ON LINE PROCESSING OF COMPACTED RELATIONS

F. BANCILHON, P. RICHARD, M. SCHOLL

INRIA B.P. 105
78153 Le Chesnay Cédex - France

ABSTRACT

Most data base machines use some kind of "filter" that performs unary relational operators (selection and projection) on relations [1 to 7]. These filters operate "on the fly" that is, at the speed of the disk, while the relation is being transferred into main memory. Processing time being proportional to relation size, it is therefore important to represent data in the most compacted way. In this paper we address the problem of satisfying the two seemingly contradictory requirements :

i) finding an "optimal" compaction scheme
ii) processing optimally compacted relations on the fly.

INTRODUCTION

Most database machines (DBM) use some kind of filter that performs unary relational operators (selection and projection) on relations (see for example [1 to 7]). These filters operate "on the fly", that is at the speed of the disk, while the relation is being transferred into main memory. Processing time being proportional to relation size, it is important to represent data in the most compacted way. Most DBM just process standard uncompacted data [2,3,4,6,7]. We are currently realizing a machine [1] that uses such a filter to process compacted relations.

In this paper we address the problem of satisfying the two seemingly contradictory requirements :

i) finding an "optimal" compaction scheme
ii) processing optimally compacted relations on the fly.

Section 1 addresses the problem of compacting relations. Compaction formats are defined for files representing a given relation. The notion of maximally compacted file is then introduced. To obtain an "optimally" compacted file, the method suggested in this section is to choose an adequate set of hierarchical dependencies and to compact the file according to that set.

We then turn to the problem of processing such compacted files (section 2).

One reasonable way of filtering compacted files is the Finite State Automaton (FSA) approach. In [1] we concentrated on the problem of realizing such a filtering mechanism and raised the following question : given a selection projection operation and given a file compacted according to some format, can we always find a FSA that performs the operations on the fly on this file. The answer was no and a restrictive class of compacted files was exhibited on which any selection projection operation can be performed on the fly. In this paper we give a complete characterization of operations that can be performed on the fly.

I. COMPACTING RELATIONS

We assume the reader familiar with relational terminology. A relation R is defined over a set of attributes U; with each $A \in U$ is associated a domain D(A), we denote $D = \bigcup_{A \in U} D(A)$.

Relations are represented by sequential files. Attribute values in these files are represented by an attribute tag (that indicates the attribute name) followed by the attribute value and ended by an end tag.

A file over U is a string over D^+. For instance if U = {Course, Student, Grade} then F_0 = Math Jones A Math Susan B Latin Mike D, is a file over U.

Definition I.1. A compaction format over U is defined recursively as follows :

1) A and A^+ are compaction formats over A
2) if $\mathcal{C}$ is a compaction format over X so is $(\mathcal{C})^+$
3) if $\mathcal{C}_1$ and $\mathcal{C}_2$ are compaction formats over X_1 and X_2 and $X_1 \cap X_2 = \emptyset$ then $\mathcal{C}_1.\mathcal{C}_2$ is a compaction format over $X_1 \cup X_2$.

Such a definition in fact yields a special subset of regular expressions. Examples of compaction formats over ABC are $(A(BC^+)^+)^+$, $(ABC)^+$ or $(AB^+C^+)^+$.

The language $\mathcal{L}(\mathcal{C})$ associated with $\mathcal{C}$ is defined by :

1) $\mathcal{L}(A) = D(A) \quad \forall A \in U$
2) $\mathcal{L}(\mathcal{C}_1.\mathcal{C}_2) = \mathcal{L}(\mathcal{C}_1).\mathcal{L}(\mathcal{C}_2)$
3) $\mathcal{L}(\mathcal{C}^+) = (\mathcal{L}(\mathcal{C}))^+$

Of course sentences from these languages are files over U. We shall say that file F satisfies compaction format $\mathcal{C}$ if :

$$F \in \mathcal{L}(\mathcal{C})$$

For instance

Math Jones A Math Susan B Latin Mike D

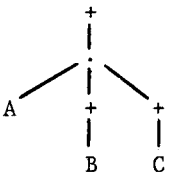
satisfies (Course Student Grade)$^+$ while

