Average case complexity analysis of RETE pattern-match algorithm and average size of join in databases

Luc Albert

▶ **To cite this version:**

Luc Albert. Average case complexity analysis of RETE pattern-match algorithm and average size of join in databases. [Research Report] RR-1010, INRIA. 1989. inria-00075548

HAL Id: inria-00075548
https://inria.hal.science/inria-00075548

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél (1) 39 63 55 11

Rapports de Recherche

N° 1010

Programme 1

AVERAGE CASE COMPLEXITY ANALYSIS OF RETE PATTERN-MATCH ALGORITHM AND AVERAGE SIZE OF JOIN IN DATABASES

Luc ALBERT

Avril 1989

RR_1010

Average case complexity analysis of RETE pattern-match algorithm and average size of join in Databases

Luc ALBERT *

Abstract. *The* RETE *algorithm [Forg 82] is a very efficient method for comparin g a large collection of patterns with a large collection of objects. It is widely used in rule-based expert systems. We studied ([AF 88] or [Alb 88]) the average case complexity of the* RETE *algorithm on collections of patterns and objects with a random tree structure. Objects and patterns are often made up of a head-symbol and a list of variable or constant arguments (OPSV [Forg 81], Xrete [LCR 88] ...). In this paper, we analyse the theoretical performance of* RETE *algorithm on th is widely used type of pattern and object with the theory of generating functions. We extend this work to the study of the performance of composed queries in relational Databases and we generalize Rosenthal's theorem on the average size of an equijoin [Rosen 81]. We give some numerical examples based on our results.*

Complexité en moyenne de l'algorithme de multi-pattern matching RETE sur des ensembles de patterns et d'objets de profondeur un

Résumé. *L'algorithme RETE ([Forg 82]) est un algorithme très efficace pour effectuer le pattern-matching (ou semi-unification) d'un grand nombre d'objets avec un grand nombre de motifs (ou patterns), notamment dans les moteurs d'inférences des systèmes experts à Base de Règles. Nous avions étudié ([AF 88] ou [Alb 88]) la compléxité en moyenne de l'algorithme RETE sur des ensembles de motifs et d'objets de structure arborescente quelconque. Objets et motifs sont souvent du type : un symbole de tête suivi d'une liste d'arguments constants ou variables (OPSV [Forg 81], Xrete [LCR 88] ...). Nous analysons, dans cet article, sa performance théorique pour ce type répandu de motifs et d'objets à l'aide de la théorie des séries génératrices. Nous étendons ces travaux à l'étude de la performance des requètes multiples dans les Bases de Données Relationnelles et généralisons le theorem de Rosenthal sur la taille moyenne d'une équijointure [Rosen 81]. Nous développons aussi numériquement sur des exemples les résultats simples et précis obtenus.*

* Institut National de Recherche en Informatique et Automatique, Domaine de Voluceau, Rocquencourt, BP 105, 78150 Le Chesnay Cedex France. mail : albert@inria.inria.fr

Laboratoire Central de Recherches, Thomson-CSF, Domaine de Corbeville, BP 10, 91401 Orsay Cedex France.

I. Introduction

The RETE pattern match algorithm [Forg 79] [Forg 82] has been introduced by C. Forgy in the line of his work on production systems [Forg 81].

Production systems, or more generally rule-based systems, are widely used in Artificial Intelligence for modelling intelligent behaviour [LNR 87] and building expert systems. They are quite easy to use and have many advantages : modularity, relative independence of each rule and the same expressivity as a Turing machine. However the inference engine is algorithmically inefficient. The most time consuming process in a rule-based system is the *pattern match* phase that consists of maintaining the set of satisfied rules among changes in the data base. This computation can represent more than 90 % of the overall computation time in an application [DNM 78].

RETE algorithm is an efficient method for computing the set of satisfied rules incrementally after each rule execution. The incremental computation is justified in expert systems applications by the fact that the execution of a rule affects a relatively small number of objects (or facts or terms) in comparison to the total number of objects. Therefore most of the previous pattern match work remains valid. RETE algorithm realizes a total indexing of the data base according to rule conditions. Conditions common to several rules are shared in such a way that several rules can be found to be satisfied by testing some patterns only once.

Forgy [Forg 79] proved, thanks to simplifying hypotheses, that with RETE algorithm the worst case time complexity for computing the set of satisfied rules is linear in the number of rules, and polynomial in the number of objects (with degree being the maximum number of conditions in a rule). In the best case the complexity is a constant. Between these extremes the sensitivity of pattern match time to the size of the data base is highly dependent on rule characteristics. We already studied the theoretical average case complexity of RETE algorithm when it compares objects and patterns having any tree structure. ([AF 88], [Alb 88]). In real applications, objects and patterns are often made up of a function symbol and a list of variable or constant arguments. The height of their usual tree structure is therefore one and we analyse in this article the average performance of the algorithm in the case of these "*flat terms*".

There are many motivations in searching for a more accurate model of computation and an average case complexity analysis of RETE algorithm. First, RETE algorithm admits many variants and optimizations, concerning the representation of local memories [Forg 79], the sharing of conditions (the ARBRE D'UNIFICATION : [Gha 87], [Alb 88]), the computation of joins [Mir 87], the total compilation principle [Fag 86] [fag 88], the parallelization of the algorithm [Gupt 84] etc ... An average case complexity analysis can be used to evaluate these optimizations and propose new ones. Second, run-time performance prediction is a necessity for the development of *real-time* expert systems [WGF 86], [SF 88]. A mathematical model of run-time requirements can be used to extrapolate from the run-time performance of a prototype the performances of the expert system in real size. It define the range of its applicability in terms of number of objects that can be treated at a given time. Third, a mathematical model can be used also to work out significant benchmarks in order to compare several implementations according to the relevant characteristic parameters of a knowledge base [GF 83]. Lastly, we shall show that all this analysis applies not only to the study of the performance of pattern-match in expert systems but also to the estimation of the average size of composed queries in Relational Databases. In this paper we present an average case complexity analysis of RETE algorithm and of the average size of composed queries in Relational Databases using the generating function theory ([Fla 85], [FS 86], [FV 87]) (that introduces mathematical methods of independent interest). One can find a more detailed version of this study in [Alb 89].

Only equality tests are considered first. In Part **2**, we briefly present RETE algorithm, and develop an Example. We define also the cost of this algorithm and precise the fundamental quantities for its computation. In Part **3**, we introduce the generating function theory that is used to analyse the average case complexity of algorithms **(3.2)** and the asymptotic analysis necessary to simplify the expressions we previously obtained **(3.3)**. Thus we obtain a result under a first model in which the Database is represented as an ordered list of terms **(3.4)**. In Part **4**, we determine the complexity under a second model that considers the Database as a multiset of terms. In Part **5**, we extend the previous results by considering several separate ranges of variation for constants **(5.1)**. Then we generalize our results for taking into account different frequency

coefficients for symbols **(5.2)** (by the way we shall consider inequality tests). In Part **6**, we consider the negation between arguments and between patterns.

Lastly in Part **7**, we apply all these results to the study of composed join in Relational Database. We illustrate the results we obtain throughout the article with some examples. The example of figure 1 is numerically developed in the appendix.

II. RETE algorithm

2.1. Presentation

The production systems we shall consider are composed of a fixed set, denoted by RB (for *Rule Base*), of *if-then rules* called *productions*, and a changing set of *facts*, called the *Working Memory* and denoted by WM. Facts are formed on a finite alphabet F of function symbols given with their arity. Arguments are taken in a finite set C of symbols of arity 0, the constants. For instance, given symbols h of arity 3, f of arity 2 and constants $a1$ and $a2$ one can form the following terms : $(f\ \ a1\ a1)$, $(f\ \ a2\ a1)$, $(h\ \ a1\ a2\ a1)$, $(h\ \ a1\ a2\ a2)$ etc ... The set of flat terms is denoted by $FT(F)$. The Working Memory is formalized as an ordered or unordered set of such terms.

The if-part of a rule (its *left-hand side*) is a conjonction of *patterns*, represented as a tuple $(P_1, \ldots, P_n)$. A pattern is a term some of whose arguments can be variable. Variables are denoted by $X, Y, \ldots$, they are taken from an enumerable set of variables V. Patterns are partial descriptions of facts. A pattern P *matches* a fact t if one can find a *substitution* of pattern's variables, $\sigma : V \rightarrow C$, such that $\sigma P = t$. For example the substitution of X by $a1$ and of Y by $a2$ in pattern $(h\ \ X\ Y\ X)$ matches the term $(h\ \ a1\ a2\ a1)$.

