プログラミング-言語・基礎・実践- 8－28
(1992. 8. 21)

# OR並列Prologにおけるプライオリティ制御機構とその応用

松田 秀雄，鈴鹿 重雄，金田 悠紀夫
神戸大学工学部情報知能工学科

本稿ではPrologのOR並列実行においてユーザの指定した評価関数の値をプライオリティとして優先度制御を行なう方式を提案している．従来のOR並列Prolog処理系では並列に実行されるゴールの数が組合せ的に増大する可能性があったが，ゴールにプライオリティを付加し高いプライオリティのものから実行することにより，実行時の探索空間を不必要に広げずに最適解に近いゴールのみを選択して実行できる．共有メモリ型並列計算機上で処理系を実現し，遺伝子情報処理の一分野である分子系統樹の推定を行なったところ高い台数効果が得られ本方式の有効性が示された．

# A Priority Control Mechanism for OR-Parallel Prolog and Its Application

Hideo MATSUDA, Shigeo SUZUKA and Yukio KANEDA,
Department of Computer and Systems Engineering,
Faculty of Engineering, Kobe University

This paper proposes a priority control mechanism for Or-parallel Prolog system. The scheduling method for controlling the Or-parallel execution plays an important role to achieve high performance. We introduce *priority* as a scheduling criteria in the Or-parallel execution. The optimal solution can be obtained as the first one by this mechanism. We implemented a system with this mechanism on a tight-coupled multiprocessor machine (Sequent Symmetry) and applied it to the estimation for molecular phylogenetic trees. The execution time for this problem was substantially decreased and the speedup factor better than linear were obtained on 20 processors.

## 1 はじめに

知識情報処理の応用分野が多様化，大規模化するにつれて，処理速度のさらなる向上が求められている．このためには並列処理技術の導入が不可欠と考えられており，現在までに種々の方法が提案されている．その中でも，特に論理型言語Prologによる並列処理については，論理式に内在している非決定性を利用したAND並列・OR並列を代表とする種々の方式が提案され，活発に研究が進められている．また，言語仕様の中に陽に実行中断・再開といった同期機構を導入した並行論理型言語もICOTなどにより研究が進められている．

これらの中でOR並列は，実行可能な複数の候補節を並列に呼び出して実行していくもので，探索問題などで特に効果を発揮する．しかし，単純に全ての呼び出しを並列に行なうと，並列度が爆発的に増え，プロセス切替えなどのオーバヘッドの増大により台数効果が抑えられてしまう．

そこで，本稿では，OR並列を制御するための新しい機構を提案する．並列に実行されるゴール間に実行優先度を表すプライオリティという概念を導入し，高いプライオリティのものから並列に実行する。これにより実行時の探索空間を不必要に広げずに最適解に近いゴールのみを選択して実行できる．プライオリティは組込み述語により設定され，値としては任意の整数値をとることができる．

プライオリティ制御機構を持つOR並列Prologの応用として分子系統樹(molecular phylogenetic tree)推定問題をとり上げた．分子系統樹とはDNAやRNAの塩基配列など分子レベルでのデータを使用してつくられた生物または遺伝子の系統樹である．

以下ではOR並列の制御方式と，その処理系の並列計算機上での実現について述べ，分子系統樹の推定に適用した結果について述べる．

## 2 プライオリティによるOR並列の制御

Prologの実行過程は，ユーザが入力した最初のゴールをルートとして，実行の各時点でのゴールを節点とする探索木の展開ととらえることができる(図1)．

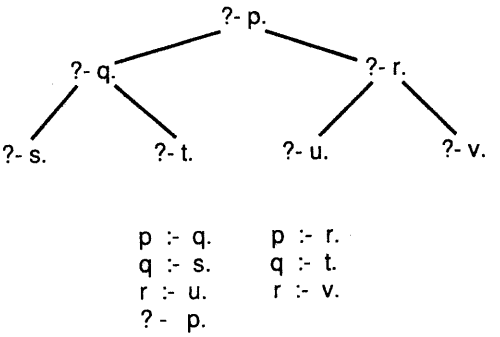


図1: 探索木

各節点から下向きに出る枝は，その節点のゴールと単一化可能な候補節を表し，探索木の一番下の葉は全てのゴールのリダクションに成功，あるいはこれ以上ゴールのリダクションができずに失敗したことを表す．成功した結果得られた変数の束縛値がユーザが入力したゴールの解となる．

