5D-4

ユニフィケーション向き計算機に関する —検討

後藤 厚宏　田中 英彦　元岡 達
(東京大学 工学部)

1．はじめに

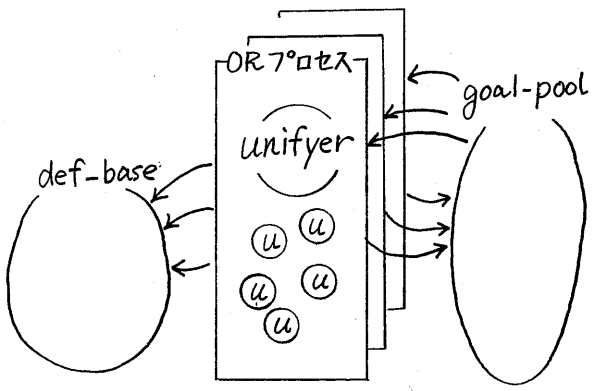
PROLOGは、①述語論理に基づいていることによる言語上の素質の良さ、及び②内在する高並列性の二つの理由から注目され、現在各方面で、言語の記述力の強化、Data Flow 等の高度並列計算機構との親和性等が検討されている。

PROLOGプログラムは、定義節の集合（定義のdata－base：def-base）とそれに対して与えられるgoal節（and で結ばれた複数のgoal）からなり、その実行は統合化操作（unification ）を基本にして行なわれる。

ここでは、この統合化をおこなう機能モジュールとしてunifyer なるものを考える。unifyer は、goal節中のひとつのgoalとdef-base中の定義節を与えられるとパターン照合による変数の結合を行ない、次にその結果として得られる置換リストを定義節のbody部に適用することにより新たなgoal節を作り出す。

PROLOGインタプリタはこの操作を求めたい変数に値が結合されるか、またはgoal節がなくなるまで繰り返す。これはAND－ORモデルによるユニフィケーションの並列実行として考えることができる。

本稿ではPROLOGマシンのベースとなるユニフィケーション並列実行マシンのイメージについて考えてみる。



［図1］ OR並列モデル

2．並列ユニフィケーションマシンの考え方

・OR並列とAND並列

OR並列とは定義間の並列性である。goal節の中のひとつのgoalについてunify される定義節は複数ある。OR並列モデルでは、あるgoal節の一つのgoalについてそれとunifiable なdef-base中の定義節の数だけの並列性を持つ。

AND並列とはgoal節を構成する複数のgoalの間の並列性である。goal節の各goalはローカル変数によって互いに束縛しあう。ひとつのgoal節について各goalを並列に処理して解を求め、その中からconsistentな解を求める。一般には各goalの解集合の表現が問題となる。そこで、goal間で共有される変数について方向付けをおこない、その制限のもとでgoalを並列に実行する。しかし、ひとつのgoalの解を求めることによって残りのgoalの解の探索空間を小さくなることもあるため、resourceが有限である以上OR並列を優先させた方がよいと思われる。

そこで［図1］に示すOR並列モデルから考える。

・ORプロセス

各ORプロセスは多数のunifyer を持つ。ORプロセスはgoal-pool からひとつのgoal節を、またdef-baseからそれと統合化可能な定義節のセットを取り出し、unifyer によって新たなgoal節（たいてい複数）をつくりだしてgoal-pool に戻す。各goal節は、導出情報とともにgoal-pool に戻されることにより独立に実行できる。

ORプロセスにおいて重要なことは、各goal節の並列度にあわせてunifyer を効率よく割り当てることである。

このようなOR並列モデルについては、すでにその並列実行の効果が充分にあることが示されている。［1］

・def-base

def-baseは多数のassertion （頭部のみの節）とrule（頭部とbodyからなる節）からなり、定義節の各項についての高速サーチが必要となる。探索キーとなる頭部の述語によって部分集合に分けた場合、ふたつの見方ができる。ひとつはプログラムの手続きとしての見方であり、定義節の多くは引数として

変数を持つruleからなる。

もうひとつは関係data-base としての見方である。ある述語についてのassertion が多数存在する場合、その述語（関係）についての関係表とみなすことができる。与えられたgoal節は、このdata-base に対するquery に相当する。

