ICOT Technical Memorandom: TM-1137

TM-1137

計画問題支援のための並列論理型制約処理系の設計と実現

上田　晴康、国藤　進（富士通）

December, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

計画問題支援のための並列論理型制約処理系の設計と実現

10－3

A Parallel Constraint Logic Language for Planning Problem and its Implementation.

上田晴康* 國藤進
Haruyasu Ueda Susumu Kunifuji
(株) 富士通研究所 国際情報社会科学研究所
International Institute for Advanced Study of Social Information Science,
FUJITSU LABORATORIES LTD.

Abstract: Generating a plan from constraints is one of the most difficult problem. To solve the problem, we need best-first search function or at least all-solution search function. We propose a new Constraint Parallel Logic Language for Planning Problem. This language has a flat-GHC-like syntax and ANDORRA-like semantics, for searching all solutions. The GHC-like commit-bar is used to prune useless or-parallel search environment in addition to head-unification. The language also has a constraint solver. There are two types of constraint. One is not-equal relationship like "f(a)≠f(X)," and the other is linear inequality like "$X \leq 4 * Y + 3$." The constraints are declared only in the guard part.

The language is implemented in KL1. We show the specifications of the language with examples. We also show the execution efficiency. Generate & test program is more efficient than generate program, because the tester intervenes in the generator and prunes useless generations.

1 はじめに

人工知能で扱われる難しい問題の一つとして、計画型の問題がある。計画型問題は、• 計画として正しいかどうかを判定する条件、• 計画がどの程度望ましいかを判定する評価基準、 を与えられて計画を探索する問題である。これは探索空間が莫大なため、ヒューリスティクスを用いて問題を解くことが多い。しかし、そのヒューリスティクスの獲得も極めて難しいため、ヒューリスティクスなしで計画問題を解く方法が望まれている。

計画型問題で計画の望ましさを判定する評価基準を与えるということは、見方を変えると「評価を最大にする」という暗黙の条件を与えているとみなすことができる。そこで、正しいかどうかを判定する条件を強い制約、評価を最大にするという条件を弱い制約と呼ぶ ([1])。

計画型の問題では、• 強い制約を解消して制約を満たす組合せを見つけ出す機能と • 全解探索ないし弱い制約を用いた最良探索 (best first search) で必要なすべての解を探す機能 の二つの機能が必要である。そこで、著者らは強い制約の解消と全解探索 (ないし将来的には最良探索) を行なう機能を持った処理系を作成した。

本論文では、この制約処理系の設計方針とその実現について述べている。

2 並列論理型制約処理系の設計

2.1 処理系の満たすべき条件

著者等は、並列制約処理系の満たすべき条件として、次の3点を考えた。

• 解であるかどうかをテストするプログラムと探索のプログラムが同一のものであること。

*連絡先: 上田晴康 〒144 大田区新蒲田 1-17-25
E-mail ueda@iias.flab.fujitsu.co.jp

- 全解探索をすること。
- 並列に解を探索すること。

テストと探索が同一のプログラムで表現できることという条件は、本処理系が計画の満たすべき条件の記述から計画を作ることを目的としていることから来ている。すなわち、正しい計画かどうかをテストするプログラムを作れば計画が出来上がるようにするためである。

全解探索をするという条件は計画問題のためには最良探索であるべきだが、そのための一般的な良いヒューリスティクスはまだ得られていない。

全解探索は効率は悪いが出て来た全ての計画のそれぞれに対して選好度を計算すれば最良の解を求めることはできる。また、全解探索の途中で選好度の低いものの枝刈をすれば最良探索となるので、将来的な拡張に適している。そこで、本処理系は全解探索を行なう。

最後の並列に解を探索するという条件は、全解探索(又は最良探索)をするに当たってつけた条件である。全解探索は大変に計算量の多い処理であり、また各々の解の探索は独立枠した処理であるため通信を行う必要がなく、並列処理に適した仕事であるためである。

2.2 本処理系の仕様

著者らは、2.1の議論に基づき次のような仕様を持った処理系を提案する。 (1)GHC的な構文を持つ。 (2)各節毎にor並列を行なう。ただし、or並列を行なう節はガードのゴールが全て失敗しなかった節とする。 (3)制約は、ガードにのみ記述できる。 図1 に、本処理系の実行の概略を示す。

2.1で示した3つの条件は、or並列に実行を行なう論理型言語が満たしている性質である。しかしor並列の探索は、それぞれの探索の候補が独立の環境を必要とするため、候補毎に環境をコピーするという重い処理を伴うにも関わらず、