Math Jones A Susan B Latin Mike D

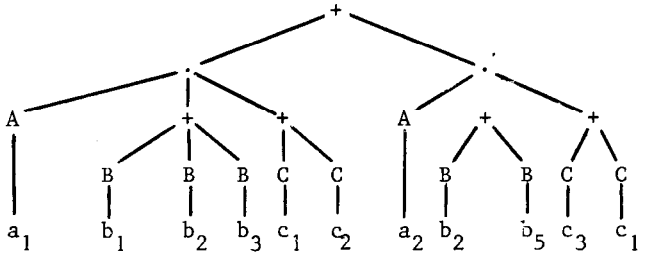
satisfies (Course (Student Grade)$^+$)$^+$.

These examples should give an intuitive feeling of the meaning of compaction formats : the last file consists of courses followed by sequences of student, grade couples.

We shall find it practical to associate with a compaction format its "syntax tree". For instance the syntax tree of $(A\ B^+\ C^+)^+$ is



We shall also find it useful to associate with every file F that satisfies $\mathcal{C}$ its syntax tree. We give an example since the notion is fairly straightforward. Consider the file : $a_1\ b_1\ b_2\ b_3\ c_1\ c_2\ a_2\ b_2\ b_5\ c_3\ c_1$ over attributes (A,B,C). It satisfies the compaction format $(A\ B^+\ C^+)^+$. The syntax tree of the sentence is



One can see that it can easily be obtained from the compaction format syntax tree through an expansion process.

Files are eventually meant to represent relations so we must define the interpretation of a file :

Definition I.2. Let F be a file over U satisfying compaction format $\mathcal{C}$ and let T be its syntax tree. With every subtree T' in T we associate an attribute set atset (T') and an interpretation int (T') which is a relation over atset as follows :

1) $\underline{\text{atset}}\ (a) = \{A\}$ where $a \in D(A)$
   $\underline{\text{int}}\ (a) = \{a\}$

2) $\underline{\text{atset}}\ (T_1.T_2) = \underline{\text{atset}}\ (T_1) \cup \underline{\text{atset}}\ (T_2)$ (note that these are disjoint sets)
   $\underline{\text{int}}\ (T_1.T_2) = \underline{\text{int}}\ (T_1) \times \underline{\text{int}}\ (T_2)$

3) $\underline{\text{atset}}\ (+(T_1\ T_2\ \ldots\ T_n)) = \underline{\text{atset}}\ (T_i)\ \ \forall\ i$ (this is always the same set)
   $\underline{\text{int}}\ (+(T_1\ T_2\ \ldots\ T_n)) = \bigcup_i \underline{\text{int}}\ (T_i)$

For instance the interpretation of the file :

$$a_1\ b_1\ b_2\ b_3\ c_1\ c_2\ a_2\ b_2\ b_5\ c_3\ c_1$$

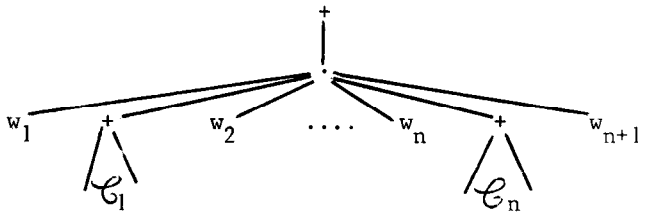
consists of tuples $(a_1\ b_1\ c_1)$, $(a_1\ b_1\ c_2)$, $(a_1\ b_2\ c_1)$ $(a_1\ b_2\ c_2)$, $(a_1\ b_3\ c_1)$, $(a_1\ b_3\ c_2)$, $(a_2\ b_2\ c_3)$, $(a_2\ b_2\ c_1)$ $(a_2\ b_5\ c_3)$ and $(a_2\ b_5\ c_1)$.

When relation R interprets file F we say that F represents R. It is clear that several files can represent the same relation. We are obviously interested into the shortest possible representation of a relation.

