4H-5 論理型プログラムの書き換えモデルにおける read-only annotationの活用について

相田 仁, 後藤厚宏, 田中英彦, 元岡 達
(東京大学 工学部)

1. はじめに

我々は現在,論理型プログラムを高並列に実行するマシンPIEの開発を進めている。PIEにおいて,動的な判断により処理の効率化を図る方法としては

i) UPにおいて生成されたゴールフレームを適切なMMに送って負荷を分散する。
ii) MM内のゴールフレームのうちで解が求まりやすいと考えられるものから優先してUPで処理する。
iii) MM内のゴールフレームのうちで以前の処理の結果から失敗に至ると判定可能なものを消去する。
iv) UPにおいてゴールフレーム中の最適なリテラルについて書き換えを行なう。

の4つが考えられる[1,2]。このうちi),ii)に関しては,シミュレーションによりいくつかの方式の検討を行ない,結果を報告した[1]。iv)のゴールフレーム中のリテラル選択に関しても,プログラマからのアドバイスなしに引数の束縛状況のみに基いて評価・選択を行なう方式についてすでにシミュレーションを行なったが,優先度評価のための点数設定がかなり難しいことがわかった[2]。

そこで今回は,リテラル選択に対し,Concurrent Prolog[3]におけるのと同様なread-only annotationを用いてプログラマがアドバイスを与える方式をとりあげ,これにより適切なリテラル選択が可能になることを示す。

2. read-only annotationとそれを用いたdemand driven的処理

プログラム1はエラトステネスのふるいにより,小さい方からN個の素数を見出すプログラムの例である。このプログラムはgenerator-consumerの形をしているので,通常のprologのように,常に一番左のリテラルについて書き換えを行なったのでは(以下left-firstな実行と呼ぶ),generatorばかりが処理されて,解を見出すことができない。

このようなプログラムにおいてgenerator側とconsumer側の同期をとるメカニズムとして,read-only annotationが提案されている[3]。read-only annotationとは,リテラルの引数に現れる未束縛変数に付加される属性であり,

- read-only annotationのついた変数は,自由変数以外と単一化できない。
- 上記の単一化を行なった場合には,相手の変数にread-only属性がひきつがれる。

というものである。プログラム1において,?がread-only annotationである。

PIEにおける書き換えモデルでは,単一化が成功した場合,定義節の右辺はもとのリテラルの位置に挿入される。そこで,consumerの入力変数にread-only annotationをつけ(プログラム1),かつ,常にconsumerがproducerよりも左側に来るようにリテラルを配置しなおすと,left-firstな実行に

```
primes(N,O) :- integers(2,I,F),sift(N,I?,O),outstream(O?,F).

integers(_,[],F) :- nonvar(F),true.
integers(N,[N|I],F) :- var(F),N1 is N+1,integers(N1,I,F).

sift(0,_,[]).
sift(N,[m(_)|I],O) :- N > 0,sift(N,I?,O).
sift(N,[P|I],[P|O]) :- N > 0,number(P),N1 is N-1,filter(P,I?,R),sift(N1,R?,O).

filter(P,I,O) :- filter1(1,P,I,O).

filter1(_,_,[],[]).
filter1(P,P,[H|I],[m(H)|O]) :- filter1(1,P,I?,O).
filter1(N,P,[H|I],[H|O]) :- N < P,N1 is N+1,filter1(N1,P,I?,O).

outstream([],[]).
outstream([X|S],F) :- write(X),outstream(S?,F).
```

Program 1. Finding first N prime numbers

おいて，consumerの単一化ができない場合に限ってproducerが書き換えられる。そこでdemand driven的なメカニズムにより，必要最少限の書き換えにより解を得ることが可能である。

しかし，この方式では，左のリテラルから順にとにかく単一化をはじめてみて，read-only変数への束縛を行なおうとした時点でその単一化をとりやめ(suspension)，次のリテラルの単一化を試みる。これは単一化が成功するまでにいくつものリテラルに対応する定義節を参照する必要があり，好ましくない。

3. read-only annotationを利用したリテラル選択

あるゴールフレームが与えられたとき，定義節等を参照することなく，ゴールフレーム内に含まれる情報のみから適当な判断をして，前節で述べたdemand drivenによる実行において最終的に単一化に成功するリテラルを選び出すことができれば，コストの点で最も優れている。

実際に実行をおいかけてみると，そのようなリテラルは，前回の単一化により値の定まった変数をread-only変数として含んでいたリテラルである場合が多いことがわかる。そこで，read-only annotationのついた変数が束縛された後も，その引数にannotationが残るものとすれば，

①annotationのついた引数に値が束縛されている場合には，そのリテラルを優先して書き換える。 —bind-first(1)—

というリテラル選択ストラテジが考えられる。

この方式の有効性を確認するため，いくつかのリテラル選択方式のもとで10個の素数を求めた場合の書き換えの実行回数を表1に示す。表中で－で示した欄は，generatorばかりが書き換えられて，解が得られない場合を表わす。

上記の方式では，プログラム1のリテラル順で実行した場合(a)に，suspensionが全くなしに実行が可能であり，余分な書き換えもわずかにおさえられている(generator 1回分)。これに対し，consumerが左に来るようにリテラルの並べかえを行なった場合(b)には，left-firstな実行と同じく，余分な書き換えが全くなくなるかわりに，suspensionが生じる。これはconsumerがひととおり動きおわった時点で①を満たすリテラルがなくなるためである。そこで①に加えて

②値を束縛されたread-only引数を含むリテラルがない場合には，read-only引数を全く持たないリテラルを次に優先して書き換える。—bind-first(2)—

のようにすると，リテラルの配列順にかかわらず，suspensionなしに実行が可能であった。

Table 1. Execution profile

	(a) Program 1. user pred.	suspension	system pred.	(b) Reordered user pred.	suspension	system pred.
left-first	-	-	-	251	1123	405
bind-first(1)	262	0	422	251	246	405
bind-first(2)	262	0	422	262	0	422
depth-first	-	-	-	-	-	-
breadth-first	361	36	603	471	57	805

(- : fallen into infinite loop)

4. read-only annotationの新しい意味づけ

前節で述べたようなリテラル選択が行なわれる状況下では，read-only annotationがプログラム中で持つ意味として，単一化の禁止という消極的な意味に加えて，書き換えるべきリテラルを指示するという積極的な意味が生まれる。この新しい意味づけはプログラムの停止性や効率を保証するうえで，きわめて重要な役割を持つものと考えられる。

5. おわりに

書き換えモデルによる論理型プログラムの実行において，read-only annotationを用いてプログラマがアドバイスを与えることにより，ほぼ最適に近いリテラル選択を行なうことができることを示した。

今後さらに，read-only annotationをゴールフレームのAND分割に役立てる方式についても検討を進める予定である。

参考文献

[1] 丸山 他：高並列推論エンジンPIE～並列度のシミュレーションとその評価，EC83-39，1983年12月.

[2] 後藤 他：高並列推論エンジンPIEにおける並列処理の効率化手法について，EC83-9，1983年5月.

[3] Shapiro, E.Y.: A Subset of Concurrent Prolog and Its Interpreter, ICOT Technical Report TR-003, February 1983.