**Abstract**
In this paper we introduce bag relational algebra with aggregation over a particular representation of incomplete information called c-tables ([Grah84]). The *c-table* representation is an extension of the Codd relational model ([Codd75]) with richer semantics for null values. The reason c-tables were chosen for our exploration is that they are the least expressive relational representation of incomplete information over which relational algebra is closed and can be "well defined" (see [ImLi84]). We justify the need for duplicate preserving relational algebra over this representation of incomplete information by introducing aggregation over c-tables with duplicates.

## 1. Introduction

Many times when information is entered into databases, the values of some of the fields are left empty for various reasons. In some cases partial information about the blank fields is available, but existing relational databases don't allow such information to be processed. Imielinski and Lipski in [ImLi84] were one of the first to propose richer semantics of null values which allows information about null values to be entered into a relational database. However their model didn't support duplicate semantics. Later on some research was done in the area (see [LiWo94]), but no justification for exploring duplicate semantics for incomplete information was given.

It is our believe that incomplete databases have an important role in the area of information systems. However, in order to become popular, tools for efficient storage and retrieval of such information need to be developed. Although the problem of efficient storage is somewhat solved, it turns out that the problem of querying incomplete information is intrinsically computationally expensive. This is probably the reason little work was done in the area of querying incomplete information (see [Grah91], [Reit86], [YuCh88] and [Lips95]) and no work was done in the area of aggregating over incomplete information.

In this paper we explore the problem of querying and aggregating incomplete information. Even though, the time and space complexity of most of the presented methods is higher then polynomial, we believe the proposed algorithms are feasible in practice. The reason for our optimism is that in most real world problems the size of the uncertain information is a small percentage of the total size of the database. Since the major complexity of the proposed algorithms comes from the size of the uncertain information, if this information is small, there is hope that the proposed algorithms can be applied in practice and will run in reasonable space and time.

The major contributions of this paper include:

- extending c-tables to have linear conditions
- defining duplicate semantics for c-tables
- proposing an algorithm for normalizing c-tables
- proposing algorithms for calculating bag relational algebra operations over c-tables
- proposing algorithms for performing aggregation over c-tables

In what follows in section 2 we define a representation of incomplete information as c-tables with linear conditions, explore the fundamental properties of linear conditions and of c-tables over them and present algorithms for their manipulation. In particular, in section 2 an original algorithm for normalizing c-tables is proposed. In section 3 we define bag relational algebra operators over c-tables and give algorithms for their implementation. In section 4 the problem of aggregating over c-tables is explored and finally in section 5 a summary or the presented material and areas for future research are presented.

## 2. C-tables with linear conditions

The problem of representing incomplete information in the relational model is almost as old as the relational model itself (see [Bisk81],[Codd75],[Codd79],[Gran77] and [ImLi81]). For example null values appear in relational tables and their meaning can be no information available, only partial information available, value not applicable and s.o. Most work on null values has concentrated on the first two meanings of a null value. Known representation of relational tables adapting those meanings of nulls include:

- Codd tables
- Naive tables
- Horn tables
- C-tables

Codd tables are relational tables, where the value of some the fields can be null. Naive tables are an extension of Codd tables, where nulls are marked, i.e. each null is given a label and nulls having the same label are mapped to the same constant. C-tables are naive tables with local conditions associated which each tuple and a single global condition associated with each table (a more detailed overview follows in *section 2.1*). Horn tables are a special kind of c-tables, where the conditions that can appear are restricted to horn clauses.

Imielinski and Lipski ([ImLi84]) have proven that from the above four representations, relational algebra can be "well defined" only over c-tables. The c-tables defined in [ImLi84] have local conditions over ($\mathcal{R}$,>,=) (i.e. Boolean expressions with variables and constants defined over the ordered set $\mathcal{R}$ extended with **>** and **=**) and no global conditions. However, to achieve closure over aggregation we will extend the local conditions to be over ($\mathbb{R}$,**>** ,=,+ )∪(**S**,=,≠), where $\mathbb{R}$ is the set of real numbers and **S** is the set of strings over some finite alphabet. As well, we extend the [ImLi84] definition of a c-table by adding global conditions (first introduced in [Grah84]). Note that, we don't explore ($\mathbb{Z}$,**>**,=,+), where $\mathbb{Z}$ is the set of integers because the general problem of deciding whether a condition over this system is satisfiable is not decidable (see [Mati70]). Also note that, extending the definition of a c-table with global conditions doesn't add any expressive power, but allows for a more compact representation.

A good survey of the existing approaches to representing and querying incomplete information can be found in [Grah91].

## 2.1. Definitions

Formally, we introduce a c-table $T_C$ as a finite, unordered set of *c-tuples* plus a global condition $T_G$. A *c-tuple* in an extension of a relational tuple with a local condition. More precisely, if $T$ is a Codd table with attributes $A_1, A_2, ..., A_n$, a relational tuple is a sequence of $n$ mapping from $A_i$ to $\boldsymbol{D}(A_i)$, for $i$=1..$n$, where $\boldsymbol{D}$ is used to represent the domain of an attribute. On the other hand a c-tuple is a sequence of mappings from $A_i$ to $\boldsymbol{D}(A_i) \cup V_i$ plus a local condition over $L_C$, where $V_i$ is used to represent a possibly infinite, countable set of variables over $\boldsymbol{D}(A_i)$. As mentioned earlier, local and global conditions can range over $(\mathbb{R},>,=,+) \cup (\mathbf{S},=,\neq)$.

Before formally introducing the semantics of a c-table, let's look at the example in table 2.1.

| **name** | **school** | **condition** |
|---|---|---|
| John | $y$ | $x=1$ |
| Mark | $y$ | $x\neq1$ |
| q | z | TRUE |

g.c. $(q\neq$"Mark"$)\wedge(q\neq$John$)\wedge(z\neq y) \wedge ((y=$"Western"$)$ or $(y=$"Eastern"$))$
***Table 2.1.*** *An example c-table*

The different parts of a c-table will be referred as the *main* part, where the actual data is shown, the *local condition* part and the *global condition* part. In the above example *x,y,z* and *q* are used to represent variables ranging over the corresponding domains. Since our model is limited only to the domains of real numbers and strings, the domain type of a variable that doesn't appear in the main part of a c-table can be inferred by the context in which it appears. In the above example, x ranges over the set of real numbers, the reason being the existence of the local condition $x$=1.

The example c-tables expresses that either there are no students or that there is a student, whose name is not John or Make. As well, either there are no other students or exactly one of John or Mark are students. Moreover the one that is in school, studies in either "Western" or "Eastern", but not in the same school as the first student. Note that in this example (and trough out the paper) we are using the *Closed World Assumption* ([Reit78]). This assumption states that a database representation contains only the things that are known to be true, i.e. it doesn't contain things for which we don't have enough information to determine whether they are true or not.

To formally define the semantics of a c-table, Imielinski and Lipski in [ImLi84] introduce a function *Rep*, which maps a c-table $T_C$ to a possibly infinite set of Codd tables. In [ImLi84] this function is defined relative to the open world assumption. We define it relative to the close world assumption as:

$$Rep(T_c)=\{T \mid \exists\, v, \text{ s.t. } v(T_c)=T\}$$

In the definition $v$ is a mapping that maps the variables in $T_c$ to constants in the corresponding domains, it is generalized to a c-tuples $t_C$ of $T_C$ as

$$v(t_C) = \begin{cases} v(main(t_C)) \text{ if } v(lc(t_C)) = TRUE \wedge v(gc(T_C)) = TRUE \\ \varepsilon \text{ otherwize} \end{cases}$$

In the formula the functions *main* and *lc* are used to denote the main part and the local condition part of a c-tuple and *gc* - the global condition of a c-table. The symbol $\varepsilon$ is used to represent the empty set. The value of the tuple $v(main(t_C))$ is calculated by substituting the variables in $t_C$ with the corresponding constants to which $v$ maps them. The mapping $v$ is further extended to a c-table $T_C$ as the bag of tuples $\{|v(t_C)|\}$, where $t_C$ are the c-tuples of $T_C$. The presented definition of $v$ applied to a c-table is unique and differs from the definitions presented in [ImLi84] and [Grah91]. The reason being, that unlike related research, we define semantics for c-tables with **duplicates**.