OR並列では，この探索木を，ルートから幅優先に並列に展開する．OR並列は，与えられた条件を満たす解を全て求めるような問題で特に有効であるが，探索木の分岐ごとに単純に並列にゴールを実行していたのでは，ゴールの数が解くべき問題の並列度を越えて増加してしまう恐れがある．そこで，プログラム中で並列実行すべき節を宣言し，それ以外の節は逐次に実行することにしている．同様の宣言はAurora[1)]など多くの並列Prolog処理系でとり入れられている．

しかし，ある評価関数を最大に(または最小に)する最適解を求めるようなときには，通常の逐次Prologと同様`setof`，`bagof`といった組込み述

語により解の集合を求めた後，それらの中から最適解を選択することになる．探索木が非常に大きく広がるような問題では結果的に無駄な処理を数多く行なう可能性が高く，実行時間，メモリ消費ともに膨大なものとなってしまう恐れがある．

そこで本稿では，OR並列での効率の良い最適解探索のため，プライオリティによる実行制御を提案する．プライオリティは，概念的には探索木の各枝についたラベルであり，その枝の下のゴールの実行の優先度を表す．全体の実行はスケジューラにより管理され，プライオリティの高い枝から順に選択されて，その下のゴールが実行される．これにより探索木の展開は理想的には常に最適解の方向に向かうことになり，無駄な展開を抑えることができる．

プライオリティは次の3つの組込み述語により設定される．

```
setPriority(プライオリティ値)
upPriority(プライオリティ増加値)
downPriority(プライオリティ減少値)
```

setPriorityは引数の値(整数)を新しいプライオリティとして設定する．upPriority，downPriorityはそれぞれ現在のプライオリティに引数の値を加えるかまたは引いたものを新しいプライオリティとして設定する．これらを使って，プログラムは以下のように記述される．

```
ヘッド :-
    評価関数計算，プライオリティ設定，ボディ．
```

節の呼出しごとにその時点での評価関数の値が計算され，プライオリティが設定される．プライオリティは新たに設定されるまで以前の値を引き継ぐことにしているので，全ての節で評価関数計算とプライオリティ設定を書く必要はなく，書かなかった場合は節を呼び出したゴールのプライオリティがそのままボディゴールのプライオリティとなる．

# 3 並列計算機上での実現

## 3.1 並列計算機の構成

処理系の実現を，Sequent社製の並列計算機Symmetry S81上で行った．Symmetryは，要素プロセッサとしてIntel 80386(クロック16MHz)を使用した共有メモリ型の並列計算機である．使用した並列計算機では28台のプロセッサが実装されている．

## 3.2 実行モデル

本処理系では，**PE(Prolog Engine)**とタスクという2つの概念でPrologプログラムを並列に実行する．PEとは逐次Prolog処理系と同様，プログラムを深さ優先で逐次に実行するソフトウェア仮想マシンである．PEはDYNIXから見ればユーザプロセスの1つであり，実行に先だって複数個生成されて，次に述べるタスクを処理していく．DYNIXではユーザプロセスの数が少ない時には，プロセスとプロセッサとは1対1に対応するので，その場合PEをプロセッサと見なすことができる．

タスクはゴールの逐次的な実行過程である．2節で述べたように，並列実行されるゴールの数を抑えるため，並列宣言で宣言されていない節は全て1つのタスクとして逐次的に実行する．並列宣言されている節の実行では節の数に応じて複数のタスクが生成され，プライオリティでソートされた実行可能キュー(Ready Queue)につながれる．

タスクのPEへの割当ては，グローバルなスケジューラではなく個々のPEにより行なわれる．PEは(1)タスクが割り当てられていない，(2)割り当てられたタスクを完了した，(3)割り当てられたタスクより高いプライオリティを持つタスクが新たに実行可能キューにつながれた，のいずれかで，実行可能キューから新たにタスクを獲得してそれを実行する．なお，(3)の条件は各PE

により節の呼出しおよび2節のプライオリティ設定時にチェックされる．

この実行モデルはAurora[1]で採用されているSRIモデルとほぼ同じものであるが，(a) 変数束縛がSRIモデルでは束縛アレイであるのに対して本モデルでは分割スタック (後述) である，(b) タスク切替えのタイミングがSRIモデルでは基本的にタスク終了時であるのに本モデルではプライオリティ制御により節の呼出しごとに起こり得る，などの点が異なっている．