We say that the left-hand side of a rule $(P_1, \ldots, P_n)$ is *instanciated* (or that the rule is *satisfied*) when there exists a tuple of facts $(t_1, \ldots, t_n)$ with $t_i \in WM$, called the instance, and a substitution σ such that $\sigma P_i = t_i$. We remark that since a pattern in a rule can match several facts in the Working Memory, a rule can be instanciated in multiple ways. The then-part of a rule (its *right-hand side*) is a sequence of *actions* that can add (resp. remove) a fact in (resp. from) the Working Memory.

Example : Let h be a function symbol of arity 3, f and g of arity 2 and constants $a0$, $a1$, $a2$, $a3$, $a4$, $a5$. We can consider the three following rules, in which we omitted the right-hand side :

$$(R1 : \quad (g\ \ X\ Y)\ \neg(f\ \ a0\ X)\ (h\ \ a1\ X\ Y) \longrightarrow \ \ldots)$$
$$(R2 : \quad (h\ \ a0\ X\ a1)\ (g\ \ X\ Y)\ (f\ \ a0\ a1) \longrightarrow \ \ldots)$$
$$(R3 : \quad (f\ \ X\ a1)\ (f\ \ Y\ a2)\ (h\ \ a2\ X\ Y) \longrightarrow \ \ldots)$$

(The negation "$\neg$" is studied in Part **6**.)

The cycle of inference consists in three steps:

- *match* the patterns with the facts in the Working Memory in order to determine the set of satisfied rules (called the *conflict set*) : this is the *pattern match* phase;
- *select* one rule 's instance in the conflict set;
- *execute* the actions of the rule.

In **RETE** algorithm the pattern match step is not separated from the execution step. In fact, it occurs at each modification in the Working Memory, that is at initialisation time when facts are entered, and then at each assertion or retraction during execution steps. Since the **RETE** multi-pattern match algorithm is solely concerned with the left-hand sides of rules, from now on we shall identify rules with their left-hand sides (we shall consider right-hand sides in **5.2** when we distinguish apparition frequencies of symbols). Rules are compiled in a discrimination network. At run-time when a fact is asserted or retracted, it is processed in the network from the root. If the fact matches a pattern it is memorized in (or removed from) the network. If other memorized facts can jointly satisfy a rule, the instance is added to (or suppressed from) the conflict set. In this way both rules and facts are represented in the network.

Two types of tests are distinguished in rule left-hand sides:

- *one-pattern tests* are tests concerning solely the fact to be characterized. They test the equality of the function symbol and of constants between the fact and the pattern. When a variable has several occurrences in the pattern, they test the equality of the corresponding constants of the fact.

- *multi-pattern tests* are the tests of several patterns belonging to the same left-hand side of a rule. When the same variable appears in several patterns, they check the equality of the corresponding constants of facts. They perform a *join* in the sense of Databases (*i.e.* they produce as output the cartesian product of input data discriminated with tests).

In RETE pattern-match algorithm, one-pattern tests are executed first, then multi-pattern tests are executed at each join with memorized facts to verify consistent binding of variables across multiple patterns in a rule's left-hand side. Consequently the network is formed of two parts.

Figure 1 represents the RETE network corresponding to the three previous rules.

The first part is a tree composed of one-pattern tests, called the *discrimination tree*. In this tree each node is labelled with a single one-pattern test. The discrimination tree has as many leaves as the number of different patterns in the left-hand sides of the rules; that is 8 leaves in our example. (two patterns being equal up to variable renaming). We define the *i-pattern* of a node i as the pattern corresponding to the tests cumulated on the path from the root to node i (*i.e.* the pattern needed to reach node i). The root-pattern is a variable, *i.e.* a non-selective pattern (it matches any fact). The i-pattern of the leaves of the discrimination tree are all the patterns which appear in the rule left-hand sides. Several successors to a node correspond to several branches to follow. At run-time one fact typically reaches several leaves which correspond to different matching patterns. Thus in our example, the pattern $(h\ a0\ X\ Y)$ corresponds to node **8** and term $(f\ a0\ a1)$ reaches nodes **4**, **5** and **10**.

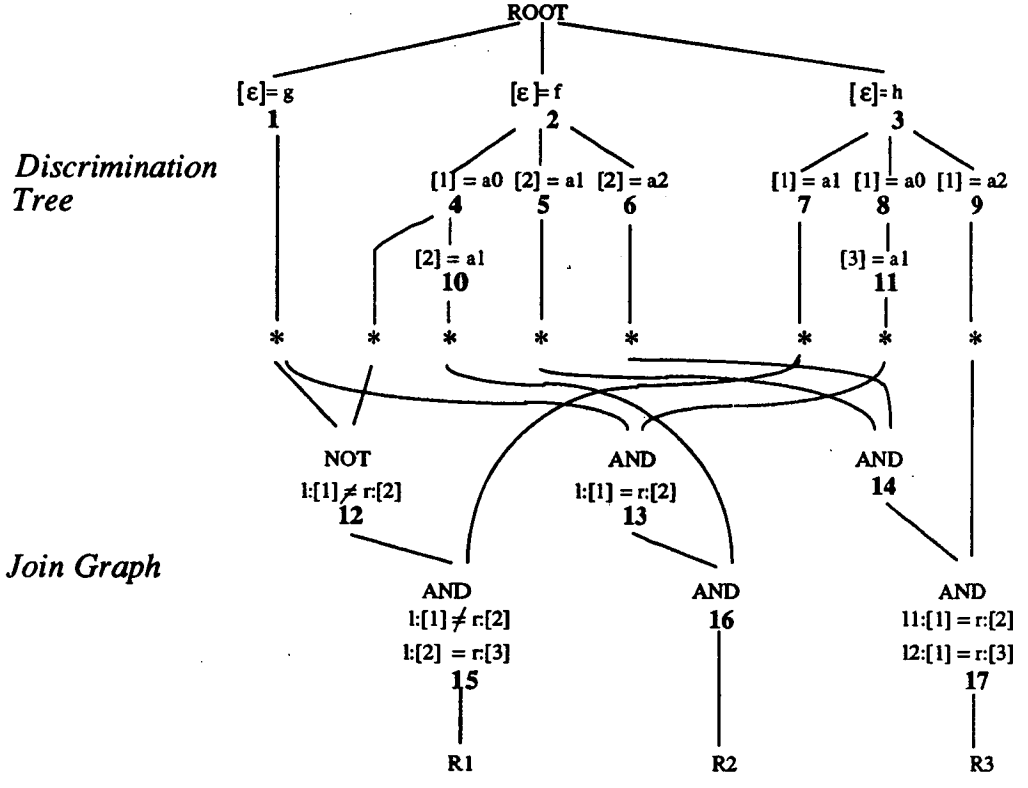


Figure 1 † : Rete Graph

† We denote by [j] the j^{th} leftmost son of the root ([ε]) of the i-pattern ; r (resp. l) denotes the pattern of the right (resp. left) input of a join node (lj precise the j^{th} pattern of the left input list).

The second part is a graph formed with binary joins between the leaves of the initial part. In these nodes, i-patterns are tuples of patterns $(P_1, \ldots, P_i)$, in which the sharing of variables between patterns is determined by the multi-pattern tests. Outputs are in a one-to-one correspondence with the rules. i-pattern of output nodes are the left-hand sides of the rules up to variable renaming. They are reached at run-time with the instances of the rules.

At run-time when a tuple reaches a join node, it is memorized in a *local memory* or *memory node*, and the memory of the opposite input of the join node is searched in order to find a compatible tuple according to the multi-pattern tests. Thus at node **17** of figure 1, are memorized in left input, pairs of the type $(f\ X\ a1)(f\ Y\ a2)$, and in right input $(h\ a2\ U\ V)$ and we select as output triplets with the following multi-pattern tests $X = U$ and $Y = V$. In many implementations the right input of a join node is restricted to be a leaf of the initial tree (not the output of a join node) as in figure 1, so that the structure of the join network is a set of *combs*. This eliminates a possibility of optimization consisting of grouping together the most selective patterns together (we shall study the general case to perform calculations).

Hashing techniques can be used to reduce the complexity of the search in local memories, by exploiting all equality tests in a join in order to hash the input memories. In the same way, if inequality predicates are to be considered, the local memories can be organized in search trees.