Intuitively, the meaning of the *Rep* function is that, given a c-table $T_C$ the function returns all possible representations of $T_C$, i.e. all Codd tables that $T_C$ could represent under different valuations[1] $v$. We define the semantics of a c-tables $T_C$ as the set of Codd tables it represents - i.e. as $Rep(T_C)$. Possible representations of the c-table defined in table 2.1 are:

| **name** | **school** |
|---|---|
| John | Western |
| Paul | Eastern |

| **name** | **school** |
|---|---|
| Mark | Eastern |
| Mike | Northern |

*Table 2.2. Possible representations of the c-table defined in table 2.1.*

Note that, these are only two of the possible representations and in particular, there may be infinitely many representations of a c-table. As well, the order of the tuples in the presented example is preserved. This is done to show that order preservation is possible, however, as stated earlier, throughout this paper we will analyze c-tables as unordered bag of c-tuples in order to simplify the reasoning.

## 2.2. Satisfaction decidability of linear conditions trough variable elimination

Before proceeding further with analyzing c-tables, a procedure for satisfaction checking of linear conditions is presented. This procedure will be helpful later when an algorithm for normalizing a c-table is presented.

The first to propose such an algorithm for linear conditions was Tarski in [Tars51]. He proposed a satisfaction decidability algorithm for ($\mathcal{R}$, +,*,>,=), for $\mathcal{R}$ the set of real numbers. A more efficient algorithm for the less expressive system ($\mathcal{R}$,+,>,=) we are interested in is developed by Ferrante and Rackoff in [FeRa75], where $\mathcal{R}$ is an ordered group (under addition) that is divisible and torsion-free (the set of real numbers

[1] From now on, we will use the term valuation to denote the mapping of the set of variables in a c-table to constants in the corresponding domains

and the set of rational numbers are such groups). In this section we will present Ferrante and Rackoff's algorithm.

**Theorem 2.2.1** (From [FeRa75]) Let $\varphi(x_1,x_2,..,x_n)$ be a quantifier free Boolean expression over ($\mathcal{R}$, +,>,=), where $\mathcal{R}$ is an ordered, devisable and torsion-free group and $x_1,.., x_n$ are all the variables in $\varphi$. Then there exist a quantifier free Boolean expression $\varphi'(x_2,x_3,..,x_n)$, such that the expressions $(\exists x_1)\varphi(x_1,x_2,..,x_n)$ and $\varphi'(x_2,x_3,..,x_n)$ are equivalent.

*Proof* (following the proof from [FeRa75]):

First, we will rewrite $\varphi$ as a conjunction of disjunction of atomic formulas of the form $x_1<t_i$, $t_i<x_1$ or $x_1=t_i$, where $i$=1 to $n$ and $n$ is the number of atomic formulas. This can be done by converting $\varphi$ in conjunctive normal form and then solving for $x_1$ in each atomic formula. We will denote by $U$ the set of all $t_i$ (note that $t_i$ doesn't contain $x_1$).

Next, we will present a constructive proof, i.e. we will construct $\varphi'(x_2,x_3,..,x_n)$ as $\varphi_{-\infty} \vee \varphi_{\infty} \vee \bigvee_{t,v\in U} \varphi(\frac{(t+v)}{2},x_2,...,x_n)$, where $\varphi_{-\infty}$ is defined as the value of $\varphi$ when $x_1$ is close to $-\infty$ and $\varphi_{\infty}$ is defined as the value of $\varphi$ when $x_1$ is closed to $\infty$. More precisely $\varphi_{-\infty}$ is calculated by substituting the atomic formulas in $\varphi$ of the form $x_1<t_i$ with *TRUE* and of the form $x_1=t_i$ and $t_i<x_1$ with *FALSE*. Similarly $\varphi_{\infty}$ is calculated by substituting the atomic formulas in $\varphi$ of the form $x_1<t_i$ and $x_1=t_i$ with *FALSE* and of the form $t_i<x_1$ with *TRUE.*

**(=>)** Let for some numbers $r_2,..,r_n$ , $(\exists x_1)\varphi(x_1,r_2,..,r_n)$ be true. Then we have that either:

**(i)** $x_1<\min_i(t_i(r_2,..,r_n))$

**(ii)** $x_1=t_i(r_2,..,r_n)$ for some $i\in[1..n]$

**(iii)** $t_i(r_2,..,r_n)<x_1<t_j(r_2,..,r_n)$, where $\neg\exists k\in[1..n]$, s.t. $t_i(r_2,..,r_n)< t_k(r_2,..,r_n)<t_j(r_2,..,r_n)$

**(iv)** $x_1>\max_i(t_i(r_2,..,r_n))$

If **(i)** is the case then $\varphi_{-\infty}$ is true, if **(ii)** is the case then i.e. $\varphi(\frac{(t_i+t_i)}{2},r_2,...,r_n)$ is true, if **(iii)** is the case then $\varphi(\frac{(t_i+t_j)}{2},r_2,...,r_n)$ is true and if **(iv)** is the case then $\varphi_{\infty}$ is true, i.e. $\varphi'(r_2,...,r_n)$ is true, which is what we had to prove.

**(<=)** Let for some real numbers $r_2,..,r_n$ , $\varphi'(r_2,...,r_n)$ be true. Then one of the disjunctions in $\varphi'(r_2,...,r_n)$ is true, i.e. $\exists x_1$, s.t. $\varphi(x_1,r_2,..,r_n)$, which is what we had to prove. □

**Corollary**: The problem of satisfaction decidability over ($\mathcal{R}$, +,>,=), where $\mathcal{R}$ is ordered, devisable and torsion-free group is decidable in DSPACE($2^{cn}$), where $n$ is the size of the formula and $c$ a constant.

*Proof:* Given en expression $\varphi(x_1,x_2,..,x_n)$ over $(\mathcal{R}, +,>,=)$, to decide if there exists a valuation which satisfies it, one has to eliminate all variables. If after eliminating all variables an expression evaluating to TRUE is computed, then the original expression is satisfiable, otherwise it is not. For a proof of the time and space bounds see [FeRa75]. □

Note that in our c-table definition we allow conditions over $(\mathbb{R},> ,=,+ )\cup(\mathbf{S},=,\neq)$. One way we can apply the algorithm of [FeRa75] to our case is to modify it to allow for string variables. In particular the formula $\exists x_1 \varphi(x_1,x_2,..,x_n)$ where $x_1$ is a string variable is equivalent to $\varphi(c_1,x_2,..,x_n)\vee\varphi(c_2,x_2,..,x_n)\vee\ldots\vee\varphi(c_k,x_2,..,x_n)\vee\varphi(d,x_2,..,x_n)$, where $c_1,..c_k$ are the string constants in $\varphi$ and $d$ is a new string constant different from the constants in $\varphi$. Note that the above reasoning cannot be applied if we allow lexicographical order comparison of strings in formulas, because there isn't a string between any two strings (e.g. there is no string between "*aa*" and "*aaa*").

## 2.3 Linear condition simplification

In the c-table normalization procedure, a way for simplifying linear conditions by removing redundant information will be needed. Note that a linear condition is a Boolean expression and as such it can be expressed as a disjunction of conjunctions.
An intuitive representation of a conjunction of conditions is a multi-dimensional polyhedra defining a semi-linear set. Therefore a linear condition can be thought as a set of disjoint polyhedras. Note however, that a given linear condition can be expressed as a set of polyhedras in more than one way. This is the reason it is impossible to define a canonical form for linear conditions, which has the property that two conditions are identical if and only if they have the same canonical form.

We will first examine an algorithm proposed in [LaMc92] for normalizing a conjunction of linear equalities and inequalities. More precisely a conjunction of atomic linear conditions can be represented by the system $Ax\leq b$, $Ex=d$, $\neg(c_ix=f)$, where $A$ and $E$ are matrices, $b$, $d$, $c_i$ and $f$ are vectors and $x$ is a variable vector. This system represents a polyhedra, and can be canonized by simplifying conditions and removing the redundant ones. The fundamental theorem from [LaMc92] is:

**Theorem 2.3.1** If two sets of atomic conditions over $(\mathbb{R},+,>,=)$ define the same point set, where $\mathbb{R}$ is the set of real numbers, their canonical forms will have identical set of equality conditions, the same inequality conditions up to multiplication by a positive scalar and the same set of negative conditions.

The above theorem can be used to check the equality of two conjunctions of atomic conditions. The proposed algorithm in [LaMc92] runs in polynomial time, can be implemented to run on parallel machines and relies on calls to a module that solves linear programs. As well the algorithm recognizes sets of unsatisfiable atomic conditions and reports them as such. Part of the algorithm deals with elimination of redundant conditions, which is an extension of the research done in [LHM89].