A few definitions are first necessary. The general form of a format is :

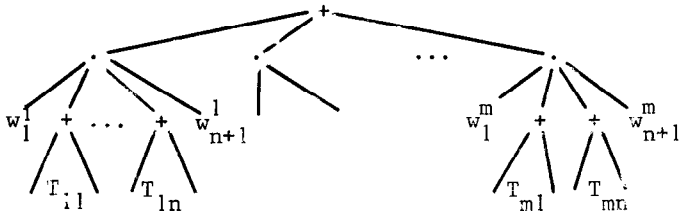
$$\mathcal{C} = (w_1\ \mathcal{C}_1^+\ \ w_2\ \mathcal{C}_2^+\ \ldots\ w_n\ \mathcal{C}_n^+\ \ w_{n+1})^+$$

where the w_i's are sequences of attributes and the $\mathcal{C}_i$'s are CF's. This corresponds to the following tree



We call $w_1\ w_2\ \ldots\ w_{n+1}$ the header of the compaction format $\mathcal{C}$, and $\mathcal{C}_1\ \ldots\ \mathcal{C}_n$ its tails.

Let now F be a file satisfying $\mathcal{C}$ and T its syntax tree, it has the following form :



Trees $T_{11}\ \ldots\ T_{1n}\ \ldots\ T_{m1}\ \ldots\ T_{mn}$ will be called subfiles of F. They satisfy the CF's $\mathcal{C}_1^+\ \ldots\ \mathcal{C}_n^+$.

Definition I.3. File F is said to be maximally compacted with respect to $\mathcal{C}$ iff

1) The projection of F on the header of $\mathcal{C}$ contains no duplicate.
2) each subfile of F is maximally compacted. □

Example

$$a_1\ b_1\ c_1\ a_1\ b_1\ c_2\ a_1\ b_2\ c_1\ c_2\ a_2\ b_1\ b_4\ c_1\ c_2$$

which satisfies $(A\ B^+\ C^+)^+$ is not maximally compacted because its header is A and its projection on A is :

$a_1\ a_1\ a_1\ a_2$

while $a_1\ b_1\ b_2\ c_1\ c_2\ a_2\ b_1\ b_4\ c_1\ c_2$ is maximally compacted. Note that they represent the same relation.

Theorem I.1. Let R be a relation and F a file maximally compacted w.r.t $\mathcal{C}$ that represents R, then :

$\forall$ F' $\in \mathcal{L}(\mathcal{C})$ s.t. F' represents R
length (F') $\geq$ length (F) □

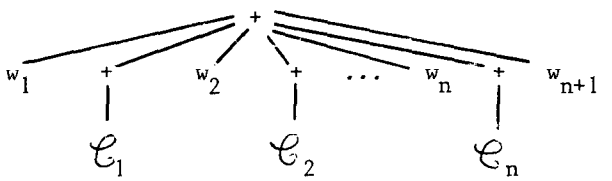
Proof : Let $\mathcal{C} = (w_1\ \mathcal{C}_1^+\ w_2\ \mathcal{C}_2^+ \ldots \mathcal{C}_n^+\ w_{n+1})^+$ let $X = w_1\ w_2 \ldots w_{n+1}$. Any $x \in R(X)$ must appear at least once in the file F'. It appears once and only once in F (by definition of maximal compaction) therefore the header of F' is not shorter than that of F. The same process can then be repeated for all subfiles of F'. Q.E.D.

The next question is : Given a CF $\mathcal{C}$ on a set of attributes U, which are the relations R(U) that have a maximally compacted representation according to $\mathcal{C}$.

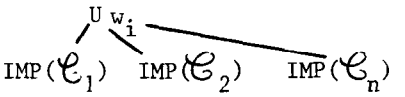
In order to determine that set of relations we introduce the set of integrity constraints associated with a compaction format. These integrity constraints happen to be a special set of multivalued dependencies namely hierarchical dependencies as defined by Delobel [8].

We characterize the set of dependencies implied by a compaction format (denoted IMP($\mathcal{C}$)) in terms of Delobel's generalized hierarchical decomposition (GHD). We need to consider the syntax tree associated with the compaction format.