プログラム:

```
a(a,X) :- X >= 0 | b(X), c(X).
a(b,X) :- X < 0  | b(X), d(X).

b(-2).  b(-1).  b(0).  b(1).  b(2).
c(-2).  c(2).
d(-1).  d(1).

e(X) :- X < 0  |  true.
```

実行例:

ゴール	flat GHC	本処理系
a(a,2)	成功	成功
a(b,-1)	成功	成功
a(a,2)	失敗	失敗
a(a,X)	b(X), c(X) 共に中断して dead lock	b(X) で それぞれX=0,1,2という値を持った3つのプロセスに分かれ、そのうちX=2のプロセスだけが成功する。
a(Y,X), e(X)	a(Y,X),e(X) 共に中断して dead lock	非決定性のないように e(X), a(Y,X), d(X), b(X) の順に実行が進み、e(X) では X<0、a(Y,X) では Y=b、d(X) では X=-1 がそれぞれ実行される。

図 1: GHC と本処理系の実行の対比

単純なor並列Prologはヘッドユニファイできた節という細かい単位でor並列が起きて効率が悪い。

そこで、本処理系は非決定性の起きる時点をコミットバーの直後とすることにより、非決定性の起きる時点を厳密に記述できるようにした。この結果、不必要なor並列の起きるの防ぐだけでなく、or並列の並列の粒度を粗くすることもでき、並列実行に対してより効果的である。

また、著者らは制約の記述をガード内に限定した。制約は計画の満たすべき条件であり、テストの拡張であるととらえたため、ガードに記述するのが自然であると考えたからである。

2.3 ガードの実行

ガードの実行は、ヘッドユニフィケーションと、ガードに書かれたユニファイ、制約、テスト述語の実行からなる。それぞれの実行は、成功、失敗の他に延期という状態で終了する。

延期が起きるのは、ガードの外へ情報を出そうとした時か、テストに必要な情報が足りなかった

場合である。ガードの外へ情報を出すことは、コミットしてor並列のための環境が作られるまで延期される。テストに必要な情報が足りなかった場合は、必要な情報が得られるまで延期される。

例えば、ユニファイを実行すると失敗するか、新たな置換(substitution)と共に成功する。この置換は、そのガードにローカルに実行されるが、グローバルな実行はコミットまで延期される。

制約は、制約が明らかに充足不能であるばあいは失敗し、そうでなければその節がコミットされた後で制約解消系が起動される。明らかに充足不能かどうかの判定は制約によって異なる。

ガード中の全てのゴールが成功又は延期する時その節はコミットされる。

2.4 or並列の制御

一つのゴールに関して、全ての節のガードが実行された後、コミットした各節のボディが実行される。一つのゴールに対して、一つの節のみがコミットされた場合は、決定的な実行ができるが、複数の節がコミットした場合非決定的な解の探索を行なう必要がある。

本処理系では、不必要なor並列を押えるために、ANDORRA([2])の様に、全ての決定的な実行を行った後に非決定的なor並列の実行を行う.

本処理系は、このために決定的実行用と非決定的用の2本のリダクションキューを持つ。非決定的なゴールは決定的実行用のキューが空になった後で取り出され、変数の環境と制約を各節毎にコピーしてからボディの実行をする。

2.5 ボディの実行

ボディの実行では、ボディに記述された述語の実行の前に、ガードで延期された置換のグローバルな環境への適用と,ガードで実行された制約の解消を行なう。その後、ボディに書かれた述語を通常の論理型言語と同様に実行する。

2.6 本処理系の実行例

本処理系の直観的な理解を助けるために N-Queen を計算する例を示す。

```
queens(A,X) :- true |
    length(A,X),
    range(A,X),
    not_attack(X).

length(A,[_|X]) :- A>= 1 | A1 is A - 1,
    length(A1,X).
length(0,[]) :- true | true.

range(A,[]) :- true | true.
range(A,[X1|X]) :- true | range1(A,X1),
    range(A,X).

range1(A,A) :- A >= 1 | true.
range1(A,X) :- A >= 2 |  A1 is A - 1,
    range(A1, X).

not_attack([X1|X]) :- true |
    not_attack1(X1,1,X),
    not_attack(X).
not_attack([]) :- true | true.

not_attack1(Q,N,[Q1|Qr]) :- N1 is N+1,
    Q1 =\= Q,
    Q11 is Q1+N, Q1 =\= Q11,
    Q12 is Q1-N, Q1 =\= Q12|
    not_attack1(Q,N1,Qr).
not_attack1(_,_,[]):- true | true.
```