The algorithm we present for simplifying a linear condition $C$ is:

**1.** Convert $C$ in disjunctive normal form (i.e. as a disjunction of conjunctions) to get $C=c_1 \vee c_2 \vee .. \vee .c_n$.
**2.** Normalize each conjunction $c_i$ using the algorithm in [LaMe92]
**3.** Scan the conjunctions $c_i$ from left to right, adding and modifying conjunctions as needed. More precisely, the first iteration is to mark the first conjunction $c_1$ as processed. At the $k^{th}$ iteration, find the intersection of $c_k$ with each of the processed conjunctions. More precisely, if $g_1, ... g_m$ are the processed conjunctions so far, calculate the normal forms of $g_i \wedge c_k$, $g_i \wedge \neg c_k$, $\neg g_i \wedge c_k$ for $i$=1 to $m$, to compute the new set of processed conjunctions. Note that [LaMe92]'s procedure can return that some of the conditions are unsatisfiable and we can eliminate those.
**4.** The simplified value of $C$ will be $g_1 \vee g_2 \vee .. \vee g_m$ , where $g_1, g_2, ..., g_m$ are the processed conjunctions at the end of the algorithm in step 3. If there are no marked processed conjunctions, then $C$=*FALSE.*

Note that the above algorithm is exponential time, because [LaMe92] algorithm runs in polynomial time and $n + n3^n$ is an upper bound on the number of calls we make to it, where $n$ is the initial number of conjunctions. This is because in step 2 of the algorithm we make $n$ calls to the normalization procedure. In step 3, at the $k^{th}$ iteration ($k$>1) we may have as much as $3^{k-2}$ processed conjunctions and therefore as much as $3^{k-1}$ calls to the normalization procedure. So the whole step 3 can use at most $\sum_{k=2}^{n} 3^{k-1} \leq n3^n$ calls to the normalized procedure. Note that, since the problem of satisfaction decidability in general is DSPACE($2^{cn}$), where n is the size of the formula (see [Fera75]) and the above algorithm can be used for satisfaction checking we can not hope to improve the proposed procedure to run in polynomial time.

Unlike the case in section 2.2., the algorithm presented in this section can be applied to conditions over $(\mathbb{R}, >, =, +) \cup (\mathbf{S}, =, \neq)$ without any modification. The reason is that equality and inequality conditions are treated in the algorithm separately from the $\geq$ conditions.

## 2.4 C-table normalization:

Note that there may be different c-tables representing the same set of relational tables, i.e. it may be the case that $T_C' \neq T_C''$ but $Rep(T_C')=Rep(T_C'')$. In this situation, we will say that $T_C'$ and $T_C''$ are equivalent and we will write $T_C' \approx T_C''$. It is possible to derive a methodology for determining whether two c-tables are equivalent or not. The fact that this can be done in exponential time was proven in [Grah91]. We will present an alternative method for doing so by comparing the normalized forms of two c-tables. Theorem 2.4.2, which will be stated later in this section, tells how to determine if two normalized c-tables are equivalent or not. We propose the following algorithm for normalizing a c-table $T_C$:

**1.** If for some c-tuple $t_C$, the expression $lc(t_C) \wedge gc(t_C)$ is not satisfiable, then remove $t_C$ from $T_C$ (theorem 2.2.1. and its corollary show how linear expression satisfiability can be tested).

**2.** If for some c-tuples $t_C'$, $t_C'' \in T_C$, $t_C'$ and $t_C''$ are *unifiable*, then remove $t_C'$ and $t_C''$ from $T_C$ and add a new tuple with main part $\overline{X} = \{x_1, x_2, ..., x_n\}$, where $n$ is the arity of $T_C$ and $x_i$ are newly introduced variables, and with local condition $(\overline{X} = main(t_C') \wedge lc(t_C')) \vee (\overline{X} = main(t_C'') \wedge lc(t_C''))$. We define two c-tuples $t_C'$ and $t_C''$ to be unifiable iff the expression $main(t_C') = main(t_C'')$ is satisfiable (note that here = is generalized to denote vector equality) and $lc(t_C') \wedge lc(t_C'') \wedge T_G$ is not satisfiable.

**3.** After step 2 cannot be applied any more, look for triplets, quadruplets, and s.o. that are unifiable and combine them in a similar fashion.

**4.** Move $T_G$ to all local conditions, i.e. for every $t_C \in T_C$ set the local condition of $t_C$ equal to $lc(t_C) \wedge T_G$. Then normalize all local conditions using the algorithm presented in section 2.3. Put TRUE as the global condition of the resulting c-table.

**5.** For every c-tuple $t_C \in T_C$, if $main(t_C)$ contains a variable $x$ for attribute $A_i$ of $T_C$, and $lc(t_C) => (x=c)$ is a valid expression, where $c$ is a constant, then replace $x$ with $c$ in $main(t_C)$.

| A | B | C | condition |
|---|---|---|---|
| 1 | 2 | u | $x=1 \wedge y=3$ |
| z | 2 | v | $x=2 \wedge q=3$ |
| p | w | r | $x=t$ |
| 1 | 4 | 1 | $z \neq 1 \wedge v \neq u \wedge w \neq 2$ |

g.c.: $t \neq 1 \wedge t \neq 2$

| A | B | C | condition |
|---|---|---|---|
| z' | w' | u' | $((z'=1) \wedge (w'=2) \wedge (u'=u) \wedge (x=1) \wedge (y=3)) \vee$ $((z'=z) \wedge (w'=2) \wedge (u'=v) \wedge (x=2) \wedge (q=3)) \vee$ $((z'=p) \wedge (w'=2) \wedge (u'=r) \wedge (x=t))$ |
| 1 | 4 | 1 | $z \neq 1 \wedge v \neq u \wedge w \neq 2$ |

g.c.: $t \neq 1 \wedge t \neq 2$

***Table 2.4.1:*** *A c-table and the result of applying steps 1, 2 and 3 of the normalization process to it.*

The example in table 2.4.1 shows how the first three steps work over an example c-table. Step 4 will move the global condition and combine it with the local conditions and then normalize the local conditions. Step 5 will not be applied for this example.

**Theorem 2.4.1** The so presented normalization algorithm is correct, i.e. $Rep(Norm(T_C))=Rep(T_C)$ for any c-table $T_C$, where the function *Norm* is used to denote the normalization process. As well it runs in DSPACE($2^{cn}$) where $n$ is the size of $T_C$.

*Proof (Sketch)*: To prove the first part of the theorem we need to use induction and show that each of the above 5 rules are equivalence preserving - i.e. if $T_C$ is a c-table and $O_i$ is used to denote the application of the $i^{th}$ rule, we need to show that $O_i(T_C)\approx T_C$, for $i$=1 to 5. It is quite easy to verify that this is the case for $i$=1, 4 , 5. For $i$=2 note that the algorithm examines pair of c-tuples that are mutually exclusive, i.e. can not both appear in the same representation. This is because the pre-condition of applying this step is that for every valuation $v$, $v[lc(t_C')\wedge lc(t_C'')\wedge gc(T_C)]$=FALSE. Since there is a valuation $v$, that makes the main parts of the c-tuples equal, we can combine them and "join" their conditions. In this way at most one of the two original c-tuples together with its condition will appear in each representation, i.e. the set of representations of the c-table is not changed. The analyzes for $i$=3 are similar.

To prove the time bound note that in the proposed normalization procedure steps 1, 2 and 3 combined may require as much as $O(n+\binom{n}{1}+\binom{n}{2}+..\binom{n}{n})=O(2^n)$ iterations, where n is the number of tuples (this is because we first explore single c-tuples, then pairs of c-tuples, then triplets and s.o.). As well each iteration checks whether a condition is satisfiable, which as proven in section 2.2.1. can be done in DSPACE($2^{cn}$) time where n is the size of the formula and $c$ a constant. Step 4 does the normalization of the local conditions and can be performed in exponential time (see section 2.3). Step 5 can be performed in polynomial time and therefore the whole algorithm can be performed in DSPACE($2^{cn}$) where $n$ is the size of the c-table □

Note that event though the complexity of the above algorithm is high, things are not as bad as they look. Assuming that the number of variable occurrences in the normalized table is relatively small, and the local and global conditions are few and short the procedure will run relatively fast and its practical use will be feasible. The reason being, that the computational complexity of the algorithm comes from the symbolic manipulations of the conditions.