The general form is :



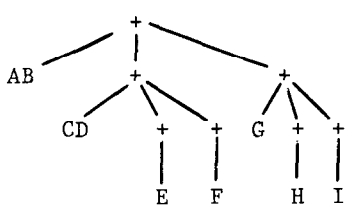
We recursively define IMP($\mathcal{C}$) (which is a tree) as follows :

IMP($\mathcal{C}$) = 

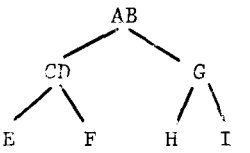
Let us consider for example the following CF :

$\mathcal{C} = (AB(CD\ E^+\ F^+)^+\ (G\ H^+\ I^+)^+)^+$

Its syntax tree is :



IMP($\mathcal{C}$) is :



Thus IMP(CF) is obtained from the syntax tree of the compaction format only by removing the '+' signs. We recall [8] that a GHD is a particular set of hierarchical dependencies.

In the case of the above example, we have :

AB $\twoheadrightarrow$ CDEF | GHI
ABCD $\twoheadrightarrow$ E | F
ABG $\twoheadrightarrow$ H | I

We can now state our main theorem.

Theorem II.1. Let $\mathcal{C}$ be a CF over U
Let IMP($\mathcal{C}$) be its implied GHD.
R(U) has a maximally compacted representation according to $\mathcal{C}$ iff R(U) satisfies IMP($\mathcal{C}$). □

Proof :

1) Only if part :

IMP($\mathcal{C}$) is a set of hierarchical dependencies.
Let $X \twoheadrightarrow X_1 | \ldots | X_n$ be a dependency in IMP($\mathcal{C}$)
Let F be a maximally compacted file according to $\mathcal{C}$.
Let F_x be the restriction of F on value x of X
(F_x is the subfile of F such that its value on X is x and that contains every tails associated to X-value x)
Let $F_x|X_i$ be the values of INT(F_x)[X_i] (i.e. a set of tuples on X_i). We know that x appairs once and only once in F (since F is maximally compacted). Thus the only tuples in INT(F)[X $X_1 \ldots X_n$] that have X-value x are in the following set :

$\{x\} \times F_x|X_i \times \ldots \times F_x|X_n$

By iterating on every X-value in F we have :

INT(F)[X $X_1 \ldots X_n$] = INT(F)[X X_1] *...* INT(F)[X X_n]

Thus INT(F) obeys $X \twoheadrightarrow X_1 | \ldots | X_n$. By iterating we show that it obeys the GHD implied by $\mathcal{C}$.

2) if part :

Let R(U) be a relation that satisfies a gives GHD.
Let $\mathcal{C} = (w_1\ \mathcal{C}_1^+\ w_2\ \mathcal{C}_2^+ \ldots w_p\ \mathcal{C}_p^+\ w_{p+1})^+$ be a CF such that IMP($\mathcal{C}$) = G. We construct a maximally compacted file according to $\mathcal{C}$ that represents R.
Let $X = U\ w_i$, and Y_i = atset($\mathcal{C}_i$) then, following the definition of the GHD, the following dependency holds in R(U).

$X \twoheadrightarrow Y_1 | \ldots | Y_p$

We use a notation defined by Delobel [8] :

$$\forall x \in R(X) : R(x,Y) = \{y \in R(Y)/xy \in R(XY)\}$$

We recursively define the compaction of R(U) according to $\mathcal{C}$ as the following file named F :

$$F = COMP[R(U),\mathcal{C}] =$$

$$x_1[w_1]COMP(R(x_1,Y_1),\mathcal{C}_1)\ldots x_1[w_p]COMP(R(x_1,Y_p),\mathcal{C}_p)$$
$$x_1[w_{p+1}]x_2[w_1]COMP(R(x_2,Y_1),\mathcal{C}_1)\ldots x_2[w_{p+1}] \ldots$$
$$x_n[w_1]COMP(R(x_n,Y_1),\mathcal{C}_1)\ldots x_n[w_{p+1}].$$

Where : $R(X) = \{x_1,\ldots,x_n\}$ and $x_i[w_j]$ is the projection of x_i on w_j.

