが 12 のときの実行時間を 6 のときのそれと比較すると，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．

表2では，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るよりも，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，処理速度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．

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	-	-

表2　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，Quasar Prolog に実装された受動的制約の機能と，その性能評価を示した．現在実装されている制約解消機構は受動的制約に限定されているが，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．

参考文献

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

[2] 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).

[3] 砲崎：制約論理型プログラミング言語の変数管理方式，情報処理学会，人工知能研究会資料，AI-74-1，(1991).