推論機能と関係データベースの融合 — その質問処理方式 —

5L-7

土井 晃一† 大森 匡† 嶺野 和夫†† 吉田 敦† 田中 英彦†
† 東京大学 工学部 †† 富士通株式会社

1. はじめに

ruleとfactが大量にある論理型言語を動かすには二次記憶としてデータベースを使う。しかし推論エンジン(IE)とデータベースをただつないだだけでは、両者間の通信の手間や探索空間に関する影響が無視できない。そこで推論・検索を、関係データベースマシンに対し、単一化操作を行なえるよう拡張した演繹データベースマシン(DDB)上で行う方法がある。本論文では論理型言語としてprologに話を限定して、与えられたprologのプログラムをネットワーク形式にコンパイルし、それに基づいて単一化・縮退を実行する際の戦略(関係代数を使うかどうか)の決定について述べる。

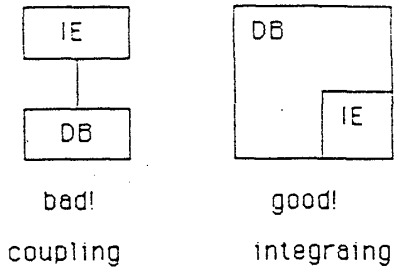

fig.1 two configulations for DDB.

2. prolog-compiler

2. 1 prolog-compiler

prologを関係代数によって実装するために、prologの一つのclauseをデータベースの一つのtransaction とし、prologを実行するための操作もまた一つのtransactionとする。そして演繹データベースユニットとしてunify-processor とsortorからなるユニットを考え、複数の演繹データベースユニットが相互にデータをやりとりしながらprologを実行していく状況を考える。このような演繹データベースユニットが並列に動きながらprologを実行していくことになる。prologの実行に対して可能な限り前処理を行うものとし、具体的には関係データベース機能の使い方を実行前に可能な限り決定しておくものとする。実行前に決定できない部分はruleの再帰呼出しを行う部分と、実行中でのruleの追加(assert)と削除(retract)である。これに対し我々はこの部分をどのように関係代数の実行と組み合わせて実行するかについても考察した。従って本研究で作成しているものは一種のprolog-compiler であるとも考えられる。

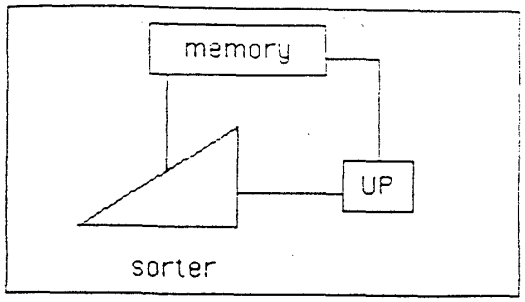

fig.2 DDB unit

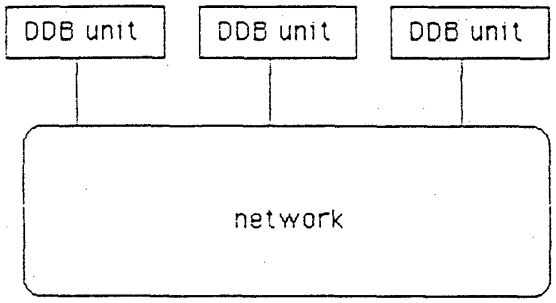

fig.3 DDB unit network

2. 2 rule-cell

prologのclauseをtransaction と考えるので、ruleに関する情報を保持しておくための実行の単位が必要となる。この実行の単位をrule-cell と名付ける。rule-cell にはbody各々のsub-goalに対して、引数の各々がatom、functor 、変数であるときの子のclause検索の戦略(relationを使わないときは子のclauseへのポインタ)が書いてある。この戦略によってunify-joinをするかどうか、を決定する。

Integration of Inference Function and Relational Database - Its methodology of query processing -

Kouichi DOI †, Tadashi OMORI †, Kazuo MINUNO††, Atsushi YOSHIDA †, Hidehiko TANAKA †
† The University of Tokyo, †† FUJITSU LIMITED