### 3.3 処理系の実現

タスクの実現のため，各タスクの実行に必要な情報が**TCB**(Task Control Block) として設定される (**表1**)．

表1: TCB内の情報

| |
|---|
| 次のタスクのTCBへのポインタ (実行可能キューにつながれたとき) |
| 親タスクのTCBへのポインタ |
| 兄弟タスクのTCBへのポインタ |
| 割り当てられたスタックのベースアドレス |
| 子タスクの数 |
| 実行状態 |
| PEのレジスタ退避領域 |
| プライオリティの値 |

タスク切替えは前節で述べたタイミングで生じるが，その操作は，(1) 実行可能キューの先頭にあるTCBを1つ獲得，(2) PEのレジスタの値を現タスクTCB内のレジスタ退避領域に保存，(3) 現タスクのTCBを実行可能キューにつなぐ，(4) 獲得したTCBのレジスタ退避領域の値をPEレジスタに設定の順に行なわれる．

PEはWAM[2]を拡張する形で実現されている．Prologプログラムは WAMコードを拡張した中間コードにコンパイルされる．中間コードはエミュレータで実行されるのではなく，各命令をCの関数とみなしてさらにCコンパイラによりコンパイルされ，PEの起動，タスクのスケジューリングなどを行なうランタイム・モジュールとリンクされて直接実行可能なオブジェクトに変換される．以下にWAMからの拡張点を示す．

**para_try命令** WAMのtry命令をOR並列用に拡張したもの．並列宣言された複数の候補節を呼び出す時実行され，(1) 並列選択点の作成，(2) 候補節の数だけの子タスクの生成と実行可能キューへの登録，(3) 現在のタスクの切離し (全ての子タスクが終了するまで実行を中断する) と次のタスクの獲得を行なう．

**unite命令** 同一の親から生成された複数のタスクを統合する．あるタスクが並列選択点を越えてバックトラックする時，そこでは複数のタスクが生成されたはずなので，単純にバックトラックすることはできない．そこで，このunite命令により，(1) 中断している親タスクのTCB中の子タスクの数を減らしていき，(2) 自分が最後の子でなければタスクの消滅，最後であれば並列選択点を除去しさらにその前の選択点までバックトラックする．

### 3.4 メモリ管理

Guptaと Jayaraman[3]は，OR並列では，(a) 束縛環境の生成，(b) 変数アクセス，(c) (新たなタスク生成による) タスク切替え，の3つを同時に定数時間で抑えられるようなメモリ管理方式が存在しないことを証明した．さらに本処理系ではプライオリティ制御を行なうので，この3つ以外に，(d) プライオリティの変更によるタスク切替えが生じる．これらのトレードオフをどのように選ぶかによって処理系が性格付けられることになる[4]．

本処理系では，メモリ管理については分割スタックを使ったコピー方式をとっている (**図2**)．

メモリは固定長のブロックに分割され，タスクの実行に必要な領域はこのブロックを最初に必要な個数だけ割り当てることによって確保される．スタックなど実行時に動的に増加する領域につ

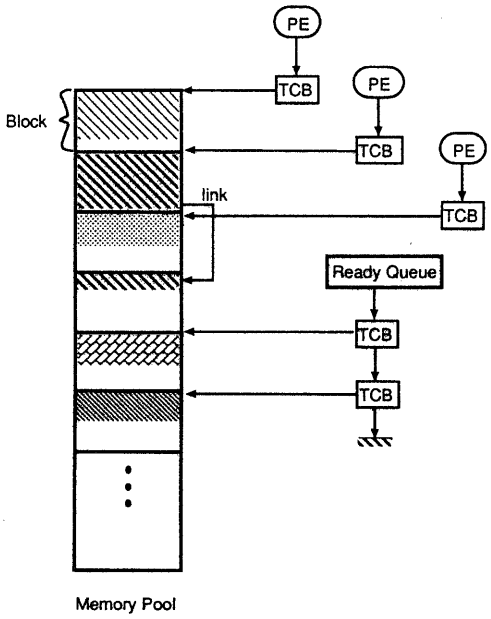


図 2: 分割スタックによるメモリ管理

いては，それがブロックの境界からあふれると，新たなブロックを確保しそこへリンクを張って領域拡張を行なう．

コピー方式では，新たなタスクを生成する場合，束縛環境や選択点などの情報を探索木のルートから全てコピーする必要があるので上述の (c) が定数時間内に行なえないが，他の 3 つは定数時間内に行なえる．本処理系の特徴である (d) が定数時間内に抑えられるようにすることが重要と考え，この方式を選んだ．