The definition is sound since every $IMP(\mathcal{C}_i)$ is a GHD satisfied by $R(x,Y_i)$ for all x in R(X), and F is maximally compacted since every x_i in R(X) appears once in F_X and since the definition is recursive.

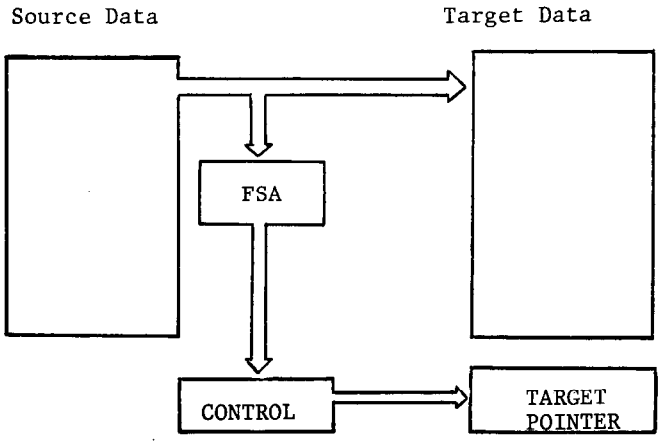
To end the proof the reader only has to notice that by construction INT(F) = R(U). Q.E.D.

This theorem gives us a tool to choose CF's over sets of attributes in order to get the maximal compaction for the physical structure of relations. We have to check if there exists a GHD over the relations we want to store, and to create a compaction format such that the GHD is the set $IMP(\mathcal{C})$. Then we can compact our relations, the compaction process being information lossless. If there is no such dependency, we can choose any simple right compaction format and compact the relations according to this format in order to get maximally compacted files, obviously, this process is generally less efficient in terms of space than the compaction according to a general compaction format.

For example, compacting R(Course,Hour,Room, Student,Grade) according to $(C(HR)^+(SG)^+)^+$ gives shorter files than compacting according to $(C(HR(SG)^+)^+)^+$.

II. PROCESSING COMPACTED RELATIONS

To filter compacted relations, the FSA approach was suggested in [1]. Such a filtering mechanism is shown below.



Data is read one byte at a time from a source into the target buffer at the address indicated by the target pointer. Data is also sent to a FSA that controls the target pointer : it can increment that pointer, save it into a stack, push or pop this stack.

To each selection/projection operation corresponds a FSA, i.e. a specific set of transition and output functions. This set constitutes a program stored in the filter memory.

The transition from a state to another will depend upon the characters read from the source.

In the sequel we assume the automaton is dealing with unit length words whose type it recognizes, i.e., we ignore for clarity the lexical analysis : an attribute value is represented by one word.

To each state correspond one or several among the following output functions (commands to the target pointer)

(1)		TP ← TP + 1	The next word in sequence is read from the source and written in sequence into the target buffer.
(2)	DROP	PUSH TP on S	The target Pointer's content is pushed onto Stack S.
(3)	RESET	TP ← POP(S)	S is popped into the target pointer, i.e. the next word will be written at the address memorized in S. Clearly, this means erasing all words written into the buffer since the last memorization.
(4)	ERASE	POP S	S is popped. This implies keeping all words written in the target buffer since the last memorization.

II.1. Right simple compaction formats

Let $(R|C)[X]$ denote the projection on $X \subseteq U$ of the restriction by boolean condition C of relation R. The above filtering mechanism was designed to perform restriction/projection operations on any compacted relation.

Consider file F : $(Class,Prof,(Student,Grade)^+,(Hour,Room)^+)^+$ and the query q_1 : $(R|Grade=B)[Class,Student]$. It is possible to generate a FSA that performs this query "on the fly" on file F (for details, see [1]).

However it is easy to find counterexamples where a query cannot be performed on the fly. In particular, on the above example of file F, with the following query we cannot generate a FSA to filter F on the fly. q_2 : $(R|Grade=B \vee Room=1108)[Class, Hour,Student]$.