与えられた問題によって、手続き的性格が強い場合と関係データ的性格が強い場合とがある。

ところで、ユニフィケーションの並列実行で問題となるのは、def-baseにたいするアクセス競合である。def-baseは、その更新を考えなければ各unifyer にそのコピーを置くことができるが、解く問題が大きい場合は全てのコピーを持つことが難しくなる。しかし、問題が大きくプログラムが複雑になってきた場合、言語のレベルにおいて大域的な制御構造による階層化が導入されると考えるのが自然である。

def-baseではこのような制御構造を活用する。つまり、各unifyer がdef-baseのcache を持ち、ひとつまたは複数のunifyer のグループごとにサブプログラム単位の定義節をかかえる。この場合、手続き的に記述されたruleが中心となる。但し、定義節のworking-set が活きるようなダイナミックなスケジューリングが必要である。

また、大きいプログラムではassertion の数が相当多くなると思われる。ＰＲＯＬＯＧは、データとプログラムが同一の形をしていることがひとつの特徴であるが、その物理データ構造を考えた場合両者を同一に扱うことは効率的でないと考える。そこでdef-baseからasertionの集合を取り出してassertion-baseとして独立させることも有効である。

・goal-pool

各goal節の独立性により、goal-pool は分散させることが可能である。但しＯＲ並列モデルは基本的にbreadth-first な実行であるため、ＯＲプロセスの起動においてgoal節のand 結合を活かしたスケジューリングが必要である。

たとえば、言語上、各goalの間の評価の優先順位が指定されていない場合は、「引数が定数に結合されているものから」といった優先順位によるdepth-first の導入が考えられる。

またbreadth-first な実行では重複解が数多く現われる可能性がある。部分的depth-first を導入する場合、goal-pool に対するglobalな監視人を置くことによって資源を節約できる。しかし、監視のためのoverheadと節約できる資源とのtrade-off に依存する。

３．並列ユニフィケーションマシンのイメージ

高パーフォマンスなユニフィケーション向き並列アーキテクチャを考える上でのポイントは、①構成モジュールの種類が少なく、各々がシンプルであること、②モジュール間の通信量が少ないことである。ユニフィケーションの並列実行マシンを考える場合、①のモジュールとしてunifyer を中心に考えるのが自然である。問題となるのは②の要求を満たすようにdef-baseとgoal-pool を分散させることである。

［図２］にマシンのイメージを示す。

goal-pool は多バンク構成の構造メモリであり、常に多数のgoal-manager（gm）によってgoalの優先順位が評価される。goal-allocater（ga）はgoal-pool からactiveなgoal節を取り出し、内部にある定義節のテーブルを参照して必要な数のunifyer を起動するか、または基本述語プロセッサ（bp）を起動する。起動されたunifyer は与えられたgoal節と指定された定義節のユニフィケーションをおこなって新goal節を生成し、ふたたびgoal-pool に戻す。

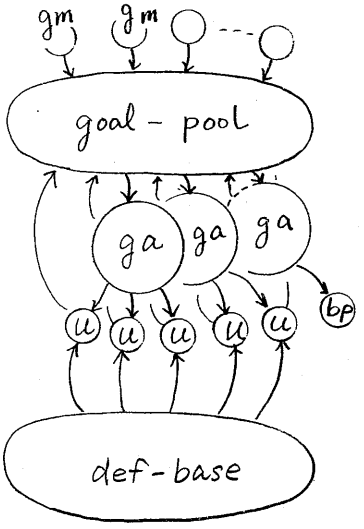
４．おわりに

ＰＲＯＬＯＧはプログラムにおけるアルゴリズムをロジックとコントロールに分離できる言語である。ＰＲＯＬＯＧマシンを考える場合、ロジックは並列ユニフィケーションに対応できる。問題はそれを効率化するコントロールの記述とマシンにおける実現である。

今後は言語とマシン構造の両面からコントロールの明確化と効率化を考える必要がある。

文献

［１］．相田　他、『並列ＰＲＯＬＯＧシステム“Paralog”の性能測定』、第２４回情処全大



［図２］