図 2: N Queen のプログラム

図2のプログラムは、GHCとして読めば、N-Queenの盤面が来た時のみ成功し、それ以外では失敗するようなプログラムとして読める。

本処理系では、まずlength/2によって長さがnのリストが作られ，range/2で、リストの各要素がrange1を満たすことがわかるが、これは非決定的な実行を行なうため非決定的キューに入れられ実行が延期される。次にnot_attack/1が実行されて、Queen同士が攻撃しあわない条件が≠と一次方程式の制約として加えられる。最後に，延期されていたrange1/2が非決定的な解を作成して，not_attack/1によって入れられた制約を満たすものだけが成功する。

これに対し、GHCではor並列探索の機能がなく、ガードに書いた変数に値を割り当てること

もしないため、探索としては使えない。

逐次型 Prolog で、コミットバーを！と読み替えて実行すると、length/2 と range/2 で generate される n^n 個の解を一度ずつ not_attack/1 で test するために、大変に効率が悪い。

ANDORRA のような and-or 並列の Prolog では、本処理系と同様の実行を行なうが、≠が制約でないため、not_attack/1 の実行の前に range を行なうように指示しなくてはならずプログラム作成が困難である。又効率もやや悪い。

3 並列論理型制約処理系の実現

著者らは、本処理系を flatGHC の一種である KL1([4]) で実現した。この実現では、(1)≠ (2)線形不等式 の二つの制約を実現した.

≠は，ユニファイア同様に引数の各要素を異なる部分が出るまで比較する。変数が現れたらその変数に関する比較は延期される。

線形一次不等式は、sup-inf 法 [3] を用いて実現した。これより変数の値の範囲が常に計算されているので、その範囲のみから充足不能がわかる時、ガードで fail する。

3.1 実行効率

本処理系で、図2のプログラムを実行した時の本処理系全体のリダクション数をはかったのが図3である。単純な generate&test ではなく動的に枝刈をしているため、length(N,X),range(N,X) で単純に解を generat するよりも N-Queen の解を求める方がリダクション数が減っている。

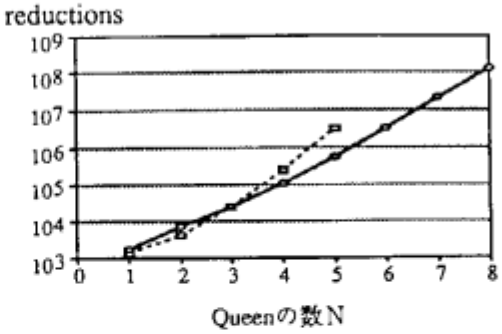

図 3: N Queen の実行のリダクション数
縦軸は本処理系の総リダクション数。
点線は length(N,X),range(N,X) 、実線は length(N,X),range(N,X),not_attack(X) の実行をプロットした。前者は N>5 ではメモリ不足のため計測不能。

4 おわりに

本論分では新しい制約論理系の設計と実現について述べた。本制約処理系は flat GHC で書かれたテストのプログラムを、or 並列的に解釈して実行することにより全解探索を行う処理系である。ガードに書かれたゴールのいくつかは制約として解釈される。

また、本処理系は単純な generate&test のプログラムをできるだけ fail しないように枝刈して実行するため、効率が良いことを示した。

今後の方針としては、● 弱い制約を用いて、最良探索を行う、● 節のオルタナティブをガード部の評価時だけでは確定せず、動的に減らしてゆくなどがある。

謝辞 本研究の一部は第五世代コンピュータプロジェクトの一環として行なわれました。また、本処理系の実現にあたり、(株) 富士通 SSL の岩内さん他に大変お世話になりました。

参考文献

[1] 上田, 國藤, 他. GRAPE の計画問題支援機能 - 並列制約論理型言語に基づくアプローチ -. 計測自動制御学会. 第 12 回知能システムシンポジウム, Oct. 1990.

[2] Haridi., S. and Brand., P. ANDORRA PROLOG - An Integration of PROLOG and Committed Choice Language. Proc. of 5th FGCS., 1988, ICOT.

[3] Shostak, R.E. et.al., On the SUP-INF Method for Proving Presburger Formulas, J. of the Association for Computing Machinery, Vol. 24, 1977.

[4] 渕 一博 監修、並列論理型言語 GHC とその応用, 共立出版, 1988