Indeed, after having scanned a set of students, either we have eliminated all those whose grade was not B in which case we may discover later that the room where the class meets is "1108", or we have kept them and discover that Room ≠ 1108, in which case we have to backup and erase them.

The question is then : given a file, compacted according to some CF, and given a query on this file, can we find a FSA that performs the query on the fly on this file ?

Theorem II.1. Given any CF and given a file F satisfying this CF, any projection/selection operation whose boolean condition does not contain any or can be performed on the fly. The formal proof was given in [1].

Theorem II.2. Let F be a file satisfying a Right Simple CF. Then any restriction/projection operation can be performed on the fly on F.

Recall a CF is Right Simple (RS) if in its syntax tree all + signs are on the right and on the same branch.

Proof : see [1].

The latter condition on the set of CF's is rather restrictive.

In order to give a complete characterization of operations that can be performed on the fly we give below an algorithm that, given any CF and an operation (containing 'or' s) decides whether the operation can be performed on the fly on a file compacted according to this CF or not. In the latter case the operation must be decomposed in several operations each of them being performed on the fly.

II.2. A necessary and sufficient condition for a query to be performed on the fly on a non RS file

We call rank of Attribute Name 1, denoted by rank(A), the number of attribute names counted before A when scanning the CF from left to right.

The boolean condition $\mathcal{C}$ of the query is of the form :

$$\mathcal{C} = C_1 \vee C_2 \vee \ldots \vee C_i \vee \ldots \vee C_n$$

where C_i is a conjunction of terms of the form < Attribute name > < comparator > < value >.

We denote by $A_M(i)$ the attribute name of condition C_i whose rank is the largest :

for all $A \in U \cap C_i$ we have $\text{rank}(A) \leq \text{rank}(A_M(i))$

We assume we have :

$$\text{rank}(A_M(1)) \leq \ldots \leq \text{rank}(A_M(i)) \leq \ldots \leq \text{rank}(A_M(n))$$

We denote by Tree(k) the subtree whose root is internal node k in the syntax tree associated with the CF.

Theorem II.3. The following property $\mathbb{P}$ is necessary and sufficient for the restriction/projection $(R|C_1 \vee \ldots \vee C_i \vee \ldots \vee C_n)[X]$ to be performed on the fly on a file compacted according to CF C_F :

$\mathbb{P}$ { For all k internal nodes of the syntax tree associated with C_F, such that $X \cap \text{Tree}(k) \neq \emptyset$. If $\exists$ C_i such that $A_M(i) \in \text{Tree}(k)$ then $A_M(j) \in \text{Tree}(k)$ for all $i < j \leq n$

As an example, consider the file

$$F : (\text{Class}, (\text{Student}, \text{Grade})^+, (\text{Hour}, \text{Room})^+)^+$$

Query q_1 : $(R|\text{Grade} = B \vee \text{Room} = 1108)[\text{Class}, \text{Hour}, \text{Student}]$ cannot be performed on the fly, since in the subtree {Student,Grade}

1) there are both an attribute to be projected (Student) and an attribute of condition C_1 (Grade = B),

2) 'Room' which belongs to condition C_2 (Room = 1108)

   - appears on the right of 'Grade' in the CF
   - and does not belong to the subtree.

On the contrary, on the same file F, query q_2 :

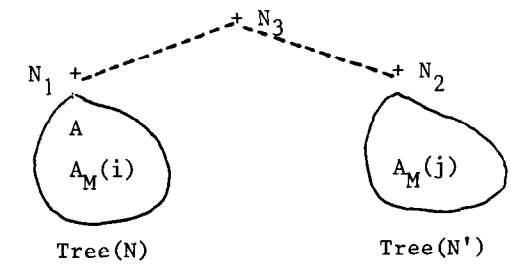
$(R|\text{Grade} = B \vee \text{Room} = 1108)[\text{Class}, \text{Hour}]$

can be performed on the fly.

Proof :

1) Necessary condition : we show that if property $\mathbb{P}$ is not satisfied, the query cannot be performed on the fly :

Assume there exists $\text{Tree}(N_1)$ including Attribute A to be projected, attribute $A_M(i)$ of largest rank in condition C_i and that attribute $A_M(j)$ of largest rank in condition C_j $(j > i)$, does not belong to $\text{Tree}(N_1)$, but to $\text{Tree}(N_2)$.