The procedure for equivalence checking between two c-tables $T_C'$ and $T_C''$ is:

**1.** Normalize $T_C'$ and $T_C''$
**2.** If the number of c-tuples in the two normalized tables is equal try to match them. Two c-tuple $t_C'$ and $t_C''$ match iff $main(t_C')=main(t_C'')\wedge lc(t_C')\wedge lc(t_C'')$ is satisfiable. An effective way to check this satisfiability is to first check that $main(t_C')=main(t_C'')$ is satisfiable which can be done in O($n$) time where $n$ is the arity of the c-tuples.
**3.** If all the c-tuples are matched, i.e. if each c-tuple from both c-tables participates in exactly one match, then $T_C'$ and $T_C''$ are equivalent, otherwise they are not.

**Theorem 2.4.2** The proposed algorithm for determining whether $T_C'\approx T_C''$ is correct and works in DSPACE($2^{cn}$) where $n$ is the sum of the size of the two c-tables and c is a constant.

*Proof (sketch)*: First note, that if two c-tables are equivalent, their normalized forms will have the same number of c-tuples. This is because the normalization procedure unifies all c-tuples that can appear as one in a representation. Therefore since the two c-tables have the same set of representations, it must be the case that they have the same number of c-tuples in their normalized form. Second note that that if the c-tuples are equivalent their must exist a matching function which matches the c-tuples of their normal forms.

To prove the time bound note that step 1 takes DSPACE($2^{cn}$) time, finding pair of tuples for which $main(t_C')=main(t_C'')$ can be done in polynomial time, however deciding the satisfiability of $lc(t_C')\wedge lc(t_C'')$ has bound DSPACE($2^{cn}$), where n is the length of the formula. Therefore the total running time of the algorithm is bounded by DSPACE($2^{cn}$) where *n* is the sum of the size of the two c-tables. □

**Corollary:** The problem of c-table equivalence can be decided in DSPACE($2^{cn}$) where *n* is the sum of the size of the c-tables and *c* is a constant.

This concludes our presentation of the normalization algorithm. In the next section we will explore the application of bag relational algebra to c-tables.

## 3. Bag relational algebra for c-tables

Now that we have defined the syntax and semantics of c-tables and have shown how they can be normalized we will show how relational algebra can be extended to handle c-tables (our choice of relational algebra is arbitrary, i.e. any language with the expressive power of relational algebra, e.g. relational calculus, can be extended to work with c-tables). In particular we will look for a sound and complete extension of relational algebra over Codd tables which preserves closure. By *closure preserving* we mean that for an arbitrary relational algebra query *q*, and for an arbitrary c-table $T_C$ we will define the semantics of *q*, so that $q(T_c)$ is a c-table. By *sound* we mean that we will define the semantics of *q*, so that only correct answers will appear in the result $q(T_C)$, or formally that $Rep(q(T_C))\subseteq q(Rep(T_C))$. Finally, by *complete* we mean that all correct answers will be in the result or that $q(Rep(T_C)) \subseteq Rep(q(T_C))$.

To define how relational algebra can be extended to handle c-tuples we will define the semantics of projection, selection, inner join, monus and duplicate elimination over c-tables with duplicate semantics. The presented definitions and proves are an extension of the ones presented in [ImLi84]. More specifically we deal with c-tables ([ImLi84] deals with v-tables that don't have global conditions), closed world assumption ([ImLi84] works with the open world assumption), duplicate tuple semantics ([ImLi84] doesn't explore duplicate semantics) and we allow + in the condition expressions ([ImLi84] doesn't). A summary of the presented relational algebra operations for the closed world assumption for c-tables, but for sets of c-tuples and conditions over ($\mathcal{R}$,>,=), where $\mathcal{R}$ is an ordered set can be found in [Grah91].

### 3.1 Projection

If $T_C$ is a c-table with attributes $\overline{A}=A_1, ..A_n$ we denote the projection over this c-table of the attributes $\overline{A}' \subseteq \overline{A}$ as $\pi_{\overline{A}'}(T_C)$. The c-table $\pi_{\overline{A}'}(T_C)$ is constructed from the tables $T_C$ by removing all columns in $\overline{A} - \overline{A}'$, and leaving the same local and global conditions. To prove that the projection is well define we have to prove that $Rep(\pi_{\overline{A}'}(T_C))=\pi_{\overline{A}'}(Rep(T_C))$ or that the sets $S_1=\{T \mid \exists\ v, \text{ s.t. } v(\pi_{\overline{A}'}(T_C))=T\}$ and $S_2=\pi_{\overline{A}'}(\{T \mid \exists\ v, \text{ s.t. } v(T_C)=T\})$ are equivalent. Let $T \in S_1=\{T \mid \exists\ v, \text{ s.t. } v(\pi_{\overline{A}'}(T_C))=T\}$, then $T$ is a representation of $\pi_{\overline{A}'}(T_C)$ for some evaluation $v$, i.e. for some $v$, $v(\pi_{\overline{A}'}(T_C)) = T$. Let $T'$ be a table constructed from $T$ by adding the columns for the attributes $\overline{A} - \overline{A}'$ and filling them with arbitrary values. Then by the definition of projection over relational tables we have that $\pi_{\overline{A}'}(T')=T= v(\pi_{\overline{A}'}(T_C))$. As well note that $T'$ is a representation of $T_C$ for a valuation $v'$ extending the valuation $v$ to the attributes $\overline{A} - \overline{A}'$, i.e. $T'=v'(T_C)$, which means $\pi_{\overline{A}'}(v'(T_C))=T= v(\pi_{\overline{A}'}(T_C))$, i.e. there exists a valuation $v'$ s.t. $\pi_{\overline{A}'}(v'(T_C))=T$, and therefore $T \in S_2$, which is what we wanted to prove. Proving the reverse direction is similar.

Trough out this section we will work with the pair of tables shown in Table 3.1.1.

$R_1$=

| **A** | **B** | **condition** |
|---|---|---|
| 2 | $x$ | $x \neq 3$ |
| 2 | 4 | $y>3$ |

g.c. $x \neq 2$

$R_2$=

| **B** | **C** | **condition** |
|---|---|---|
| 4 | 1 | TRUE |
| 2 | $z$ | $y>3$ |

g.c. $z>3$

***Table 3.1.1*** *Example c-tables on which the application of the different bag relational operators will be defined*

By applying the above definition of bag projection to calculate $\pi_B(R_2)$ we get:

| **B** | **condition** |
|---|---|
| 4 | TRUE |
| 2 | $y>3$ |

g.c. $z>3$

***Table 3.1.2*** *The value of $\pi_B(R_2)$*

## 3.2 Selection

We denote a selection over a c-table $T_C$ as $\sigma_\gamma(T_C)$ where $\gamma$ is a predicate formula over $(\mathbb{R},>,=,+) \cup (\mathbf{S},=,\neq)$ and variables $A_1,..A_n$ having the same names as the attributes of $T_C$. We construct $\sigma_\gamma(T_C)$ from $T_C$ by keeping the same global condition and adding (by conjunction) the local condition $\gamma_{\theta(t_C)}$ to each tuple $t_C$ from $T_C$ where $\theta(t_C)$ is a substitution that substitutes $t_C[A_i]$ (the value for the attribute $A_i$ in $t_C$) for each occurrence of $A_i$. To prove that we have defined selection correctly we have to prove that the result of applying standard relational selection over the possible representations of a c-table is the same as taking the representations of the so defined selections over the same c-table or that $Rep(\sigma_C(T_C))=\sigma_C(Rep(T_C))$ for any c-table $T_C$ and condition $C$ over $(\mathbb{R},>,=,+)\cup(\mathbf{S},=,\neq)$. But this is equivalent to proving that there exist valuations $v$ and $v'$ (see the definition of *Rep*), s.t. $v(\sigma_C(T_C))=\sigma_C(v'(T_C))$. However it can be easily proven that we have defined selection over c-tables in such a way that $v(\sigma_C(T_C))=\sigma_C(v(T_C))$ for any valuation $v$, which proves that we have defined selection correctly.