Negation of patterns has not been considered yet. A negated pattern expresses the non-existence of a fact matching the pattern. Negated patterns are treated in **RETE** algorithm by adding a second type of join nodes (see node **12**). In these nodes the local memory on the left input stores the number of elements in the right memory (the negated elements) which are compatible with each "left-tuple" according to the multi-pattern tests. When a counter gets to zero (resp. one) the left tuple is propagated in the network in assert (resp. retract) mode. In our example, as long as there is no fact of the type $(f\ a0\ X)$ in the Working Memory, the fact reaching the left input of the node **12** reaches the node **15**. The existential quantifier can be treated in a symmetrical way. In fact it is possible to generalize **RETE** algorithm in order to accept in the left-hand sides of rules any first order logic formula with an arbitrary degree of imbrication of quantifiers ([Fag 88]).

2.2. Cost of the algorithm

The **RETE** algorithm spends most of its time performing the pattern match step. We are going to determine the average cost of the algorithm by considering an arbitrary Working Memory and a given **RETE** network. We shall do an average calculation over all the possible input-Working Memory. More precisely we shall determine the average time needed by the algorithm to compute the conflict set from an arbitrary input-Working Memory. For the facts, the time devoted to move through the discrimination tree is negligible when compared to the time devoted to joins. We shall therefore determine the average time devoted to perform the multi-pattern tests at the join nodes. Thus our complexity measure will be the time of performing one multi-pattern test. At a join node, the global average time is proportional to the average number of tuples of facts to be tested at this node. When there is no hashing at the join nodes, this quantity is the product of the average number of tuples stored at the memory nodes in left and right input of the join node (the product of the average *size* of the memory nodes). When local memories are hashed according to equality tests performed at the join node, the average number of tuples of facts to be tested coincide with the average number of tuples that output this node with success; that is the average size of the output local memory.

The average size of memory nodes thus appears to be a fundamental quantity to calculate. We shall thus be able to determine the average cost of the algorithm and the precise average size of each local memory.

We can model the input-Working Memory as an ordered list of facts. This is a good representation of the succession of the inputs of the **RETE** network at each cycle of the inference engine. This yields the results for the *list-model*. Since the final state of the local memories is independent of the order of adding of facts of the input-Working Memory, we can also model the Working Memory with a multiset of facts (we always enable repetitions of facts in order to take into account the possibility for the same fact to be added and suppressed several times during the cycle of the inference engine). This yields the results for the *multiset model*.

III. List model

3.1. Combinatorial study

Let c_0 be the number of constants. Let c_j denotes the number of function symbols of arity j in F. The arity of the function symbols lies between m (the minimal arity) ($m > 0$) and M (the maximal one). We define the size $|t|$ of a term t as the number of the arguments of the function symbol minus m (the arity of its function symbol minus m). We associate with $FT(F)$ the characteristic generating function:

$$A(z) = \sum_{t \in FT(F)} z^{|t|} = \sum_{i=0}^{M-m} a_i z^i$$

with a_n the number of terms of size n in $FT(F)$. It is easy to see that $a_i = c_{i+m} c_0^{i+m}$. Since the number of function symbols and the number of constants are finite, the number of terms in $FT(F)$ is finite too and $A(z)$ is a polynomial with degree $M - m$. Remark that $a_0 \neq 0$. We can introduce the average size of terms in $FT(F)$, that is

$$\frac{1}{\lambda} = \frac{A'(1)}{A(1)}$$

Remark : In the case where $m = M$, we have $\frac{1}{\lambda} = 0$!! It is easy to study directly this specific case and the result we obtain is only a particular case of the formula described later. From now on, we have $0 < \frac{1}{\lambda} < M - m$.

Example : In our previous example, we have $c_0 = 6$, $c_1 = 0$, $c_2 = 2$ and $c_3 = 1$. Thus $m = 2$, $M = 3$ and $A(z) = 72 + 216z$ and $\frac{1}{\lambda} = 3/4$.

We consider that the RETE network takes as input an arbitrary Working Memory with k facts and of total size n (*i.e.* n is the sum of the sizes of terms composing the Working Memory). The generating function of a list of k terms determined by $A(z)$ is given by $A(z)^k$. Thus the n^{th} coefficient in $A(z)^k$, $[z^n]A(z)^k$, represents the number of lists of k terms with total size n.

For a node i of the discrimination tree, we introduce the generating function of the terms that match at i : $B_i(z) = \sum_{n \geq 0} b_{i,n} z^n$, with $b_{i,n}$ the number of terms of size n that match the i-pattern. We introduce some notations for a node i of the discrimination tree :

- ω_i denotes the arity of the function symbol of the i-pattern;
- x_i is the number of different variable argument of the i-pattern.

Example : for node **8** of the network of figure 1, we have $\omega_8 = 3$ and $x_8 = 2$. For the pattern $(Head\ X\ X\ Y\ a)$, we have $\omega_i = 4$ and $x_i = 2$.

It is now easy to prove that :

THEOREM 1 : *The generating function of the terms that match at a node i of the discrimination tree is :*

$$B_i(z) = c_0^{x_i} z^{\omega_i - m}$$

Example : We have at node **11** of figure 1 : $B_{11}(z) = 6z$. And we see that $B_{root}(z) = A(z)$.

Let us consider now a node i of the join network. That node outputs l-tuples of facts that partially instanciate the left-hand side of a rule. $B_i(z) = \sum_{n \geq 0} b_{i,n} z^n$ is the associated generating function. $b_{i,n}$ is the number of l-tuples of total size n that match the i-pattern. We know the l leaves j_i of the discrimination tree which match each of the components of the output-l-tuples of the node i. L_i stands for this set of the l associated patterns. Let us denote :

- $\omega_i = \sum_{j_i \in L_i} \omega_{j_i}$
- $x_i = \left(\sum_{j_i \in L_i} x_{j_i} \right) + y_i \qquad y_i \leq 0$

with y_i a correcting term due to the multi-pattern tests between variables of the l different patterns. *If we consider the global set of all the variables of the different l-patterns, x_i still represents the number of distinct variables in this set.* Thus we determine easily y_i in function of the similar variables tested at node i and we have: $y_i \leq 0$ because the join increase *a priori* the number of identical variables.

From this, we prove in the same way as previously:

THEOREM 2 : *The generating function of the l-tuples of terms which match at node i in the join network is:*

$$B_i(z) = {c_0}^{x_i} z^{\omega_i - lm}$$

From this, we deduce

THEOREM 3 : *The average number of terms that match at a node i of the discrimination tree is :*

$$\overline{b_{i,n,k}} = \frac{k{c_0}^{x_i}[z^{n-\omega_i+m}]A(z)^{k-1}}{[z^n]A(z)^k}$$

PROOF : The basic idea is to split the generating function $A(z)$ in: $A(z) = B_i(z) + R_i(z)$ with $R_i(z)$ the "rest function" *i.e.* the generating function of all the terms which don't match with the i-pattern. Then we shall mark the matching at i elements with the variable u and note:

$$A_i(z,u) = uB_i(z) + R_i(z) = A(z) + (u-1)B_i(z)$$

Thus $f_{i,n,p,k} = [u^p z^n]\left(A_i(z,u)^k\right)$ represents the number of lists of size n with k terms among which there is exactly p terms matching at node i.
Therefore $\overline{b_{i,n,k}}$ is the quotient of $\sum_{p=0}^{+\infty} pf_{i,n,p,k}$ $\left(= \sum_{p=0}^{k} pf_{i,n,p,k}\right)$ by the total number of lists of k terms and of total size n, that is

$$\overline{b_{i,n,k}} = \frac{\sum_p pf_{i,n,p,k}}{[z^n]A(z)^k}$$

We have $\left(A_i(z,u)^k\right) = \sum_{p,n} f_{i,n,p,k}u^p z^n$ thus :

$$\frac{\partial}{\partial u}\left(A_i(z,u)^k\right) = \sum_{p,n} f_{i,n,p,k}pu^{p-1}z^n$$

whence

$$\frac{\partial}{\partial u}\left(A_i(z,u)^k\right)\Big|_{u=1} = \sum_n \left(\sum_p pf_{i,n,p,k}\right)z^n$$

Therefore

$$\overline{b_{i,n,k}} = \frac{[z^n]\frac{\partial}{\partial u}\left(A_i(z,u)^k\right)\big|_{u=1}}{[z^n]A(z)^k}$$