## 4 分子系統樹の推定

プライオリティ制御付き OR 並列の応用として分子系統樹の推定を行なった．推定アルゴリズムとしては Felsenstein のアルゴリズム[5]を OR 並列実行用に修正したものを使っている．

Felsenstein のアルゴリズムでは，系統樹は図 3のような無根木 (unrooted tree) として求められる．図 3は 5 種の生物からなる系統樹を表しており，$T_1$から$T_5$までの葉ノード (図の白丸) が生物を，$I_1, I_2, I_3$の中間ノード (図の黒丸) がこれらの生物の共通の祖先を，$v_1$から$v_7$までの枝が遺伝的な距離 (ここでは突然変異による遺伝データの置換数の期待値) を表す．無根木は，通常の系統樹と違って，根すなわち全ての種の共通の祖先が表されていないが，これは Felsenstein のアルゴリズムでは個々の生物の系統樹上の位置と遺伝的な距離のみしか推定できないためである．

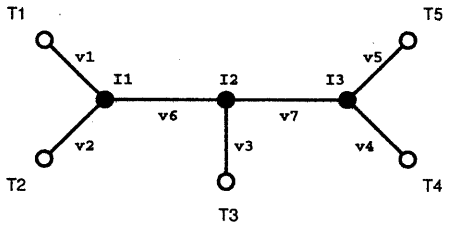


図 3: 無根木で表された系統樹

Felsenstein のアルゴリズムは，基本的には次のように推定を行なう．

**Step 1** 与えられた種を組み合わせて，可能な樹のトポロジを生成する．

**Step 2** それらのトポロジについて確率モデルをもとにその系統樹の形に各遺伝データが配置される確率を尤度とし，それが最大になるように樹の各枝の長さを計算する．

**Step 3** 尤度の最大値はトポロジによって異なるので，それらの中で最も大きい値をとるものを選ぶ (これが最終的な樹となる)

Step 1 のトポロジの数は単純に生成すると，種の数 $n$ のとき，

$$\prod_{k=3}^{n}(2k-5) = \frac{(2n-5)!}{2^{n-3}(n-3)!} \qquad (1)$$

という膨大な数になる．そこで，Felsenstein のアルゴリズムでは，この Step 1, 2, 3 を全ての種について一度に行なうのではなく，$k = 3$ つまり

3つの種からなる樹 (これは $2k-5=1$ より一意に決まる) から始め，一つずつ種を枝に付加して増やした中間的な樹について繰り返し行なう．例えば，図3の樹は $k=5$ の時を表しているが，これから次の $k=6$ の樹を作るときには6番目の種を $v_1$ から $v_7$ までの枝のいずれかに付加した7個のトポロジが生成される．

Step 2 での計算は，尤度が最大となるように各枝の長さ $v_i(i=1,\cdots,7)$ を設定していく．すなわち，

$$\frac{\partial L}{\partial v_i}=0 \quad \text{for } i=1,\cdots,7 \tag{2}$$

という連立方程式を Newton-Raphson 法により解いている．ここで $L$ は尤度を表す．

尤度は以下のように与えられる．各ノードの遺伝データの位置 $j$ についての尤度 $L(j)$ は，図3の樹では次式で与えられる．

$$L(j)=\sum_{t_1=1}^{u}\sum_{i_1=1}^{u}\sum_{i_2=1}^{u}\sum_{t_3=1}^{u}\pi_{t_1}P_{t_1i_1}(v_1)P_{i_1t_2}(v_2)\cdot$$
$$P_{i_1i_2}(v_6)P_{i_2t_3}(v_3)P_{i_2i_3}(v_7)P_{i_3t_4}(v_4)P_{i_3t_5}(v_5)$$

ここで，$t_1,\cdots,t_5,i_1,i_2,i_3$ はそれぞれ，位置 $j$ における $T_1,\cdots,T_5,I_1,I_2,I_3$ の遺伝データであり，その種類に応じて1から $u$ までの値 (例えばDNAの場合 A, T, G, C で $u=4$) をとる．$\pi_{t_1}$ は $t_1$ がその値をとる確率である．遺伝データが完全であれば，ある $t_1$ で1，それ以外は0となるが，あいまいさがあるときでも確率で与えられるようになっている (例えば，$t_1=A$ であれば $\pi_A=1$ でそれ以外の $\pi_T,\pi_G,\pi_C$ は0)．$P_{ij}(v)$ は距離 $v$ だけ時間が経過するうちに，遺伝データが $i$ から $j$ に変異する確率である．Felsenstein はこれを次式で与えている．

$$P_{ij}(v)=\begin{cases} e^{-v}+(1-e^{-v})\pi_i, & \text{for } i=j \\ (1-e^{-v})\pi_i & \text{for } i\neq j \end{cases}$$

