節単位処理モデルの提案

佐藤 健　　増沢 秀穂　　板敷 晃弘

(富士通株式会社)

1.はじめに

PROLOGを並列に処理するOR並列推論モデルが数多く提案されている。

OR並列処理においては，分散したプロセス間では通信を行う必要がないので，独立に処理を行うことができるという利点があるが，一方実行出来るプロセスの個数が増大し，資源が尽きるという事態が発生する可能性がある。

本稿では，数の爆発を抑える制御を有し，かつ通信コストの小さいOR並列推論モデルを提案する。

2.基本的な考え方

我々は実行途中において解の保持をすることによって数の爆発を抑えることができると考えた。つまり，PROLOGの実行を「解を求める手続きの呼出の列」とみなし，その手続きが呼び出されたときには，最初に求められた解を送りその他の解はそれを呼び出した側から要求があるまでは保持するという方法を取った。

しかし，PROLOGの実行を手続きの呼出ととらえるモデルには，AND プロセス及びORプロセスの２種類のプロセスによりそのサブゴールを処理しているモデル[1]が多いが，このモデルには以下のような欠点があると考えた。

①AND プロセス及びORプロセス間には，通信時間のオーバーヘッドがあると思われる。

②答の保持をORプロセス単位すなわちリテラル単位で行っているモデルがあるが，この場合は並列度があまりにも小さくなると思われる。

我々は，以上の欠点を解決するために，次のように考えた。

①通信コストを軽減するために，AND プロセスとその子のORプロセスを一つのプロセス (Unify プロセス) として，連続的に処理する。

②並列度をあげるために，リテラル単位ではなく，節単位で解を保持する。

3.節単位処理モデルの特徴

以上をまとめ，提案モデルには以下の特徴を持たせた。

①並列処理方式としてOR並列処理方式を採用し，AND 関係は逐次に処理するようにした。

②数の爆発を抑えるために，Unify プロセスに解を保持する機構を設けた。

③通信コストを軽減するためにリテラルとUnificationが成功する節単位でUnify プロセスが処理するようにした。

④並列度を上げるために，Unify プロセスごとに解を保持するようにした。

4.節単位処理方式

以上の特徴を実現するために以下の方式にした。

①解を保持する機構を実現するために，Unify プロセスには解を親プロセスに送るかどうかを表す状態を設けた。(図１参照)

Unify プロセスが生成されたとき，通常は初期状態として解を送る状態 (SEND) にする。そして，最初の解が求まったときにその解を親のプロセスへ送り，そのプロセスの状態を，解を保持する状態 (SAVE) に変えることにより，それ以降の解が求まったときはそれを保持できるようにした。そして，REDOメッセージが送られたときには保持している解を送るようにした。

②節単位で処理するためには，AND プロセスとその子のORプロセスを連続して実行させればよい。そこで我々は，AND プロセスとORプロセスの動作を参考にしながら，プロセス間のメッセージとプロセスの状態ごとにUnify プロセスの動作を以下のように定めた。

Unify プロセスの処理

(a)メッセージが質問のとき

ヘッドのみの質問の場合はUnify プロセスのSTATEをALLFOUNDにし，親のプロセスへ質問を解として返す。ボディのある場合はそのボディの最も左のリテラルとUnification が成功する節を捜し，成功する節がある場合はSTATE をSENDにし，そのリテラルの値を節のヘッドとボディに反映させ，どこかのプロセッサにそれを質問として送る。成功する節のない場合は親のプロセスへFAILを送り，自分は消滅する。

(b)メッセージが解のとき

解を AND関係にあるリテラルに反映させる。

最も右のリテラルの解のときSTATE がSENDであれば，親のプロセスにヘッドを解として返し，STATE をSAVEにし，STATE がSAVEであれば，ヘッドを解として保持しておく。最も右のリテラルの解でなければ，直ぐ右にあるリテラルに関して，Unification が成功する節を捜し，成功する節のある場合はそのリテラルの値を

節のヘッドとボディに反映させ，どこかのプロセッサにそれを質問として送る。成功する節のない場合は今解を送ってきたプロセスにREDOを送り，別の解を要求する。

(c)メッセージがFAILのとき

あるリテラルから分岐した子プロセスの最後まで残っていたものからのFAILでなければ何もしない。最後の子プロセスからのFAILがきたとき，リテラルが最も左のものであれば，親のプロセスにFAILを送り自分は消滅し，リテラルが最も左のものでなければ，直ぐ左にあるリテラルの対応する解を送ってきた子のプロセスへREDOメッセージを送る。