There exists $\text{Tree}(N_3)$ including both $\text{Tree}(N_1)$ and $\text{Tree}(N_2)$. Between two successive visits to node N_3, before N_2 is reached, $\text{Tree}(N_1)$ has been scanned one or several times. Upon each visit to $A_M(i)$ we can conclude whether C_i is satisfied or not and therefore decide whether to keep (C_i satisfied) or erase (C_i not satisfied) one or more values of A projected between two successive visits to $A_M(i)$.

Eventually when we finally leave Tree(N_1) if we have erased one or more values of A (because C_i was not satisfied), then we may get into trouble later on when scanning attribute $A_M(j)$: if condition C_j is satisfied, we should have kept all values of A, not only those for which C_i is satisfied.

On the contrary, if we had kept all values of A (even those for which C_i was not satisfied) then we encounter the risk that C_j is never satisfied before coming back to node N_3.

2) sufficient condition : clearly, when scanning the file, upon leaving any subtree,

a) either there was no attribute to be projected in this subtree, then there is no decision to be made,

b) or at least one attribute is to be projected. Then, if we had to make a decision at this level (keep or erase some projected values), it is because we were able to conclude whether a condition is satisfied or not (in this subtree we encountered the attribute of largest rank for this condition).

Property $\mathbb{P}$ implies the decision is safe, i.e. within each subtree, if we can make a decision on one condition, we can make a decision on all remaining conditions.

We choose a constructive proof to show that if $\mathbb{P}$ is satisfied, the query can be performed on the fly.

We restrict ourselves to the case where :

1) $\mathcal{C} = C_\alpha \vee C_\beta$

2) the CF is : $(A(BC^+)^+D^+)^+$, i.e., the depth of the tree is 3, there are at most three branches and there is only one attribute at each level.

3) All attributes are to be projected.

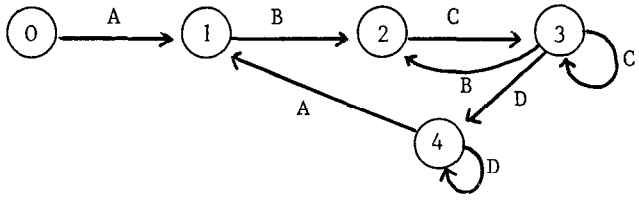
The proof can then be generalized but with a great loss of readibility.

We exhibit now a FSA that can perform the requested restriction.

Upon scanning an attribute value, the automaton can decide whether this value satisfies conditions C_α and C_β or not.

$C_\alpha(A)$ (resp. $C_\beta(A)$) is true if attribute A does not appear in condition C_α(resp. C_β) or if it actually belongs to C_α (resp. C_β) and it is tested true.

The state diagram is given below. The states (0,1,2,3,4) memorize at which level of the tree we are.



We need the following variables $E_{1\alpha}$, $E_{1\beta}$, $E_{2\alpha}$, $E_{2\beta}$, $E_{3\alpha}$, $E_{3\beta}$, $E_{4\alpha}$, $E_{4\beta}$ for memorizing the evaluation of C_α and C_β in a given state : E_{ix} is true if condition C_x has been satisfied at level i.

Finally we need six more variables, $H_{1\alpha}$, $H_{1\beta}$, $H_{2\alpha}$, $H_{2\beta}$, $H_{3\alpha}$ and $H_{3\beta}$ to remember whether the prefix of the current tuple has already been validated or not :

H_{ix} is true if until level i we have had a hit with condition C_x.

The following table gives for each state and current attribute, the transition and output functions.