And since

$$\begin{aligned}\frac{\partial}{\partial u}\left(A_i(z,u)^k\right)\Big|_{u=1} &= kA(z,u)^{k-1}\frac{\partial}{\partial u}\left(A_i(z,u)\right)\Big|_{u=1} \\ &= kA(z)^{k-1}B_i(z)\end{aligned}$$

the theorem is established. ∎

We shall now determine the average number of l-tuples of terms that match at a node i of the join network. Our demonstration will express the two antagonist sides of a join. We actually know the l nodes j_i of the discrimination tree "associated" to i, and thanks to theorem 3, we know the $\overline{b_{j_i,n,k}}$. The *expansion* side of a join can be expressed by the quantity $\overline{L} = \prod_{j_i=1}^{l} \overline{b_{j_i,n,k}}$, which represents (under the hypothesis of independent distribution laws at the j_i) the average number of l-tuples which would reach the node i if there were no multi-pattern tests. On the other hand a reasoning close to that of theorem 3 expresses the "selection" side of a join :

THEOREM 4 : *The average number of l-tuples of terms that match at a join node i from a list of total size n of L l-tuples of terms of which each component matches at an associated node j_i is :*

$$\overline{\gamma_{i,L,n}} = L\frac{[z^n]B_i(z)\left(\prod_{j_i=1}^{l} B_{j_i}(z)\right)^{L-1}}{[z^n]\left(\prod_{j_i=1}^{l} B_{j_i}(z)\right)^{L}}$$

(see [Alb 89]). From this it follows :

THEOREM 5 : *For a node i of the join network, the average number of l-tuples of terms that match the i-pattern is :*

$$\overline{b_{i,n,k}} = c_0{}^{y_i}\left(\prod_{j_i=1}^{l} \overline{b_{j_i,n,k}}\right) \quad (y_i \leq 0)$$

Those exact results could be studied using the Lagrange inversion theorem but they lead to expressions that are hard to use (multinomial coefficients ...). In order to express them simply we shall derive an asymptotic expansion with respect to k and n.

3.2. Asymptotic evaluation

To derive simple expressions, we use singularity analysis methods. We estimate the value of integrals with complex analysis methods (saddle-point method). We consider a Working Memory of size n with k terms as a given data of the algorithm. Thus we consider that k and n are both increasing.

According to the previous Section, we have to evaluate coefficients such as: $a_{n,k} = [z^n]A^k(z)$, from which we shall deduce the expression of $\overline{b_{i,n,k}}$. By Cauchy's theorem this quantity can be expressed as the following integral

$$a_{n,k} = \frac{1}{2i\pi}\int_{\Gamma}(A(z)^k)\frac{dz}{z^{n+1}}$$

where the contour Γ lies inside the domain of analycity of A and simply encircles the origin. In order to get an equivalent of this integral when k and n tend both to infinity, we shall use the saddle point method (See [dB 58] and [Henr 74]). In order to simplify this asymptotic analysis, we use the fact that, in real applications, the size of the terms is approximatively constant (say 4, 5, 6 ...). Thus, we can consider that $k = \lambda n$. The quite long calculations needed to obtain the following formula are detailed in [Alb 89] (similar ones are made in [Alb 88] and [AF 88]). We have :

$$a_{n,k} = \frac{A(1)^{\lambda n}}{2\sqrt{\pi}}\left(\frac{1}{2}(1-\frac{1}{\lambda}) + \frac{\lambda A''(1)}{2A(1)}\right)^{-\frac{1}{2}} n^{-\frac{1}{2}}\left(1+O\left(\frac{1}{n}\right)\right)$$

and this quantity has to be multiplied by δ when $A(z) = N(z^\delta)$ (HCF condition). From this we deduce (the constant δ disappears by cancelling the the $\overline{b_{i,n,k}}$ fraction) :

THEOREM 6 : *The average number of l-tuples of facts that match at a join node i is :*

$$\overline{b_{i,n,k}} = c_0{}^{x_i}\frac{1}{A(1)^l}k^l\left(1+O\left(\frac{1}{n}\right)\right) = c_0{}^{x_i}\frac{\lambda^l}{A(1)^l}n^l\left(1+O\left(\frac{1}{n}\right)\right) = \Pi_i n^l\left(1+O\left(\frac{1}{n}\right)\right)$$

with $l = 1$ when i is a node of the discrimination tree.

We can define now a matching-rate $\overline{\alpha_i}$ at a node i of the RETE network which expresses the probability for a term or a l-tuple of terms to match at a node i in the RETE network.

THEOREM 7 : *For a node i in the* RETE *network, the matching-rate is*

$$\overline{\alpha_i} = \frac{\overline{b_{i,n,k}}}{k^l} \simeq \frac{{c_0}^{r_i}}{A(1)^l}$$

Remark : Of course $\overline{\alpha_i} \leq 1$ and we can verify that at the first level of the discrimination tree (nodes **1**, **2** and **3** of figure 1 of our example), if we made the summation over all the function symbols of $FT(F)$, we find

$$\sum_i \overline{\alpha_i} = \sum_i \frac{c_i {c_0}^i}{A(1)} = 1 \ !$$

(a term of $FT(F)$ has always its function symbol in F).

3.3. Average cost of the algorithm with the list model

We can now evaluate the average cost of the **RETE** algorithm, given an arbitrary Working Memory and a fixed **RETE** network. We shall estimate the average time needed by the **RETE** algorithm to produce the conflict set from a given input-Working Memory. We determine the average number of multi-patterns tests realized at join nodes (this is as we previously said the main part of the total time cost). Let us recall that our complexity measure is the time of realization of one multi-pattern test. Since we know the average size of local memories in the **RETE** network, it is easy to determine the average number of those tests performed at the join nodes. Let us denote by l_i the number of components of the output-tuples at a join node i and β_i the number of multi-patterns tests performed at i considering only one l_i-input-tuple (in our example, $\beta_{17} = 2$, $\beta_{14} = 0$, $\beta_{13} = 1$ ...). The average number of tests performed at i is the average number of l_i-tuples tested at i multiplied by β_i. To obtain the average number of l_i-tuples that reaches i we just have to consider that at i no test is performed (*i.e.* $y_i = 0$) and evaluate the so modified $\overline{b'_{i,n,k}}$. Let us denote bye $\Pi'_i n^{l_i}$ this quantity (notice that this quantity is precisely equal to $\overline{b_{i1,n,k}} \times \overline{b_{i2,n,k}}$ where $i1$ and $i2$ design the two nodes inputs of i).

Then, we can propose

$$\overline{cost_n} \simeq K \sum_{i \ join-node} \beta_i \Pi'_i n^{l_i} = K \sum_{i \ join-node} \beta_i (\Pi_{i1} \Pi_{i2}) n^{l_i} \qquad (1)$$

with K an implementation constant.

Of course asymptotically with respect to n, we get the main part of this expression keeping only the nodes with the larger l_i. But the interest of an average case complexity analysis is the precise determination of all the proportionality constants, which can be, as we shall see in appendix, very small for large l_i, and therefore very important.

Remark : By hashing local memories according to equality tests at the join nodes, the average number of semi-unifications performed at a join node i is only the number of output-tuples *i.e.* $\overline{b_{i,n,k}}$. Thus in the above formula we just have to replace Π'_i by Π_i.

From this result, *that can be made fully precise on any case*, we can *a posteriori* assume Forgy's hypotheses (See [Forg 79]). Thus if we assume that the number of condition-patterns per left-hand side is constant equal to c in the Rule Base, the previous result becomes :

$$\overline{cost_{|WM|}} \simeq K A^c |WM|^c \left(\sum_{j=1}^{|RB|} \beta_j \Pi'_j \right) = K A^c |WM|^c \left(\sum_{j=1}^{|RB|} \beta_j \Pi_{j1} \Pi_{j2} \right)$$

with $j1$ and $j2$ denoting the two inputs of the last join node j of the Rj rule.
And we find the same type of relation proposed by Forgy: a linear cost as a function of the number of rules and a polynomial cost as a function of the size of the Working Memory $|WM|^c$ (we have determined the time needed by the **RETE** algorithm with an input of $|WM|$ modification terms, Forgy found a cost of $|WM|^{c-1}$ because he considered only one modification term as input).