(d)メッセージがREDOのとき

STATE がSAVEのときは，そのプロセスに保持している解があれば，それを親のプロセスへ送る。保持している解がなければ，最も右のリテラルの子プロセスにREDOを送り，STATE をSENDにする。STATE がALLFOUNDのときは親のプロセスにFAILを送り自分は消滅する。

図2に本モデルの実行例を示す。実行例では，プロセス3.4 が並列に処理され，プロセス3.4 から返された答に対して，それぞれプロセス9.10が生成され，それらがまた並列に処理されている。

5.並列処理性の論理的評価

本モデルの並列処理性の評価をシミュレータを作成して行った。処理時間としては，Unification 処理だけを考慮し，また一回のUnification 処理は成功，失敗とも一定時間を要するとした。

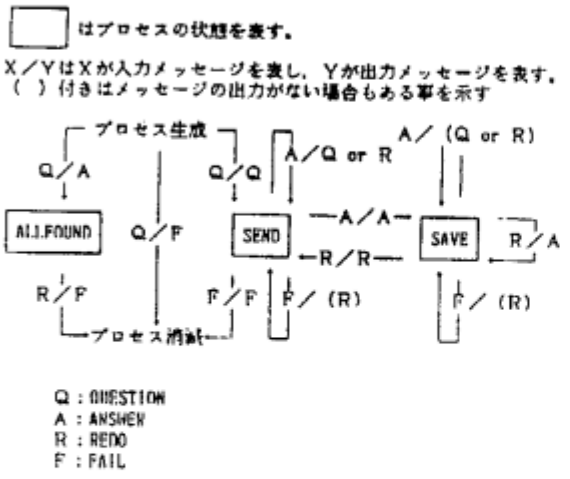

図1 Unify プロセスの状態遷移図

その結果本モデルでは，平均並列度が4QUEENで完全OR並列の0.38倍，6QUEENで0.07倍となり，数の爆発の抑制の効果が確認できた。

6.おわりに

通信コストが小さく，また数の爆発を抑える制御を有するOR並列推論モデルを提案した。提案モデルの論理的評価では，並列度が完全OR並列より小さくなっており，プロセスの個数の爆発を抑える効果を確認した。評価に使用した例題プログラムは，小規模であり，実際の応用プログラムにおける1 つのモジュールに相当すると考えられる。そこで，実際の応用プログラムを考えた場合には，並列度がどの程度大きくなるか及び有限資源の環境下で，それが妥当な並列度であるか等を評価する必要がある。

なお本研究は第5世代計算機プロジェクトの一環として，ICOTの委託で行ったものである。~~ただし，論文中の見解は必ずしもICOTのそれではない。~~

参考文献

〔1〕Conery.J.S. and Kibler.D.F. : "Parallel Interpretation of Logic Programs ". ACM conf. of Functional Language (1981) .

祖父（X,Y）：- 親（X,Z），父（Z,Y）．
親（X,Y）：- 父（X,Y）．
父（一郎，太郎）．
父（太郎，与作）．
親（X,Y）：- 母（X,Y）．
父（花子，万作）．
母（一郎，花子）．

上記のデータベースに対して，？—祖父（X,Y）を行ったときの実行途中のプロセスの状態を下に示す。

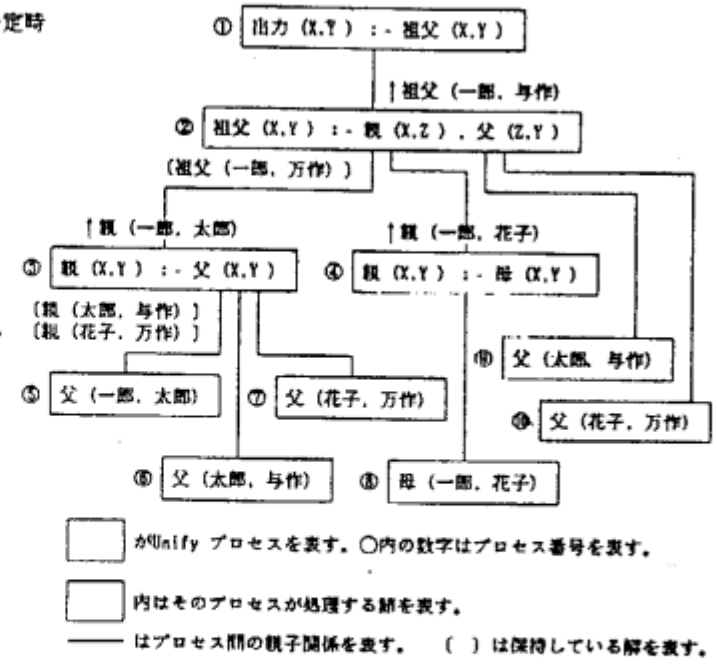

図2 実行例