State	Attribute	Transition	Output
0	A	$H_{1\alpha}=H_{2\alpha}=H_{3\alpha}=0$ $H_{1\beta}=H_{2\beta}=H_{3\beta}=0$ $E_{1\alpha}=C_\alpha(A)$ $E_{1\beta}=C_\beta(A)$	DROP
1	B	$E_{2\alpha}=E_{1\alpha} \wedge C_\beta(B)$ $E_{2\beta}=E_{1\beta} \wedge C_\beta(B)$	DROP
2or3	C	$E_{3\alpha}=E_{2\alpha} \wedge C_\alpha(C)$ $E_{3\beta}=E_{2\beta} \wedge C_\beta(C)$ $H_{2\alpha}=H_{2\alpha} \vee E_{3\alpha}$ $H_{2\beta}=H_{2\beta} \vee E_{3\beta}$ $H_{1\alpha}=H_{2\alpha} \vee E_{3\alpha}$ $H_{1\beta}=H_{1\beta} \vee H_{2\beta}$	DROP IF $E_{3\alpha} \vee E_{3\beta}=1$ then ERASE ELSE RESET
3	B	 $H_{2\alpha}=H_{2\beta}=0$ $E_{2\alpha}=E_{1\alpha} \wedge C_\alpha(B)$ $E_{2\beta}=E_{1\beta} \wedge C_\beta(B)$	IF $H_{2\alpha} \vee H_{2\beta}=1$ then ERASE ELSE RESET DROP
3or4	D	$E_{4\alpha}=H_{1\alpha} \wedge C_\alpha(D)$ $E_{4\beta}=H_{1\beta} \wedge C_\beta(D)$ $H_{3\alpha}=H_{3\alpha} \vee E_{4\alpha}$ $H_{3\beta}=H_{3\beta} \vee E_{3\beta}$	DROP IF $E_{4\alpha} \vee E_{4\beta}=1$ ERASE ELSE RESET
4	A	 $H_{1\alpha}=H_{2\alpha}=H_{3\alpha}=0$ $H_{1\beta}=H_{2\beta}=H_{3\beta}=0$ $E_{1\alpha}=C_\alpha(A)$ $E_{1\beta}=C_\beta(B)$	IF $H_{3\alpha} \vee H_{3\beta}=1$ ERASE ELSE RESET DROP

CONCLUSION

We have studied a representation of relations as compacted sequential files and the problem of processing unary relational operations on this representation.

We have given a simple criterion for obtaining a minimal representation of relation. That criterion is based on hierarchical dependencies. Then we have described a filter that can process these representations and fully characterized the set of operations it can perform.

One of the interesting properties of this approach is that is allows us to process unnormalized relations (relations satisfying MVD's are normally decomposed into smaller relations of avoid duplication thus generating frequent joins).

REFERENCES

[1] F. Bancilhon, M. Scholl, "Design of a Back End Processor for a Data Base Machine", Proceedings SIGMOD, Los Angeles, 1980, pp. 93.93g.

[2] G.P. Copeland, G.J. Lipovski, S.Y.W. Su, "The Architecture of CASSM : A Cellular System for Non-Numeric Processing", Proc. 1st Annual Symposium on Computer Architecture, Dec. 1973, pp. 121-128.

[3] C.A. Ozkarahan, S.A. Schuster, K.C. Smith, "RAP-An Associative Processor for Data Base Management", Proc. 1975 NCC, Vol. 45, AFIPS Press, Montvale, N.J., pp. 379-387.

[4] J. Banerjee, R. Baum, D.K. Hsiao, "Concepts and Capabilities of a Database Computer", ACM Trans. Database Systems, Vol. 3, N° 4, Dec. 1978, pp. 347-384.

[5] H. Auer et al., "RDBM - A Relational Database Machine", Information Systems, Vol. 6, n° 2, 1981, pp. 91-100.

[6] E. Babb., "Implementing a Relational Database by Means of Specialized Hardware", ACM Trans. Database Systems, Vol. 4, N° 1, March 1979, pp. 1-29.

[7] R. Epstein, P. Hawthorn, "Design Decisions for the Intelligent Database Machine", Proc. 1980 NCC, AFIPS press, Montvale, N.J., pp. 237-241.

[8] C. Delobel, "Normalization and Hierarchical Dependencies in the Relational Data Model", ACM Trans. Database Systems, Vol. 3, N° 3, Sept. 1978, pp. 201-202.