The result table bellow demonstrates how selection works over the c-table $R_2$, defined in Table 3.1.1.

| B | C | condition |
|---|---|---|
| 4 | 1 | 1>2 |
| 2 | $z$ | $y>3\wedge z>2$ |

g.c. $z>3$

***Table 3.2.1** Result of $\sigma_{C>2}(R_2)$*

## 3.3. Inner Join

We denote the inner join between two c-tables $T_C'$ and $T_C''$ as $T_C'\bowtie T_C''$. We would like to define inner join in such a way that $Rep(T_C'\bowtie T_C'')= Rep(T_C')\bowtie Rep(T_C'')$. The algorithm we propose for calculating the inner join $T_C'\bowtie T_C''$ is:

**1.** Rename all variables of $T_C''$, so that $Var(T_C')\cap Var(T_C'')$ is empty, i.e. make sure that the variables of the two c-tables are distinct.
**2.** Do the join, i.e. for each c-tuple of $T_C'$, find all c-tuples of $T_C''$, s.t. there exists a valuation that makes the join attributes equal and insert a c-tuple in the result containing the main body of the join of the two c-tuples, and a condition a conjunction of the local condition of the two c-tuples that are joined plus the corresponding equality condition which makes the join attributes equal.
**3.** Add a global condition to the result table comprised of a conjunction of the global conditions of $T_C'$ and $T_C''$.

To illustrate how the algorithm works consider a join of the c-tables $R_1$ and $R_2$ defined in Table 3.1.1. The resulting c-table is shown in Table 3.3.1:

| A | B | C | condition |
|---|---|---|---|
| 2 | x | 1 | $x\neq 3\wedge x=4$ |
| 2 | $x$ | z | $x\neq 3\wedge x=2\wedge y>3$ |
| 2 | 4 | 1 | $y>3\wedge$TRUE |

g.c. $x\neq 2\wedge z>3$

**Table 3.3.1** The result of $R_1\bowtie R_2$

## 3.4. Monus

Monus is the difference operators for bags. In bag relational algebra over Codd tables it is defined as: $T' \dot{-} T''=\{t_{[k]} \mid t\in T'$ and $k=max(count(t,T')\text{-}count(t,T''),0)$, where $t_{[k]}$ is used to denote the tuple $t$, repeated $k$ times and *count* is a function which returns the number of occurrences of the tuple specified as the first parameter in the table specified as the second parameter. Now, to extend this definition to $V=T_C' \dot{-} T_C''$ (i.e. to c-tables) we propose the following algorithm:

**1.** Rename all variables of $T_C''$, so that $Var(T_C') \cap Var(T_C'')$ is empty, i.e. make sure that the variables of the two c-tables are distinct.
**2.** Copy $T_C'$ in the result c-table $V$.
**3.** Construct a matrix $X[i,j]$, where $1\le i\le n=|T_C'|$, $1\le j\le m=|T_C''|$, where $|.|$ is the operator that calculates the number of c-tuples in a c-table. Set $X[i][j] = [main(t_i')=main(t_j'')] \wedge lc(t_j') \wedge gc(T_C') \wedge lc(t_j'') \wedge gc(T_C'')$
**4.** Add to $V$ the global condition:

$$\bigwedge_{j=1}^{m}[\bigvee_{i=1}^{n}(Y[1,j]=..=Y[i-1,j]=Y[i+1,j]=..=Y[n,j]=0\wedge Y[i,j]=1)]\wedge$$

$$\bigwedge_{i=1}^{n}[\bigvee_{j=1}^{m}(Y[i,1]=..=Y[i,j-1]=Y[i,j+1]=..=Y[i,n]=0\wedge Y[i,j]=1)],$$

where $Y$ is a newly introduced matrix of variables, having the same dimensions as $X$.
**5.** Add with conjunction to the $i^{\text{th}}$ c-tuple of $V$ the local condition $\urcorner[\bigvee_{j=1}^{m}(X[i,j]\wedge(Y[i,j]=1))]$

What the algorithm does, is to first rename to variables of $T_C''$, so that they become distinct from those of $T_C'$. Then it calculates the matrix $X$ and sets a restriction of the possible values for the matrix $Y$. The value of $X[i,j]$ contains the condition which must hold for the c-tuple $t_i'$ to be deleted from $T_C'$ and the tuple that deletes it to be $t_j''$. The matrix $Y[i][j]$ has the restriction that for each $j$, there exists exactly one $i$, s.t. $Y[i][j]=1$, and that for each $i$, there exists exactly one $j$, s.t. $Y[i][j]=1$. As well, the elements of the matrix $Y$ can only take the values 0 and 1. The matrix $Y$ is used to enforce the condition that every c-tuple $t_j''$ in $T_C''$ can be responsible for the deletion of at most one c-tuple of $T_C'$ and that every c-tuple $t'$ in $T_C'$ can be deleted at most once. Finally the local condition we add to the result table specify that if for some valuation, $X[i][j]$ and $Y[i][j]$ holds, i.e. if the tuple $t_i'$ in $T_C'$ matches with the tuple $t_j''$ in $T_C''$ and the valuation is such that $t_i$ can not be deleted by any tuple other then $t_j$, and $t_j$ can not delete other tuple then $t_i$, then $t_i$ should be deleted from the result, i.e. $X[i][j]\wedge(Y[i][j]=1)$ will be true. This guarantees that each tuple in $T_C'$ may disappear from a representation only if for the corresponding valuation there exists a matching tuple in $T_C''$ which evaluates to TRUE and this matching tuple $t_C''$ will delete exactly one tuple from $T_C'$, i.e. the algorithm is correct and $Rep(T_C' \dot{-} T_C'') = [Rep(T_C') \dot{-} Rep(T_C'')] \cup \{\varnothing\}$. Here $\{\varnothing\}$ is used to represent the empty table which will be constructed from a valuation for which the global condition is FALSE. We don't have perfect equality in the above formula because from the way we construct the global condition of $T_C' \dot{-} T_C''$ we allow for $\{\varnothing\}$ to be a possible representation. This is because there are valuations, for which the global condition of $T_C' \dot{-} T_C''$ our algorithm constructed will not hold. It is our believe this is an inherent problem and there is no way for it to be solved under the definition of c-tables we have adapted..

To give an example, if we have our example tables $R_1$ and $R_2$ from table 3.1.1 we can calculate $R_1 \dot{-} R_2$ as follows:

| | |
|---|---|
| $x=4 \wedge z>3 \wedge x \neq 3$ | $x=2 \wedge y>3 \wedge z>3 \wedge x \neq 3$ |
| $z>3 \wedge y>3 \wedge x \neq 2$ | FALSE |

| A | B | condition |
|---|---|---|
| 2 | $x$ | $x \neq 3 \wedge \neg((X[1][1] \wedge Y[1][1]) \vee (X[1][2] \wedge Y[1][2]))$ |
| 2 | 4 | $y>3 \wedge \neg((X[2][1] \wedge Y[2][1]) \vee (X[2][2] \wedge Y[2][2]))$ |

g.c. $(x \neq 2) \wedge ((Y[1][1]=Y[2][2]=1 \wedge Y[1][2]=Y[2][1]=0) \vee$
$(Y[1][2]=Y[2][1]=1 \wedge Y[1][1]=Y[2][2]=0))$

***Table 3.4.1*** *Shows the values of the matrix X used in calculating $R_1 \dot{-} R_2$ and the value of the resulting c-table $R_1 \dot{-} R_2$*

## 3.5 Duplicate Elimination

The last relational algebra operator which we will explore is duplicate elimination and we will denote it as $\varepsilon(T_C)$. Note that duplicate elimination can be defined as a grouping by all the attributes in the relational case. In the case of c-tables we define $\varepsilon(T_C)= group_{A1,\dots,An}(T_C)$, where $A_1,\dots,A_n$ are the attributes of $T_C$. Then $\varepsilon(\text{Rep}(T_C))= group_{A1,\dots,An}(\text{Rep}(T_C))=Rep(group_{A1,\dots,An}(T_C))=Rep(\varepsilon(T_C))$, which proves that duplication elimination is well defined. The fact that $group_{A1,\dots,An}(\text{Rep}(T_C))=Rep(group_{A1,\dots,An}(T_C))$ follows from the fact that group is well defined over c-tables. We define the semantics of grouping over c-tables and the operator group is section 4.2.
The result of $\varepsilon(R_1)$, where $R_1$ is the table defined in table 3.1.1 is:

| A | B | condition |
|---|---|---|
| 2 | $x$ | $x \neq 3 \wedge (x \neq 4 \vee y \leq 3)$ |
| 2 | 4 | $y>3 \wedge (x \neq 4 \vee x=3)$ |

g.c. $x \neq 2$

***Table 3.5.1*** *The value of $\varepsilon(R_1)$*

## 3.6. Summary

In this section we have presented algorithms for implementing bag relational algebra operations over c-tables with duplicates. The presented operators are the primitive operators (see [LiWo94]), i.e. all bag relational algebra operators can be expressed as a function of the presented operators. In the next section we will justify the duplicate semantics for c-tables we have introduced by presented algorithms for aggregating over c-tables with duplicates.