全体の尤度 $L$ は各ノードの遺伝データの長さを $m$ とすると次式で与えられる．

$$L=\prod_{j=1}^{m}L(j)$$

Step 3 で尤度が最大の樹を1つだけ選ぶので，生成されるトポロジの数は，

$$\sum_{k=3}^{n}(2k-5)=n^2-4n+4 \tag{3}$$

にまで減らすことができる．

しかし，これは式 (1) だけある可能性を式 (3) に抑えているため，最終的に必ずしも尤度が最大の樹が得られるとは限らない (ある時点で生成された中間的な樹の尤度の大小関係が，それらに次の種を加えると逆転することがありうる)．このため，Step 3 の後，局所再配置 (local rearrangement) という処理を行なう．

局所再配置とは以下のような処理をいう．まず，中間ノードの中で，それが持つ3本の枝のうち少なくとも2つの枝が別の中間ノードに接続されているようなもの (図3では $I_2$) のところで樹を切り，中間ノードに接続されている2本の枝 ($v_6$ と $v_7$) をつなぐ．そして，残りの部分樹 ($T_3$ と $v_3$) を今つないだ枝以外の枝 ($v_1,v_2,v_4,v_5$) にそれぞれつないだ時の尤度の値をそれぞれ求め，もとの樹と比べて尤度が向上するかどうか見る．

$I_2$ のような中間ノードは $k-4$ ($k$ はその段階での種の数で $k\geq 5$) 個あるが，局所再配置は尤度が向上する間だけ行なうので，その回数は1から最大 $k-4$ の間で実行時に決まる．また，1回の局所再配置で生成されるトポロジの数は，最大で $2k-6$ となる．このように，局所再配置は，個々の $k$ の値ごとに一般に $k^2$ に比例した時間がかかることになる．従って，Felsenstein のアルゴリズムは実際には $n^3$ に比例した処理時間を要する[6]．

著者等は，OR 並列 Prolog により，中間的な系統樹を並列に生成し，それらの尤度に基づいてプライオリティを設定する方法により高速な系統樹の生成を実現した (プログラムを付録に示す)．Felsenstein のアルゴリズムと同様，種を加えながら可能な樹のトポロジを生成し，それらをいったん `TreeBuffer` という領域に書き込む (`makeDenovoTree` 内の `c_buildNewTip`)．

その後，個々のトポロジについて尤度を計算する (treeEval内のc_treeEvaluate)．treeEvalはプログラムの先頭の:- paraにより並列宣言されているのでOR並列により尤度を並列に計算する．その後，Felsensteinのアルゴリズムのように尤度最大のトポロジを1つだけ選ぶのではなく，尤度の値にある重みを足したものをゴールのプライオリティとして設定する (setPriority)．

プライオリティ制御機構により，尤度の小さい樹の処理は後回しにされるため，理想的な状況では式 (1) よりもはるかに少ない数の樹を調べるだけで最終的に尤度が最大の樹が最初に求まるはずである．しかし，実際には，それでも生成される樹の数が爆発的に増加することがあるので，プライオリティの値がある値以下になれば強制的にそのタスクを失敗させ，消滅させるようにしている (limitSearch)．

尤度に重みを加えたものをプライオリティとし，ある境界値でタスクを強制終了させているので，元のFelsensteinのアルゴリズムと同様，このプログラムで得られた結果は必ずしも大域的に最適とは言えない．今のところ重みの値は入力データなどから経験的に決めている．

なお，付録のc_で始まる組込み述語は，この応用のために，C言語で書かれた組込み述語である．系統樹の個々の枝の長さや尤度の計算は大量の数値計算を必要とするので，C言語で記述しPrologからは組込み述語として使用している．これらの組込み述語の実現にはFelsensteinのアルゴリズムに基づいてイリノイ大学で作成されたプログラム[6]を共有メモリ型並列計算機向けに修正したものを使用している．