IV. Multiset model

4.1. Combinatorial formulae

As we previously explained at the end of Section **(2.2)**, we shall now model the Working Memory as a multiset of facts. We shall use the "usual" definition of the size of a term : the number of its arguments plus 1 *i.e.* the number of nodes in its usual tree representation. Thus we have a new characteristic generating function :

$$P(z) = \sum_{t \in FT(F)} z^{|t|} = \sum_{n \geq 0} p_n z^n = \sum_{l=m+1}^{M+1} (c_{l-1} c_0^{l-1}) z^l$$

We can introduce the usual average size of a term

$$\frac{1}{\lambda_u} = \frac{P'(1)}{P(1)} \quad and \quad \frac{1}{\lambda_u} = \frac{1}{\lambda} + m + 1$$

Note that $A(1) = P(1)$ is the number of terms in $FT(F)$.

We consider as input of the **RETE** network, any Working Memory of total size n (without even considering the number of its terms). The generating function of multisets of terms of $FT(F)$ is :

$$M(z) = \prod_{t \in TP(F)} \left(\frac{1}{1 - z^{|t|}} \right) = \prod_{i=m}^{i=M} \left(\frac{1}{1 - z^{i+1}} \right)^{p_i}$$

Thus $[z^n]M(z)$ is the number of multisets of terms of total size n.

Example : In our example, we have

$$M(z) = \left(\frac{1}{1 - z^3} \right)^{72} \left(\frac{1}{1 - z^4} \right)^{216}$$

We keep on using the same notations $B_i(z)$, ω_i, x_i and y_i for a node i in the **RETE** network. We thus obtain :

THEOREM 8 : *The generating function of terms or l-tuples of terms that match at a node i is :*

$$B_i(z) = {c_0}^{x_i} z^{\omega_i + 1}$$

with $l = 1$ in the case of a node of the discrimination tree.

With a reasoning close to that of theorem 3, we obtain

THEOREM 9 : *The average number of terms that match at a node i of the discrimination tree is :*

$$\overline{b_{i,n}} = \frac{{c_0}^{x_i} [z^{n - \omega_i - 1}] \{ M(z) / (1 - z^{\omega_i + 1}) \}}{[z^n] M(z)}$$

For a join node i, the order of instanciation of the l local memories j_i is fundamental because it expresses the instanciation of different patterns. The reasoning of theorem 5 is valid and the average number of l-tuples of terms that match the i-pattern is still :

$$\overline{b_{i,n}} = {c_0}^{y_i} \left(\prod_{j_i=1}^{l} \overline{b_{j_i,n}} \right)$$

4.2. Asymptotic analysis

In order to simplify the expressions we have just obtained, we shall analyse specific fractions of the type $m_n = [z^n] \prod_j (1/(1 - z^j)^{p_j})$. The modulus of all the singularities of this kind of rational fraction is always 1. We shall consider the value of the multiplicity of 1 as singularity. We obtain ([Alb 89]) :

$$m_n = \delta \prod_j \frac{1}{j^{p_j}} \frac{n^{\sum_j p_j - 1}}{\Gamma(\sum_j p_j)}$$

with δ the highest common factor of the *arity* + 1 of the function symbols of $FT(F)$. And as before, δ disappears in the expression of $\overline{b_{i,n}}$.

4.3. Results with the multiset model

Thanks to previous formulae, we obtain for the multiset model :

THEOREM 10 : *For an arbitrary input-Working Memory of total size n, at a node i*

- *of the discrimination tree, the average number of matching terms is*

$$\overline{b_{i,n}} = {c_0}^{x_i} \frac{1}{P(1)} \frac{1}{\omega_i + 1} n \left(1 + O\left(\frac{1}{n}\right)\right) = \Phi_i n \left(1 + O\left(\frac{1}{n}\right)\right)$$

- *of the join network, the average number of matching l-tuples of terms is*

$$\overline{b_{i,n}} = {c_0}^{x_i} \frac{1}{P(1)^l} \frac{1}{\prod_{j_i=1}^{l} (\omega_{j_i} + 1)} n^l \left(1 + O\left(\frac{1}{n}\right)\right) = \Phi_i n^l \left(1 + O\left(\frac{1}{n}\right)\right)$$

with j_i the l associated nodes of the discrimination tree.

We still can consider that $k \simeq \lambda_u n$, and define :

THEOREM 11 : *The matching rate at a node i of the discrimination tree is*

$$\overline{\mu_i} \simeq {c_0}^{x_i} \frac{1}{P(1)} \frac{1}{\omega_i + 1} \frac{1}{\lambda_u}$$

and for a join node i

$$\overline{\mu_i} \simeq {c_0}^{x_i} \frac{1}{P(1)^l} \frac{1}{\prod_{j_i=1}^{l} (\omega_{j_i} + 1)} \frac{1}{\lambda_u^l}$$

We can see that the multiset model yields to the *same type of expression for the cost of the algorithm*; we just have to substitute Φ_i for Π_i in Eq. (1).

Let us compare the results of the two models (let us recall that $A(1) = P(1)$). In the multiset model, there appears the ratio of the average size $\frac{1}{\lambda_u}$ to the size of the terms at the j_i. If we denote by $\sigma = \frac{1}{\lambda_u}$ the average size of terms, then

$$\overline{\mu_i} = \overline{\alpha_i} \prod_{j_i=1}^{l} \left(\frac{\sigma}{\omega_{j_i} + 1}\right)$$

and $\omega_{j_i} + 1$ is the size of the terms at the node j_i.

V. Multiple ranges of variation and probability

5.1. Multiple ranges of variation for the constants

Up to now, we have considered that there was only one set of c_0 constants in $FT(F)$. It meant that any constant could instanciate any argument of any function symbol, without any semantic consideration. We shall now distinguish p separate ranges of variation for the constants $D_1, \ldots, D_p$ of cardinality $d_1, \ldots, d_p$. We consider that an equality test can only happen between two variables of the same set. We have to precise the range of variation of each argument of each function symbol. Let i be a node of the discrimination tree. The arity of the function symbol of the i-pattern is ω_i and let us denote by $f_{i,r}$ the number of arguments of f_i that vary in D_r (we have $\sum_{i=1}^{p} f_{i,r} = \omega_i$ and of course the value of some of the $f_{i,r}$ can be zero).

This precision will lead to a slight modification of the previous results; indeed we have

$$P(z) = \sum_{f_i \in F} \left(\prod_{r=1}^{p} {d_r}^{f_{i,r}}\right) z^{\omega_i + 1} = z^{m+1} A(z)$$

Remark : Of course if $p = 1$ *i.e.* there is only one set of c_0 constants, we find again $[z^n]P(z) = c_{n-1}{c_0}^{n-1}$.

We always have $\frac{1}{\lambda_u} = \frac{P'(1)}{P(1)}$ but instead of x_i we have to define a p-tuple $x_{i,1}, \ldots, x_{i,p}$ with $x_{i,r}$ the number of *distinct* arguments of the i-pattern belonging to D_r (and the previous x_i is equal to $\sum_{r=1}^{p} x_{i,r}$). Hence

THEOREM 12 : *The generating function of the terms that match at a node i of the discrimination tree is, for the list model :*

$$B_i(z) = \left(\prod_{r=1}^{p} {d_r}^{x_{i,r}} \right) z^{\omega_i - m}$$

(for the multiset model, one just has to change the power of z).

Example : Let us consider the pattern :

$$(f \;\; a \; b \; a \; X \; Y \; X' \; X \; Z \; X)$$

with $X,\ X' \in D_1$, $Y \in D_2$ and $Z \in D_3$ and a, b some constants. Thus we have $x_{i,1} = 2$, $x_{i,2} = 1$ and $x_{i,3} = 1$ whence $B_i(z) = ({d_1}^2 d_2 d_3) z^{9-m}$.

For a node i of the join network, we denote $x_{i,r} = \sum_{j_i=1}^{l} x_{j_i,r} + y_{i,r}$ with $y_{i,r}$ (≤ 0) the corrective term due to joins in D_r at i. *With those notations we see that nothing is modified in the calculations of the previous Parts and in the final results one just has to substitute* ($\prod_{r=1}^{p} {d_r}^{x_{i,r}}$) *for* ${c_0}^{x_i}$.

This notion of multiple ranges of variation for the constants will enable us to consider a test of inequality : if at a node i, we have to test $X > a$ with a a constant of the set D and X a variable. D is therefore assumed to be ordered and we are then able to determine the set $D_a = \{ Y \in D \mid Y > a \}$ of cardinality d_a. This inequality test will therefore match d_a times as many terms as the equality test $X = b$ with b any constant in D and its cost has to be multiplied by the same constant d_a too.