## 4. Applying aggregation to c-tables

In present literature we weren't able to find any reference to applying aggregation to c-tables, or to incomplete databases for that matter. There is some research done in the area of applying aggregation to fuzzy numbers ([KCY97]) and to random variables ([Sprin79]), but the query results in those methods are approximations. On the other hand, the research done in constraint databases ([KLP98]) has explored the problem of aggregating over constraint databases. Unfortunately, the operation of aggregation in most constraint database systems is not closure preserving (e.g. first order formulas with linear or polynomial constraints - [Kupe93]).

### 4.1. Introductory examples

In this section we present algorithms for extending grouping and aggregation from bag relational algebra over Codd tables to bag relational algebra over c-tables. Let's look at an example: suppose we want to calculate $V={}_A\mathcal{F}_{sum\ (B)}\ R_1$, where $R_1$ is the c-table defined in table 3.3.1. The result is:

| A | sum(B) | condition |
|---|---|---|
| 2 | $z$ | $x\neq 3\wedge y>3\wedge z=x+4$ |
| 2 | $x$ | $x\neq 3\ \wedge y<=3$ |
| 2 | 4 | $y>3\wedge x=3$ |

g.c. $x\neq 2$

***Example 4.1.1*** *Shows the value of* $V={}_A\mathcal{F}_{sum\ (B)}\ R_1$

From this example, several important observations can be made. The most striking is that we can generalizing the above example to show that c-tables are indeed closed under the standard aggregation functions, as long as the allowed conditions are over $(\mathbb{R},>,=,+)\cup(\mathbf{S},=,\neq)$ and aggregation is allowed only over attributes defined over the $\mathbb{R}$, except for the aggregate *count* which is allowed over string attributes as well. Note however that c-tables are not closed over aggregation if the conditions are defined over $(\mathbb{R},>,=)$, which is the case in existing literature on c-tables (see [ImLi94],[Grah91]).

### 4.2. Basics of Grouping

In general we would like to be able to evaluate a relational query of the form ${}_{A_1,A_2,\ldots,A_k}\mathcal{F}_{agg(A_{k+1}),\,agg(A_{k+2}),\ldots,\ agg(A_n)}\ T_C$ where $A_1 \ldots A_n$ are the attributes of $T_C$ and *agg* is one of the operators *min*, *max*, *sum*, *count* and *avg*. In the relational case the above expression is evaluated by grouping tuples having the same values for $A_1$,..,$A_k$ into one tuple, which has this common value for its first $k$ attributes, and the rest of the fields are calculates by applying the corresponding aggregations over each group of tuples. To extend this definition to c-tables we will need to be able to group c-tuples and perform aggregation on c-tuples. In this sub-section we will explore how the grouping can be done and in the next sub-sections how aggregation can be applied to c-tuples.

Let's denote the result of grouping by the first $k$ attributes of $T_C$ as $group_{A_1,A_2,\dots,A_k}\, T_C$. The result of this operation will be a complex c-table, i.e. the value of a field may be a beg of values (see [Maki77]). More precisely, in the result of the group operation the values for the attributes $A_1$,$A_2$...,$A_k$ will be single values and for the attributes $A_{k+1}$,..,$A_n$ - bag of values. For example we would expect $group_B R_1$, where $R_1$ is defined in table 3.1.1. to be the following complex c-table:

| **A** | **B** | **condition** |
|---|---|---|
| 2 | $x$ | $x \neq 3 \wedge (x \neq 4 \vee y <= 3)$ |
| 2 | 4 | $y > 3 \wedge (x \neq 4 \vee x = 3)$ |
| 2<br>2 | 4 | $x \neq 3 \wedge y > 3 \wedge x = 4$ |

g.c. $x \neq 2$

*Table 4.2.1 The result of $group_B R_1$, where $R_1$ is defined in 3.1.1*

The algorithm for computing $V = group_A\ T_C$, where $A \cup B$ are the attributes of $T_C$ and $A = \{A_1, \dots, A_u\}$, $B = \{A_{u+1}, \dots, A_w\}$follows:

**1.** Copy $T_C$ into $V_C$.
**2.** Cluster the c-tuples of $V_C$ into biggest bags of semi-unifiable tuples relative to $A$ - we will call this *e-bags*. Two c-tuples $t_C$' and $t_C$'' are semi-unifiable relative to the set of attributes $A$ iff $main(\pi_A(t_C'))=main(\pi_A(t_C'))$ is a satisfiable formula. If a c-tuple belongs to more than one e-bag, then make copies of the c-tuple and put a copy in each e-bag. To do so add the local condition $x=i$ to the $i$th copy of the c-tuple for $i<k$ and the local condition $\overset{k-1}{\underset{i=1}{\wedge}} x \neq i$ to the $k$th copy, where $x$ is a newly introduced variable and $k$ is the number of times the c-tuple is copied.
**3.** Normalize the set of c-tuples formed in each e-bag (see section 2.4).
**4.** Partition each e-bag further into *r-bags*. To do so extract from the canonical form of the linear condition $\overset{n}{\underset{i=1}{\vee}} lc(t_i)$ (see section 2.3) the set of different conjunctions $C=\{c_i\}_{i=1..m}$, where $\{t_i\}_{i=1 \text{ to } n}$ are the c-tuples in the e-bag. From $C$ form the array $D$, such that the elements of $D$ are distinct and each element of $D$ is a disjunction of one or more conjunctions from $C$. More precisely, a disjunction of $c_i$s belongs to $D$ iff all the $c_i$s are from the same set of c-tuples' local conditions, are from only those c-tuples' local conditions and if a new $c_i$ is added to the set, one of the mentioned properties will no longer hold. Reconstruct $V_C$, by substituting each e-bag with a number of r-bags, so that the $j$th r-bag corresponds to $D[j]$. In that r-bag all the c-tuples will have the same local

condition $D[j]$ and the main parts will correspond to the c-tuples, from which $D[j]$ was formed.

**5.** From each r-bag form a set of nodes, each node corresponding to a distinct c-tuple in the r-bag (i.e. for duplicate c-tuples we will have a single node). Find all spanning undirected graphs for those nodes that are transitive and have the property that if there is an edge between the nodes $n_1$ and $n_3$ and there exists a third node $n_2$ such that $n_1 \prec_A n_2$ and $n_2 \prec_A n_3$ then there must be edges between $n_1$ and $n_2$ and between $n_2$ and $n_3$. By definition $n' \prec_A n''$ holds, where $n'$ and $n''$ are nodes corresponding to the distinct c-tuples $t_C'$ and $t_C''$ and $A$ a set of attributes, iff $\pi_A(t_C'')$ can be constructed from $\pi_A(t_C')$ by giving values to some of the variables in $t_C'$ (note that $\prec_A$ is a partial order). For example (1 2 $x$ 5) $\prec_{(A,B,C)}$ (1 2 3 4), where $A$,$B$,$C$ are the first three attributes of the c-tuples and $x$ is a variable.

**6.** Partition each r-bag $r$ in $V_C$ further into *f-bags*, where the set of f-bags corresponds to the set of graphs associated with $r$ constructed in step 5. More precisely, given a r-bag $r$, and a possible graph $G$ associated with it, a f-bag consists of a bag of complex c-tuples, each complex c-tuple corresponding to a complete sub-graph of G (since $G$ is transitive it is a disjoint set of complete sub-graphs). If the nodes in a complete sub-graph belong to the c-tuples $t_1,t_2,...,t_p$ , the corresponding complex c-tuple will have the single value $(x_1,x_2,...,x_u)$ for the attributes $A_1,...,A_u$ by which the grouping is done, the bag of values $\{|\pi_{A_{u+1},A_{u+2},...,A_w}(main(t_i))|\}_{i=1...p}$ for the attributes $A_{u+1},..A_w$ and the local condition $L_r \wedge \bigwedge_{i=1}^{p}[(\pi_{A_1,...,A_u} main(t_i)) = (x_1,x_2,..,x_u)]$ where $L_r$ is the local condition of the r-bag $r$.