付録のプログラムに20種の古細菌 (Archaeabacteria) の塩基配列を入力データとして与え，実行したときの時間と台数効果を表2に示す．元の文献[6]のプログラムを1台のプロセッサで実行したときの時間は，37,738秒であった．表2からわかるように，プロセッサ台数の大きいところで線形な値以上の台数効果が得られた (プロセッサ20台のとき1台と比べ34倍，文献[6]のプログラムの実行時間と比べても29倍)．これはプライオリティ制御により探索木の枝刈りが行なわれ探索効率が向上したためと考えられる．

表2: 分子系統樹生成の実行時間

| PE台数 | 1 | 2 | 4 | 8 | 16 | 20 |
|---|---|---|---|---|---|---|
| 実行時間 | 44.4 | 22.3 | 10.0 | 5.0 | 1.6 | 1.3 |
| 台数効果 | 1.0 | 1.9 | 4.4 | 8.9 | 28 | 34 |

(注) 実行時間の単位は1000秒

## 5 おわりに

本論文ではOR並列実行をプライオリティにより制御する方式を提案し，その処理系の共有メモリ型並列計算機上での実現について述べた．応用例として，分子系統樹の推定問題をとりあげ，本方式によるプログラミングと実行時間を示した．それによると，プロセッサ台数の大きいところでは，プライオリティ制御による探索効率の向上から線形以上の台数効果が得られ，本方式の有効性が示された．

**謝辞**

遺伝データおよび本研究の基礎となった分子系統樹推定プログラムを提供いただき，数々のご教示をいただいたイリノイ大学Carl Woese教授，Gary Olsen博士，アルゴンヌ国立研究所Ross Overbeek博士に感謝いたします．

## 参考文献

1) Lusk, E. et al.: The Aurora OR-Parallel Prolog System, *New Generation Computing*, Vol.8, No.7, pp.243-271 (1990).

2) Warren, D. H. D.: An Abstract Prolog Instruction Set, SRI Technical Note 309 (1985).

付録 分子系統樹生成プログラム

```
:- para treeEval/8.

go(WeightList,Limit,Likelihood) :-
  c_init, c_allocTreeArea(NumSpec),
  c_buildFirstTree(NumTips,Tree),
  makeDenovoTree(WeightList,Limit,NumTips,
                 NumSpec,Tree),
  c_freeTreeArea.

makeDenovoTree([Weight|WeightList],Limit,
               NumTips,NumSpec,Tree) :-
  NumTips<NumSpec,
  c_buildNewTip(Tree,TreeBuffer,NumTrees),
  NumTips1 is NumTips+1,
  treeEval(Weight,Limit,NumTips1,TreeBuffer,
           0,NumTrees,NewTree,Likelihood),
  makeDenovoTree(WeightList,Limit,NumTips1,
                 NumSpec,NewTree).
makeDenovoTree(_,_,NumTips,NumSpec,Tree) :-
  NumTips>=NumSpec, c_showBestTree(Tree).

treeEval(Weight,Limit,NumTips,TreeBuffer,TreeId,
         NumTrees,Tree,Likelihood) :-
  TreeId<NumTrees, c_allocTreeArea(_),
  c_treeEvaluate(TreeBuffer,TreeId,Tree,Likelihood),
  Priority is Likelihood+Weight,
  limitSearch(Limit,Priority), setPriority(Priority).
treeEval(Weight,Limit,NumTips,TreeBuffer,TreeId,
         NumTrees,Tree,Likelihood) :-
  TreeId<NumTrees, TreeId1 is TreeId+1,
  treeEval(Weight,Limit,NumTips,TreeBuffer,TreeId1,
           NumTrees,Tree,Likelihood).

limitSearch(Limit,Priority) :- Priority<Limit, !,
  fail.
limitSearch(_,_).
```


3) Gupta, G. and Jayaraman, B.: On Criteria for Or-Parallel Execution Models of Logic Programs, In *Proc. of NACLP'90,* pp.737-756 (1990).

4) 市吉伸行: 論理型言語の並列処理方式, 情報処理, Vol.32, No.4, pp.435-449 (1991).

5) Felsenstein, J.: Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach, *J. of Molecular Evolution,* Vol.17, pp.368-376 (1981).

6) Olsen, G., Woese, C., Hagstrom, R., Matsuda, H. and Overbeek, R.: Inference of Phylogenetic Trees Using Maximum Likelihood, In *Proc. of the First Delta Application Workshop,* pp.247-262 (1992).