戦略はミクロな戦略（clauseの探索の方法）とマクロな戦略（演繹データベースユニットの割付けの最適化を図る方法）に分かれる。

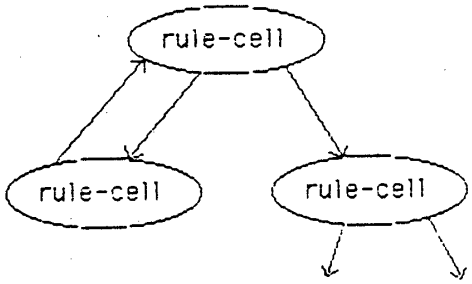

fig.4 network of rule-cell

p1(X,Y) :- p2(X,Z),p3(Y,Z).

↓

rule-cell

predicate name	p1
pointer to children of sub-goal1	if any
strategy of sub-goal1(atom)	relation
strategy of sub-goal1(variable)	pointer
strategy of sub-goal1(functor)	relation

fig.5 content of rule-cell

2．3ミクロな戦略決定

ミクロな戦略決定としては、子のclause探索法としてclauseへのポインタを完全に張る方法とrelationを使う方法に分かれる。これを（a）selection の場合と（b）unify-joinの場合とに分けて考えてみる。

（a）selection の場合

clauseへのポインタを使った場合は、子のclauseすべてにポインタを張ってOR候補を探してくるので、検索時間はアドレス計算の時間とディスクへのアクセスの時間との和になり、結果の候補節の数に比例する。その結果候補節の数が一定のときには、selection の成功率に比例し、成功率が一定のときには、候補節の数に比例する。

relationを使った場合、候補節の絞り込みをsorterを使って行うとすると、候補節の全数にだけに依存し、実際の成功率には無関係になる。

つまり候補節の数が一定のときselectivty（selection の成功率）が一定以上だと（b）の方がよい。このクロス・ポイントは候補節の全数（少ないときだけ）と成功率から上述のように決まるからどちらの戦略を取るかをrule-cell に書いておく。更にclauseへのポインタを使う戦略を使った場合は参照するclauseへのポインタをrule-cell の中に実際に書いておく。

（b）unify-joinの場合

clauseへのポインタを使った場合はselection のときと同様に結果の候補節の数に比例する。またrelationを使った場合はjoinをする前の二つのrelationの大きさの和に比例する。つまりこの場合もselection のときと同様に成功率のみに依存する。

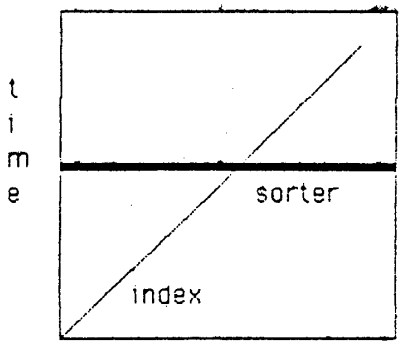

fig.6 number of OR candidate

2．4マクロな戦略決定

マクロな戦略決定では、演繹データベースユニットの使用効率を上げるためにrule-cell の割付けを考える。そのためにはrule／goalグラフのpath長の長いものを優先的に、重いrule-cell （実行の収束時に重くなるrule-cell ）を優先的に、ネックとなるrule-cell （実行の途中で重くなるrule-cell ）を優先的に実行するアルゴリズムを考える。rule-cell の重さはその子のrule-cell の数に比例するとして算出され、基本的には子のrule-cell の数の多いものを優先的に割付ければよい。

2．5assert,retractがあった場合

prologのclauseとしてassert,retractがあるときは、実行前に候補節の絞り込みの成功率が決定できないために前処理は完全にはできない。predicate に対するロックを導入することにより、並列に動くprologでのassert,retract が可能になる。この場合どのrule-cell を使って現在のrule-cell を実行しているのかを知るための実行履歴が必要になる。

3．おわりに

以上のようなシステム構成でprologを動かすことは、演繹データベースの実際の応用の方法を示すことになる。このモデルが動くことをこれから検証していく予定である。