Before giving an explanation of why the above algorithm is correct, i.e. why $group_{A_1,...,A_u}(Rep(T_C)) = Rep(group_{A_1,...,A_u}(T_C))$ holds, let's give an example of how the above algorithm can be applied to a concrete example. If we want to calculate $group_{A,B}(R)$, where R is given in table 4.2.2 we will first set $V_C=R$ and then cluster the c-tuples into e-bags relative to the attributes $A,B$ to get the e-bags shown in table 4.2.3. In the example $t$ is a newly introduced variable that is used to make two copies of the first tuple of $R$.

| **A** | **B** | **C** | **condition** |
|---|---|---|---|
| $x$ | $y$ | 1 | $(x+y=3)\vee(x>4)\vee(x<0)$ |
| $x$ | 3 | 2 | $[x+y=3\wedge x<2]\vee x<0$ |
| 2 | 3 | 3 | $x>5$ |
| 2 | 3 | 4 | TRUE |
| 3 | 4 | 5 | TRUE |

*Table 4.2.2. C-table R*

| A | B | C | condition |
|---|---|---|---|
| $x$ | $y$ | 1 | $[(x+y=3)\vee(x>4)\vee(x<0)]\wedge t=1$ |
| $x$ | 3 | 2 | $[x+y=3\wedge x<2]\vee x<0$ |
| 2 | 3 | 3 | $x>5$ |
| 2 | 3 | 4 | TRUE |

| A | B | C | condition |
|---|---|---|---|
| $x$ | $y$ | 1 | $[(x+y=3)\vee(x>4)\vee(x<0)]\wedge t\neq 1$ |
| 3 | 4 | 5 | TRUE |

*Table 4.2.3 Shows the e-bags relative to A,B for the c-table R in table 4.2.1*

Applying step 4 to the first e-bag, we will get $C$ as the array shown bellow: (the first row show $C$[0-5], the second $C$[6-10])

| | | | | | |
|---|---|---|---|---|---|
| $x<0\wedge t=1$ | $0\leq x<2\wedge x+y=3\wedge t=1$ | $2\leq x\leq 4\wedge x+y=3\wedge t=1$ | $4<x\leq 5\wedge t=1$ | $x>5\wedge t=1$ | $x+y\neq 3\wedge 0\leq x\leq 4$ |
| $x<0\wedge t\neq 1$ | $0\leq x<2\wedge x+y=3\wedge t\neq 1$ | $2\leq x\leq 4\wedge x+y=3\wedge t\neq 1$ | $4<x\leq 5\wedge t\neq 1$ | $x>5\wedge t\neq 1$ | |

and $D$ as the array bellow: (the second row shows the values of D and the first for which c-tuples are the corresponding disjunctions achieved)

| | | | | | |
|---|---|---|---|---|---|
| {1,2,4} | {1,4} | {1,3,4} | {2,4} | {3,4} | {4} |
| $C[0]\vee C[1]$ | $C[2]\vee C[3]$ | $C[4]$ | $C[6]\vee C[7]$ | $C[10]$ | $C[5]\vee C[8]\vee C[9]$ |

Similarly applying rule 4 to the second e-bag we get $C[0]=(x<0\wedge t\neq 1)$, $C[1]=(0\leq x<4\wedge x+y=3\wedge t\neq 1)$,$C[2]=(x\geq 4\wedge t\neq 1)$,$C[3]=$ $(x+y\neq 3\wedge 0\leq x\leq 4\wedge t\neq 1)$, C[5]= $(t=1)$ and $D[0]=C[0]\vee C[1]\vee C[2]$, $D[1]=C[3]\vee C[5]$. The corresponding r-tables are shown in example 4.3.3.

| A | B | C | condition |
|---|---|---|---|
| $x$ | $y$ | 1 | $(x<0\wedge t=1)\vee\ (0\leq x<2\wedge x+y=3\wedge t=1)$ |
| $x$ | 3 | 2 | |
| 2 | 3 | 4 | |
| $x$ | $y$ | 1 | $[(2\leq x\leq 4)\wedge(x+y=3)\wedge t=1]\vee\ [(4\leq x\leq 5)\wedge(t=1)]$ |
| 2 | 3 | 4 | |
| $x$ | 3 | 1 | $x>5\wedge t=1$ |
| 2 | 3 | 3 | |
| 2 | 3 | 4 | |
| $x$ | 3 | 2 | $(x<0\wedge t\neq 1)\vee(\ 0\leq x<2\wedge x+y=3\wedge t\neq 1)$ |
| 2 | 3 | 4 | |
| 2 | 3 | 3 | $x>5\wedge t\neq 1$ |
| 2 | 3 | 4 | |
| 2 | 3 | 4 | $(x+y\neq 3\wedge 0\leq x\leq 4)\vee(\ 2\leq x\leq 4\wedge x+y=3\wedge t\neq 1)\vee$ $(4<x\leq 5\wedge t\neq 1)$ |

| **A** | **B** | **C** | **condition** |
|---|---|---|---|
| $x$ | $y$ | 1 | $(x+y=3\wedge t\neq 1)\vee(x>4\wedge t\neq 1)\vee(x<0\wedge t\neq 1)$ |
| 3 | 4 | 5 | |
| 3 | 4 | 5 | $x+y\neq 3\wedge\ 0\leq x\leq 4$ |

*Table 4.3.3 The partitioning of the e-bags from table 4.2.3 into r-bags*

For the possible graphs of the first r-bag in the first e-bag we may have three possible graphs with edges {(1,2)}, {(2,1)} or {(1,2),(2,3),(1,3)} (here $(a,b)$ is used to denote an edge between the nodes corresponding to the $a$th and $b$th c-tuple of the r-bag). The corresponding complex c-tuples for this r-bag are shown in table *4.3.4*

| **A** | **B** | **C** | **condition** |
|---|---|---|---|
| $z$ | $w$ | 1 | $[(x<0\wedge t=1)\vee\ (0\leq x<2\wedge x+y=3\wedge t=1)]\wedge(z=x\wedge w=y=3)$ |
| $w$ | | 2 | |
| $z$ | $w$ | 2 | $[(x<0\wedge t=1)\vee\ (0\leq x<2\wedge x+y=3\wedge t=1)]\wedge(z=x=2\wedge w=3)$ |
| | | 4 | |
| $z$ | $w$ | 1 | $[(x<0\wedge t=1)\vee\ (0\leq x<2\wedge x+y=3\wedge t=1)]\wedge(z=x=2\wedge w=y=3)$ |
| | | 2 | |
| | | 3 | |

*Table 4.3.4 Shows the complex c-tuples for the first r-bag in table 4.3.3*

The rest of the computations for $group_{A,B}(R)$ are done similarly and are not shown.

To substantiate why the proposed algorithm for calculating $group_A(T_C)$ works where $A$ is a sub-set of the attributes of $T_C$, let's look at its steps. Step 1starts the algorithm by copying $T_C$ in the result c-table. Step 2 clusters the c-tuples into e-bags. relative to the attributes A. Note that a set of c-tuples will appear in the same complex c-tuple after the grouping is done, iff the c-tuples are part of the same e-bag. Step 3 normalizes the e-bags, where part of the normalization is the removing of the global condition of the c-table.

Step 4, partitions each e-bag further into r-bags. What is done in this step is that the space over which the local conditions of the c-tuples in the e-bag is defined is partitioned into non-overlapping polyhedras. Each r-bag corresponds to single polyhera, or to a disjunction set of polyhedras, so that this is the biggest linear space over which all the local conditions of the included c-tuples are true, and non of the local conditions of the other c-tuples in the r-bag are true. Note that a set of c-tuples will appear in the same complex c-tuple after the grouping is done, iff the c-tuples are part of the same r-bag. As well the r-bags partition the possible valuations, i.e. under every valuation the condition of at most one r-bag of every e-bag will be true.

Next, what step 5 and 6 do is to examine the c-tuples in each r-bag and see under different valuations which c-tuples will much. Each of the constructed graphs corresponds to a valuations. In a graph there is an edge between two nodes if under the corresponding valuation it is true that $\pi_A(\text{main}(t_C'))=\pi_A(\text{main}(t_C''))$, where $t_C'$ and $t_C''$ are the corresponding c-tuples. It can be proven that a graph is valid, i.e. a corresponding valuation exists iff (1) the graph is transitive (2) if there is an edge between the nodes $n_1$ and $n_3$ and there exists a third node $n_2$ such that $n_1 \prec_A n_2$ and $n_2 \prec_A n_3$ then there are edges between $n_1$ and $n_2$ and between $n_2$ and $n_3$. This is why all graphs having those two properties are constructed and those graphs show which c-tuples in the r-bag will be grouped relative to the attributes *A*, relative to the different valuations.

In this sub-section we examined how grouping can be done for c-tables. In the next sub-section we will extend this knowledge to show how once the grouping has been done aggregation can be performed.