5.2. Probability

The models studied up to now consider a uniform distribution over all the terms of the same size n in $FT(F)$. In order to get an even more likely modelling, we shall consider a model taking into account the fact that certain symbols are more frequent than others (see [Alb 88], [AF 88] and [FSS]). Thus we present here a *weighted model*. Let w be a *weight function* that assigns to each function symbol $f \in F$ (resp. to each constant c) a non negative real number $w[f]$ (resp. $w[c]$). Then this weight is extended multiplicatively to terms. If t is a term, its weight is defined as

$$w[t] = \prod_{symbol \in t} w[symbol]$$

In order to use a probabilistic weighted model for symbols we choose :

$$\sum_{f \in F} w[f] = |F| \quad and \quad \sum_{c \in D_r} w[c] = d_r \quad \forall\ r$$

(with $|F|$ the cardinality of F). We have chosen to use a weight function rather than a probabilistic model (in which all the above summations would have been equal to 1) to maintain the compatibility with the multiple ranges of variation and to keep similar results. We have indeed :

THEOREM 13 : *The generating function associated to our set of weighted terms is :*

$$W(z) = \sum_{t \in TP(F,w)} w(t) z^{|t|} = \sum_{f_i \in F} w(f_i) \left(\prod_{r=1}^{p} {d_r}^{f_{i,r}} \right) z^{\omega_i + 1}$$

(for the usual size for example).

For a node i of the discrimination tree, we define the weight of the i-pattern W_i as the product of the weight of its already instanciated symbols (variables having by convention a weight equal to 1). For a node i

of the join network, we define the weight of the i-pattern as $W_i = \prod_{j,=1}^{l} W_{j,}$. With these conventions, once more, *all the previous results are only modified by the multiplicative constant W_i, the weight of the i-pattern.*

One can determine *statically* the weights of the symbol by considering the initial Working Memory, or *dynamically* by considering the right handside of the rules and the adding or suppression of certain patterns.

VI. The negation

We have explained at the end of **(2.1)** the functioning of a NOT join node (note that the length of output tuples is the length of the left input tuples). We can easily extend the previous calculations to a model taking into account the *negation*. More precisely, we can also consider non-equality tests between variables and the negation of condition-patterns in the left-hand side of the rules.

In the former case one considers negation inside patterns. Let us consider the following example : $(Head\ X\ \neg X)$. The average number of terms matching with this pattern is equal to the average number of terms matching with $(Head\ X\ Y)$ minus the average number of terms matching with $(Head\ X\ X)$. That can be easily extended to the negation of several variables in a pattern.

Let us consider now the negation of a condition-pattern or of a tuple of condition-patterns.

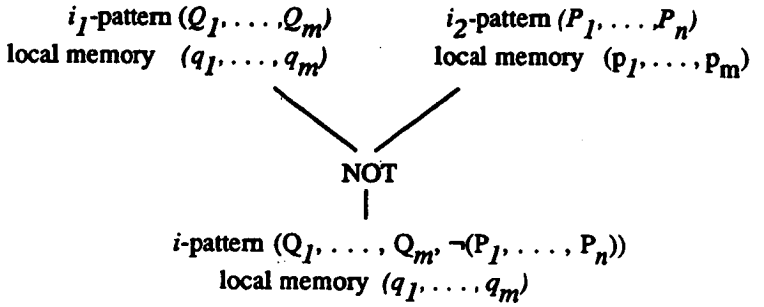


Figure 2 : NOT Join

Let us consider a NOT-join node i (see figure 2). We know the matching-rate of the node i_2 : $\overline{rate_{i_2}}$ and the matching-rate of the node i_1 : $\overline{rate_{i_1}}$. The matching rate of the pattern $\neg(P_1, \ldots, P_n)$ is $1 - \overline{rate_{i_2}}$ and thus the matching rate of the node i corresponding to the i-pattern $(Q_1, \ldots, Q_m, \neg(P_1, \ldots, P_n))$ is $\overline{rate_i} = \overline{rate_{i_1}}(1 - \overline{rate_{i_2}})$ (if the variables of all the P_i are distinct from the variables of all the Q_j, otherwise we modify in this formula $rate_{i_2}$ with the previous reasoning as for node **12** in our example, see appendix). The matching rates of the join nodes that are under this node i (*i.e.* that can be reached from i) are modified in the same way: we determine their usual matching rate forgetting the existence of the tuple of patterns $\neg(P_1, \ldots, P_n)$ and we multiply this result to obtain the real matching rate by the coefficient $(1 - \overline{rate_{i_2}})$ (that implies that the computation of the matching rates is easier top-down in the network).

VII. Composed queries in Relational Databases

In this Part, we shall show that our previous calculations can be easily extended to estimate the average size of a composed query (with possible selections) in a Relational Database.

7.1. Model for the equijoin

From now on, according to Database's vocabulary, we will represent the *relation* $R[X, Y]$ instead of the previous pattern $(R\ X\ Y)$. To determine a *query* in a Relational Database means to find the number of *tuples* (previously terms) that instanciate a relation. *Selection* operations correspond to the tests of the discrimination tree. The *size of a relation* is from now on the average number of tuples that instanciate this relation (we have to forget the previous notion of size of a term).

The type of join we have studied is called *equijoin* in Relational Database; we shall keep on using the word join in what follows. We represent the join between a relation A and a relation B on the attribute (or argument) X by $A[X,Y] \bowtie B[X,Z]$. We are going to determine the average number of tuples (l-tuples) that instanciate a composed query (after possible selections) (see [Alb 89], [Rosen 81], [GP 84] and [GP 88]). For example we want to find the average size of

$$R[a_0, X, Y] \bowtie S[a_1, X, Z] \bowtie T[Y, Z, U, b, Z'] \bowtie \ldots$$

which can be seen as the instanciation of the left-hand side of a rewriting rule.

7.2. Combinatorial study with a Database of k tuples

The tuples are always formed as in $FT(F)$ and we denote their set by $T(F)$. We still denote by $D_1, \ldots, D_p$ the ranges of variation for the constants and by $f_{i,1}, \ldots, f_{i,p}$ the p-tuple that describes the variation of the arguments of a relation symbol f_i. Since we are not interested in the size of a tuple, we can propose for all of them a size 1. Thus the generating function associated with $T(F)$ is :

$$Q(z) = \sum_{f \in F} \left(\prod_{r=1}^{p} d_r^{f_{i,r}} \right) z$$

We have to adapt the modelling for the Relational Database. There are indeed no repetitions in Databases and we shall represent a Relational Database with a *set* of tuples. The generating function that counts the sets of tuples of $T(F)$ is :

$$E(z) = \prod_{t \in T(F)} (1+z) = (1+z)^{\sum_{f \in F} (\prod_{r=1}^{p} d_r^{f_{i,r}})} = (1+z)^{Q(1)}$$

For a relation i, $x_{i,r}$ still denotes the number of distinct variables of i belonging to D_r; we have

THEOREM 14 : *The generating function of the tuples that instanciate relation i is*

$$B_i(z) = \left(\prod_{r=1}^{p} d_r^{x_{i,r}} \right) z$$

We can now prove (see [Alb 89]) :

THEOREM 15 : *Considering any Relational Database with k tuples, the average number of tuples that instanciate a relation i is :*

$$\overline{b_{i,k}} = \left(\prod_{r=1}^{p} d_r^{x_{i,r}} \right) \frac{k}{Q(1)}$$

When we consider a composed join i, we define the same corrective factor $y_{i,r}$ due to the join in D_r. The reasoning of theorem 5 is still valid and we have :

THEOREM 16 : *The average number of l-tuples that instanciate a composed relation i (from l initial relations) is :*

$$\overline{b_{i,n}} = \left(\prod_{r=1}^{p} d_r^{x_{i,r}} \right) \frac{k^l}{Q(1)^l}$$

and the matching rate of this composed join is :

$$\overline{\varrho_i} = \frac{(\prod_{r=1}^{p} d_r^{x_{i,r}})}{Q(1)^l}$$

Let us note that the results we have obtained are exact (and they correspond to the estimation found with the multiset model).

Often we have some more precise information on a Relational Database, and this will enable us to precise the previous study.