## 4.3. Performing the aggregation

Now, that we have define how grouping over c-tables can be done, aggregation is straight-forward. Let's have the expression $V={}_{A_1,A_2,\ldots,A_k}\mathcal{F}_{agg(A_{k+1}),\,agg(A_{k+2}),\ldots,\,agg(A_n)}(T_C)$, where the *agg*s are linear expressions and the sets $A=\{A_1,\ldots A_k\}$ and $B=\{A_{k+1},\ldots,A_n\}$ are disjoint and their union yields all the attributes in $T_C$. Then we can calculate the result by first computing the complex c-table $group_A T_C$, and then aggregating over the *B* attributes by introducing new variables in the main part of the result and moving the aggregations to the local condition. This can be done because we allow linear conditions in the local conditions of the c-tables. More precisely, suppose we have the following complex c-tuple in $group_A T_C$

| **A** | **B** | **condition** |
|---|---|---|
| a | $b_1$ | *c* |
| | ... | |
| | $b_n$ | |

*Example 4.3.1. A general complex c-tuple*

Then this complex c-tuple will appear in $V={}_A\mathcal{F}_{agg\ (B)}\ T_C$ as:

| **A** | **B** | **condition** |
|---|---|---|
| *a* | *x* | $c \wedge con(x,agg,b_1,\ldots,b_k)$ |

*Table 4.3.2. The result of applying aggregation of the form $\mathcal{F}_{agg\ (B)}\ T_C$ to the complex c-tuple from table 4.3.1*

The table bellow shows how the condition *con* is formed for the different values of *agg*:

| ***con*** | $\boldsymbol{con(agg,b_1,\ldots,b_k)}$ |
|---|---|
| *min* | $\bigwedge_{i=1}^{n}(x \le b_i)$ |
| *max* | $\bigwedge_{i=1}^{n}(x \ge b_i)$ |
| *count* | n |
| *sum* | $x=\sum_{i=1}^{n} b_i$ |
| *avg* | $x+x+\ldots+x=(\sum_{i=1}^{n} b_i)$ (where *x* is added *n* times) |

*Table 4.3.3. Show how the function* ***con*** *is computed*

For an example of applying aggregation to c-tables see the example in table 4.1.1. The reasoning of why the aggregation as defined above indeed produces the correct c-table, i.e. why $Rep({}_{A_1,A_2,\ldots,A_k}\mathcal{F}_{agg(A_{k+1}),\, agg(A_{k+2}),\ldots,\; agg(A_n)}(T_C)) = {}_{A_1,A_2,\ldots,A_k}\mathcal{F}_{agg(A_{k+1}),\, agg(A_{k+2}),\ldots,\; agg(A_n)}$ $Rep(T_C)$ is straight forward and follows from the correctness of the grouping algorithm and the correctness of the *con* operator as defined in table 4.3.3. The complexity of the proposed algorithm is exponential, the reason being it contains calls to the computationally expensive procedures from sections *2.3* and *2.4.*

### 4.4. Summary

In this section we have presented an algorithm for aggregating over c-tables. Though the presented algorithm is computationally hard, its complexity is dependant on the size of the uncertain information, which means that if it is small we could expect the algorithm to run in reasonable time. This concludes our overview of querying c-tables. It remains to summarize the presented material and mention areas worthy of further investigation.

## 5. Conclusion and future research

In this paper we have presented algorithms for querying c-tables extended with linear conditions. We have chose this representation because it is the leas expressive representation of incomplete information for which bag relational algebra with aggregation has closure and is well defined. As expected, the running time of most of the presented algorithms is non-polynomial, the reason being that manipulating linear expressions is inherently computationally expensive.

We outline the following topics, which we believe are a natural extension of the presented work:

- improving the efficiency of the presented algorithms; possible tradeoff for doing so can be:
  - loosing some of the semantics for the uncertain information
  - approximating the result
- exploring integrity constraints for incomplete information and how they can be used for semantics query optimization
- extending research done in relational databases to incomplete databases, i.e. view maintenance, view update, transaction control, integrity constraint validation, etc.
- exploring semantics for c-tables with order

## References:

**[Bisk81]** Biskup, J. A Formal approach to null values in database relations. In *Advances in Database Theory*, H. Gallaire, J.Minker, and J.M.Nicolas, Eds. Plenum Press, New York, pp. 299-341, 1981

**[Codd75]** Codd, E.F. Understanding relations (Installment #7). *FDT Bull. of ACM-SIGMOD 7*, 3-4 (Dec. 1975), pp. 23-28, 1975

**[Codd79]** Codd E.F. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst. 4,* 4 (Dec. 1979), pp. 397-434, 1979

**[FeRa75]** Jerrante F., Rackoff C., A Decision Procedure for the First Order Theory of Real Addition with Order, *SIAM J. Comp*, 4:1:69-76, 1975

**[Grah84]** G. Grahne. Dependency satisfaction in databases with incomplete information. In *Proc. of Intl. Conf. on Very Large Data Bases*, pages 37-45, 1984

**[Gran77]** Grant. J. Null values in relational data base. *Inf. Process. Lett*. 6,5 (Oct. 1977), pp. 156-157, 1977

**[Grah91]** Grahne G. *"The problem of Incomplete Information in Relational Databases"*. Springer-Verlag, Berlin, 1991

**[ImLi81]** Imielinski, T., Lipski W On representing incomplete information in a relational data base. In *Proceedings of the 7th International Conference on Very Large Data DBases* (Cannes, France, Sept. 9-11) ACM, New York, 1981, pp. 388-397, 1981

**[ImLi84]** Imielinski, T., Lipski W., Incomplete information in relational databases, *J. Assoc. Comput.* March. 31 4, Oct. 1984, pp. 761-791, 1984

**[KCY97]** Klir G., Clar U., Yuan B., Fuzzy set Theory. Foundations and Applications, Prentice Hall, 1997

**[KKR90]** Kanellakis, P.C., Kuper G.M., and Revesz P. Z.. Constraint query languages. In *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'90)*, pp. 299-313. ACM Press, 1990

**[KKR95]** Kanellakis, P.C., Kuper G.M., and Revesz P. Z.. Constraint query languages. *Journal of Computer and System Sciences*, 51(1), pp26-52, 1995

**[KLP98]** Kuper G., Libkin L., Paredaens J. (editors), Constraint Databases, Springer, 1998

**[Kupe93]** Kuper, G.M. Aggregation in constraint databases. In Vijay Saraswat and Pascal Van Hentenryck, editors, *proceedings of the 1st International Workshop on Principles and Practice of Constraint Programming (PPCP '93)*, pp. 161-172, MIT Press, 1993

**[LaMc92]** Lassez J.L., McAloon K., Applications of a Canonical Form of Generalized Linear Constraints. in *Journal of Symbolic Computation* (1992) 13, pp.1-24, 1992

**[LHM89]** Lassez, J.L., Huynh T., McAloon K. Simplification and elimination of redundant arithmetic constraints. *Proceedings of NACLP 89*, MIT Press, 1989

**[LiWo94]** Libkin L., Wong L., Some properties of query languages for bags. In *Proc. Database Programming Languages*, Springer Verlag, pp. 97-114, 1994

**[Lips95]** Libkin L., Query language primitives for programming with incomplete databases, In *Proceedings of DBPL'95*, 1995.

**[Maki77]** Makinouchi A., A consideration of normal form of non-necessarily-normalized relations in the relational data model. In *Proc. of Intl. Conf. on Very Large Data Bases*, pp. 447-453, 1977

**[Mati70]** Matiyasevich Y., Enumerable Sets are Dispantine. *Dokladu Akademii Nauk SSSR*. vol. 191*,* pp. 128-138, 1970

**[Reit78]** Reiter, R. On closed world databases, *Logic and databases*, Plenum Press, 1978

**[Reit86]** Reiter, R. A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values. *JACM* 33(2): pp. 349-370, 1986

**[Reve93]** Revesz, P.Z. A closed-form evaluation for Datalog queries with integer (gap)-order constraints. *Theoretical Computer Science (TCS)*, 116(1/2), pp. 117-149, 1993

**[Sprin79]** Melvin Dale Springer, The algebra of random variables, *Wiley series in probability and mathematical statistics*, 1979

**[Tars51]** Tarski, A. A Decision Method for Elementary Algebra and Geometry, University of California Press, Berkeley, CA, 1951

**[YuCh88]** Yuan, Li Yan and Chiang, Dian-An, A sound and Complete Query Evaluation Algorithm for Relational Databases with Null Values, ACM 1988