7.3. Databases with known sizes of relation

In real applications, one often has an idea of the initial sizes of relation in the initial Relational Database (empirically or with a distribution law). Therefore, we shall not perform average calculations over an arbitrary Database of k tuples any longer but over an arbitrary Database that follows these characteristics. Let us denote k_i the supposed size of relation i; we can propose a more precise generating function for $T(F)$:

$$Q(u_1, \ldots, u_{|F|}) = \sum_{j=1}^{|F|} \left(\prod_{r=1}^{p} {d_r}^{f_{j,r}} \right) u_j$$

with u_j marking the maximal size of relation j. Thus, the generating function of sets formed with tuples of $T(F)$ is :

$$E(u_1, \ldots, u_{|F|}) = \prod_{j=1}^{|F|} (1 + u_j)^{\prod_{r=1}^{p} {d_r}^{f_{j,r}}}$$

$[{u_1}^{q_1} \ldots {u_{|F|}}^{q_{|F|}}] E(u_1, \ldots, u_{|F|})$ is the number of Relational Databases with exactly q_j tuples that instanciate relation j. We obtain :

THEOREM 17 : *The generating function of the tuples that instanciate relation i is :*

$$B_i(u_1, \ldots, u_{|F|}) = \left(\prod_{r=1}^{p} {d_r}^{x_{i,r}} \right) u_i$$

and we find ([alb 89]) :

THEOREM 18 : *The average number of tuples that instanciate a relation i is :*

$$\overline{b_{i,k_1,\ldots,k_{|F|}}} = \left(\prod_{r=1}^{p} {d_r}^{x_{i,r} - f_{i,r}} \right) k_i \qquad (x_{i,r} - f_{i,r} \leq 0)$$

for an arbitrary input Relational Database with q_j tuples that instanciate relation j.

Example : Let us consider the relation $A[a_0, Y]$. If there is k_a tuples that have A as relation symbol in any input Relational Database; if the range of variation for the first variable (resp. the second) of A has d_1 elements (resp. d_2); we have

$$\overline{b_{k_a}} = \frac{k_a}{d_1}$$

since $f_1 = 1$, $f_2 = 1$, $x_1 = 0$ and $x_2 = 1$.

Remark : As we see, the average size of relation i only depends on its initial size. *From now on, we shall not need the initial sizes of other relations.*

The average size for a composed join i, can always be obtained with the reasoning of theorem 5. Thus

THEOREM 19 : *Let us consider any input Relational Database with q_j tuples that instanciate relation j and a multiple equijoin from l initial relations j_i. The average number of l-tuples that instanciate relation i is :*

$$\overline{b_{i,k_1,\ldots,k_{|F|}}} = \left(\prod_{r=1}^{p} {d_r}^{x_{i,r}} \right) \frac{\prod_{j_i=1}^{l} k_{j_i}}{\prod_{j_i=1}^{l} \prod_{r=1}^{p} {d_r}^{f_{j_i,r}}}$$

and the matching rate of i is :

$$\overline{\lambda_i} = \frac{(\prod_{r=1}^{p} {d_r}^{x_{i,r}})}{\prod_{j_i=1}^{l} (\prod_{r=1}^{p} {d_r}^{f_{j_i,r}})}$$

with $x_{i,r}$ the number of distinct variables of relation i in D_r.

Example : Let us consider the average size of the standard join i :

$$R[X,Y] \bowtie S[X,Z]$$

Let us assume that we know the size of the initial relations $R[X,Y]$ and $S[X,Z]$, *i.e.* respectively r and s. Let us denote by d_X, d_Y and d_Z the sizes of the ranges of variation of variables X, Y and Z. We have $x_{i,X} = 1$, $x_{i,Y} = 1$, $x_{i,Z} = 1$, $f_{R,X} = 1$, $f_{R,Y} = 1$, $f_{R,Z} = 0$, $f_{S,X} = 1$, $f_{S,Y} = 0$ and $f_{S,Z} = 1$ whence

$$\overline{b_{i,r,s}} = d_X d_Y d_Z \frac{rs}{d_X d_Y d_X d_Z} = \frac{rs}{d_X}$$

and we find again the well-known result of Rosenthal ([Rosen 81]).

Lastly we shall detail the example of Section **7.1** :

$$R[a_0, X, Y] \bowtie S[a_1, X, Z] \bowtie T[Y, Z, U, b, Z']$$

Let us denote the sizes of the ranges of variation of the variables by d_X, d_Y, d_Z (Z and Z' are two different variables that vary in the same range), d_U, d_a and d_b (the sizes of the range of variation of constants a_0, a_1 and b respectively). We know the number of tuples with function symbol R, S or T, *i.e.* r, s and t respectively. We have $f_{R,X} = 1$, $f_{R,Y} = 1$, $f_{R,Z} = 0$, $f_{R,U} = 0$, $f_{R,a} = 1$, $f_{R,b} = 0$, $f_{S,X} = 1$, $f_{S,Y} = 0$, $f_{S,Z} = 1$, $f_{S,U} = 0$, $f_{S,a} = 1$, $f_{S,b} = 0$, $f_{T,X} = 0$, $f_{T,Y} = 1$, $f_{T,Z} = 2$, $f_{T,U} = 1$, $f_{T,a} = 0$, $f_{T,b} = 1$. and for relation i $x_{i,X} = 1$, $x_{i,Y} = 1$, $x_{i,Z} = 2$, $x_{i,U} = 1$, $x_{i,a} = 0$, $x_{i,b} = 0$. Therefore we have with theorem 19 :

$$\overline{b_{i,r,s,t}} = d_X d_Y d_Z{}^2 d_U \frac{rst}{d_a d_X d_Y d_a d_X d_Z d_Y d_Z{}^2 d_U d_b} = \frac{rst}{d_X d_Y d_Z d_a{}^2 d_b}$$

VIII. Conclusion

We have precised in Section **(2.2)** the notion of average cost for **RETE** algorithm. This average time complexity is given by formula (1) in Section **3.3.** Formula (1) gives the average cost to compute the set of satisfied rules from the set of initial facts and a fixed set of rules with **RETE** algorithm. This result is given in function of the average sizes of local memories in the **RETE** network. These quantities are formulated according to the different models in theorem : 6 for the list model, 10 for the multiset model, at the end of **5.1** when you consider multiple ranges of variation for constants, at the end of **5.2** when you consider different probabilities on symbols. All these expressions can be computed from the parameters of any particular Rule Base and Working Memory.

These theoretical results are experimented on the inference engine Xrete [LCR 88] and we develop for this system an automatic performance analyser from the results of this paper (Clark developed "similar" experimentations on LISP language [Clark 79]). Besides, these theoretical results enable us to propose, as an optimization of the algorithm, a reordoring of the nodes of the network in order to decrease the sizes of local memories.

We saw in Section **5.1** and in Part **6** that we can take into account a test of inequality and negation. As it has been mentioned in the case of the hashing of local memories, the formulae we have obtained can be adapted to many variants of **RETE** algorithm (see [Alb 88] for the **ARBRE D'UNIFICATION**).

We can use the same reasoning in Relational Databases. Indeed, in Part 7 theorem 16, we have presented the average size of a composed equijoin (with possible selections) considering any Relational Database with k tuples. We even have precised these results when the initial sizes of relations to be joined are known (**7.3** theorem 19). These results are also used to improve queries in Relational Databases. (system COSMA [RS 89]).

The work presented in this article, proves once more the power and the easy use of the generating function theory for the precise analysis of algorithms.

IX. Bibliography

[Alb 87] L. Albert, "Présentation et évaluation de la complexité de l'algorithme RETE de multi-pattern matching dans les systèmes de règles de production", rapport de DEA, Université de PARIS XI, ENS, rapport de recherche 87-8 LCR Thomson-CSF, 1987.

[Alb 88] L. Albert, "Présentation et évaluation de la complexité en moyenne d'algorithms de filtrage dans les moteurs d'inférences (Rete et arbre d'unification)", *Revue d' intelligence artificielle*, volume 2, Hermes, 1988, pp. 7-40.

[Alb 89] L. Albert, "Complexité en moyenne de l'algorithme de multi-pattern matching RETE sur des ensembles de patterns et d'objets de profondeur 1", to appear in INRIA Research Report, 1989.

[AF 88] L. Albert, F. Fages, "Average case complexity analysis of the RETE pattern match algorithm", proceedings of the 15^{th} International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, 317, Springer-Verlag, Tampere, Finland, July 1988.

[dB 58] NG de Bruijn, *Asymptotic Methods in Analysis*, Dover 1958.

[CKS 86] C.Choppy, S.Kaplan, M.Soria, "Algorithmic complexity of term rewriting systems", Proceedings of the First Conference on Rewriting Techniques and Applications, Dijon, France, 1986.

[Clark 79] D.W. Clark, "Measurements of Dynamic List Structures Use in Lisp", IEEE Transactions on Software Engineering SE-5(1), pp. 51-59. (Jan. 1979).

[Dieu 68] J. Dieudonné, *Calcul infinitésimal*, Hermann, 1968.

[Dufr 84] P. Dufresne, "Contribution algorithmique à l'inférence par règles de production", Thèse Université Paul Sabatier, Toulouse, 1984.

[DNM 78] J. McDermott, A. Newell, J. Moore , "The efficiency of certain production system implementations", in *Pattern-Directed Inference Systems* (Waterman et Hayes-Roth ed.) Academic Press, New York, 1978, pp. 155-176.

[Fag 86] F. Fages, "On the proceduralization of rules in expert systems", First France-Japan Symposium on Artificial Intelligence, Programming of Future Generation Computers, Addison-Wesley, Eds. M. Nivat and K. Fuchi, Tokyo, Nov., 1986.

[Fag 88] F. Fages, "Rulebased extension of programming langages", Proceedings of Les systèmes experts et leurs applications, Avignon, 1988.

[Fla 85] P. Flajolet, "Mathematical methods in the analysis of algorithms and data structures", INRIA, Research Report 400, 1985. To appear in A Graduate Course in Computer Science,Computer Science Press, 1987.

[Fla 87] P. Flajolet, "The symbolic operator method", *Mathematical methods in the analysis of algorithms and data structure*, L.N.C.S., Springer Verlag, to appear, 1987.

[Forg 79] C. Forgy, "On the efficient implementation of production systems", PhD Thesis, Carnegie Mellon University, 1979.

[Forg 81] C. Forgy, "OPS-V user's manual", Computer Science Department, Carnegie Mellon University, Pittsburgh, MA, 1981.

[Forg 82] C. Forgy, "RETE, a fast algorithm for the many patterns many objects Match problem", *Artificial Intelligence* 19, 1982, pp. 17-37.

[FS 86] P. Flajolet, R. Sedgewick, "Mathematical analysis of algorithms", Computer Science 504, Lecture Notes for Princeton University, 1986.

[FSS] P. Flajolet, P. Sipala et J.M. Steyaert, "The analysis of tree compaction in symbolic manipulations", preprint.

[FV 87] P. Flajolet, J. Vitter,"Average Case Analysis of Algorithms and Data Structures", in *A Handbook of Theoretical Computer Science*, North-Holland, 1987.

[GD 84] M. Ghallab, P. Dufresne, "Moteurs d'inférences pour systèmes de règles de production : techniques de compilation et d'interprétation", Colloque d'Intelligence Artificielle, Marseille, Oct. 1984, pp. 89-103.

[GF 83] A. Gupta et C.L. Forgy, "Measurements on production systems", Carnegie Mellon University, Technical Report CMU-CS-83-167, 1983.

[Gha 80] M. Ghallab, "New optimal decision tree for matching patterns in inference and planning system", 2nd Int. Meeting on Artificial Intelligence, Leningrad, Oct. 1980.

[GP 84] D. Gardy, C.Puech, "On the size of projection: a generating function approach", Information Systems, Vol. 9, No 3/4, pp. 231-235, 1984.

[GP 88] D. Gardy, C.Puech, "On the effect of join operations on relation sizes", to appear in ACM Transactions On Database Systems.

[Gupt 84] A. Gupta, "Parallelism in production Systems : the sources and the expected Speed-up", Carnegie Mellon University Technical Report CMU-CS-84-169, 1984.

[Henr 74] P. Henrici, *Applied and Computational complex Analysis*, Volumes 1-3, Wiley, New-York.

[LCR 88] Laboratoire central de Recherches, *Xrete : manuel de référence*, Thomson-CSF, domaine de Corbeville 91401 Orsay Cedex France, 1988.

[Mir 87] D.P. Miranker, "TREAT: A Better Match Algorithm for AI Production Systems", Proceedings of the 1987 National Conference on Artificial Intelligence, Seattle, Washington, 1987.

[MM 78] A. Meier, J.W. Moon, "On the altitude of nodes in random trees", *Canadian Journal of Math* 30, 1978, pp. 997-1015.

[Rosen 81] A.S. Rosenthal, "Note on the expected size of a join", SIGMOD Record 11(4), pp. 19-25, july 1981.

[RS 89] M. Regnier, E. Simon, "Efficient evaluation of production rules in a DBMS", to appear, 1989.

[SF 83] J.M. Steyaert, P. Flajolet, "Patterns and pattern match in trees : an analysis", *Information and Control* 58, 1983, pp. 19-58.

[SF 88] Schang T. and Fages F. "A Real-Time Expert System for On-Board Radar Identification" 55th Symposium AVP-AGARD on Software Engineering and its Applications to Avionics, 1988.

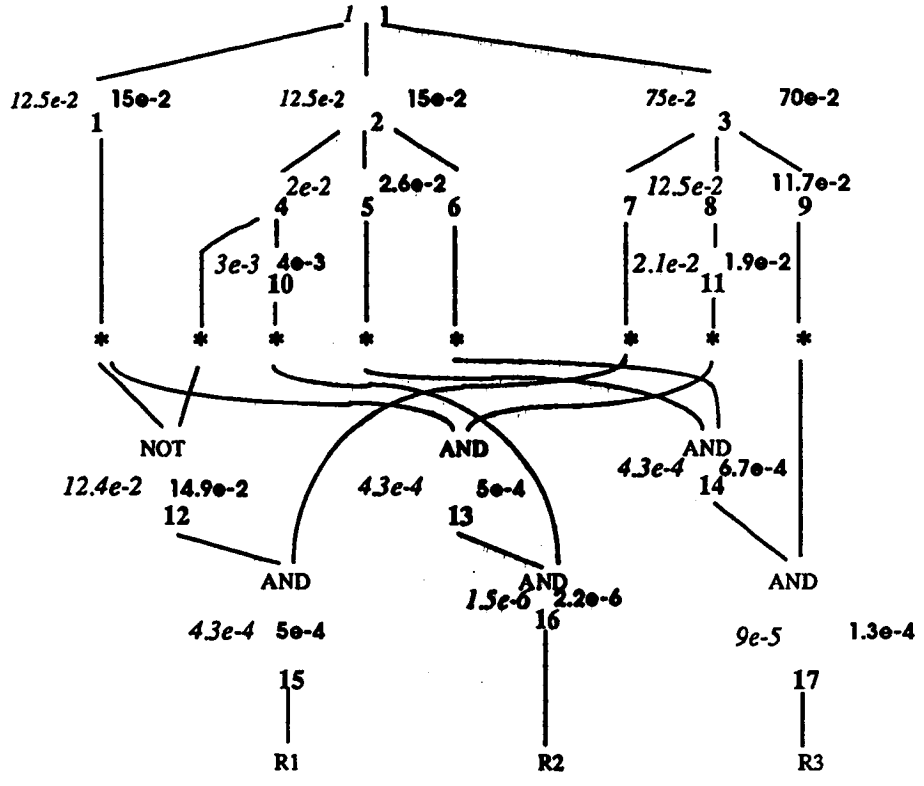
[Stey 84] J.M. Steyaert, "Complexité et structures des algorithmes", Thèse d'Etat, Université de Paris 7, 1984.

[Vien 86] G. Viennot, "La combinatoire bijective par l'exemple", Université de Bordeaux 1, 1986.

[WGF 86] M.L. Wright, M.W. Green, G.Fiegl, P.F. Cross, "An Expert System for Real-Time Control", SRI International, in IEEE Software, March 1986.

Appendix

We develop numerically the example of the three rules of Section 2.1, the RETE network of which is represented in figure 1. In 3.1, we precised the characteristics of this example (generating function, average size, ...). We present on the drawing of the network the matching rate of each node for the list and the multiset models with only one range of variation for constants and a uniform distribution. Matching rates are in *italic* for the list model and **bold-faced** for the multiset model.



(for writing conveniences, we wrote but once the matching rates at nodes 4, 5, 6 and 7, 8, 9. Because we have $\overline{rate_4} = \overline{rate_5} = \overline{rate_6}$ and $\overline{rate_7} = \overline{rate_8} = \overline{rate_9}$ in both models).

We obtained the matching rate at node 12 with :

$$\overline{rate}((g\ \ X\ Y), \neg(f\ \ a0\ X)) = \overline{rate}(g\ \ X\ Y) - \overline{rate}((g\ \ X\ Y), (f\ \ a0\ X))$$

Remark : Note that the average number of terms or of l-tuples of terms that match at a node i is obtained by multiplying the matching rate by k^l with k the number of terms of the input Working Memory and l the length of the output tuples of terms at i.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique