Data Exchange: Algorithm for Computing Maybe Answers for Relational AlgebraQueries

S. M. Masud Karim
Computer Science and Engineering Discipline, Khulna University, Khulna 9208, Bangladesh.
E-mail: masud@cse.ku.ac.bd

Abstract—The concept of certain answers is usually used in the definition of the semantics of answering queries in data exchange. But as certain answers are defined as a relation without null values, the approaches for answering queries over databases do not always lead to semantically correct answers. In order to obtain all possible semantically correct answers, maybe answers have been considered along with certain answers. In this paper, an algorithm is presented that produces semantically correct maybe answers to simple relational algebra queries over target schemas of data exchange under the closed world assumption. As there may be infinitely many maybe answers, sophisticated and restricted representation techniques are used to represent them compactly. Then using the compact representation, an algorithm to compute possible (i.e., maybe) answers incrementally is designed. Finally the algorithm is implemented for a fragment of relational algebra.

Index Terms—close world assumption, data exchange, maybe answers, query answering.

I. INTRODUCTION

Data exchange, also known as data translation is the problem of transforming a given instance of a source database schema to an instance of a target database schema by satisfying a set of constraints and reflecting the given source data as accurately as possible. The accuracy and completeness of data exchange largely depend on the semantics of the data exchange problem and are best verified by answering queries over target instances in such a way that is semantically consistent with the source data. In [8], the concept of maybe answers is coined in the definition of the semantics of answering queries and a finite representation, referred to as fair representation of the maybe answers is defined. But that finite representation fails to address various simple query answering scenarios. This paper redefines the finite representation and presents an algorithm to compute possible answers using the defined representation.

Combining specifications provided in [1, 4, 7, 8, 9], it can be stated that a data exchange setting is a triple $E = (\sigma, \tau, \Sigma)$, where $\sigma$ is the source schema, $\tau$ is the target schema and $\Sigma$ is a set of constraints. The goal of a data exchange problem associated with a data exchange setting is to find a target instance $T$ of a given source instance $S$ such that $S$ and $T$ satisfy all constraints in $\Sigma$ and produce answers to the queries written over $\tau$ in such a way that is semantically consistent with the information in $S$ over $\sigma$. The target instance $T$ is called a solution of the data exchange problem.

One of the most crucial observations of data exchange problem is that a given instance of data exchange problem may have infinitely many solutions; again there may be no solution. Another crucial observation of data exchange problem is the conceptual difficulty associated with the context of query answering. As there may be many solutions for a given instance of data exchange problem, each will produce its own answer for a specific query, that means, there will be as many answers for a specific query as the solutions. Therefore, the question comes into attention that what makes an answer ‘right’?

It was shown in [3, 4] that the canonical universal solution and the core are good for answering conjunctive queries with inequalities. The concept of certain answers, the answers that occur in the intersection of all possible answers is used in the definition of the semantics of answering queries. But as certain answers are defined as a relation without null values, approaches for answering queries over databases lead to semantically incorrect answers. In [8], the concept of both maybe answers (the union of all possible answers) and certain answers are combined at the levels of individual solutions and all solutions.

In [3, 4], Open World Assumption (OWA) is used, while Closed World Assumption (CWA) is considered as the standard assumption for data exchange in [8]. While OWA permits new facts to be added to the databases, CWA does not allow adding new facts to database except those consistent with one of the incomplete tuples in the database. In [5], an alternative approach based on the concept of locally-controlled open world database is introduced. In this approach, portions of a database in the traditional closed world database can be defined as open. This concept is used in [9] as a standard for schema mapping and data exchange.
The purpose of this paper is to present an algorithm that produces all possible semantically correct answers to queries posed against relational schemas under the closed world assumption. This purpose is associated with the following specific objectives: (i) As maybe answers are inherently infinite, a sophisticated and restricted finite representation has been developed in order to provide the user an upper approximation for the possible answers of a query, (ii) an algorithm has been designed to compute possible answers incrementally, and (iii) finally the algorithm has been implemented for a fragment of relational algebra (i.e. a simple set of SQL queries).

II. LITERATURE REVIEW

A. Data Exchange Problem

A data exchange setting is a triple \( E = (\sigma, \tau, \Sigma) \), where \( \sigma \) is the source schema, \( \tau \) is the target schema (it is assumed that there is no common relation names in \( \sigma \) and \( \tau \) ) and \( \Sigma \) is a set of constraints. The set \( \Sigma \) is a combination of two sets, denoted \( \Sigma = \Sigma_s \cup \Sigma_t \), where \( \Sigma_s \) provides the specification of the relationship between the source and the target schemas (i.e., source-to-target dependencies, in short STDs) and \( \Sigma_t \) expresses data dependencies on the target schema. The focus of attention is restricted only to two types of constraints: tuple-generating dependencies (TGDs) and equality-generating dependencies (EGDs). The data exchange setting in consideration is assumed to be associated with a finite set of TGDs as \( \Sigma_s \) (i.e., STDs) and a finite set of EGDs and target TGDs as \( \Sigma_t \). When writing a TGD or EGD, the universal quantifiers are usually omitted.

Definition 2.1 (Data Exchange Problem): A data exchange problem associated with a data exchange setting \( E \) is the function problem: find a target instance \( T \) of a given source instance \( S \) such that \( S \) and \( T \) satisfy all constraints in \( \Sigma \), provided that such a target instance exists.

The target instance \( T \) is called a solution of \( S \) under the data exchange setting \( E \). It is showed in [7] that, if there is no constraint (dependency in the form of EGDs and TGDs) on target schema, then solutions always exist.

B. Database with Incomplete Information

The most common concept used for modeling incomplete information in the context of relational databases is null value. A null value is placed for an attribute of a relation whose value cannot be represented by an ordinary constant. The unknown null value represents that the attributes value is missing or not known. The nonexistent null value represents that value of an attribute in a tuple does not exist. Most of the researchers consider the null values as existent but unknown in the context of data exchange. Assume that \( \text{CONSTANT} \) is the set of all ordinary values (constants). Let \( \text{NULL} \) be an infinite set of values, called marked null values such that \( \text{CONSTANT} \cap \text{NULL} = \emptyset \). A database instance with incomplete information is an instance whose domain is a subset of \( \text{CONSTANT} \cup \text{NULL} \). Usually source instances are complete relational databases, i.e., their domains are subsets of \( \text{CONSTANT} \) and target instances are relational databases with incomplete information.

C. Valuation

A valuation is a partial map \( v: \text{NULL} \rightarrow \text{CONSTANT} \). If \( T \) is an instance with incomplete information and \( v \) is a valuation defined on all the nulls in \( T \), then \( v(T) \) be the instance of the same schema over \( \text{CONSTANT} \) in which every null \( \bot \) present in \( T \) is replaced by \( v(\bot) \). Then a potential infinite object \( \text{REP}(T) \) can be defined as

\[
\text{REP}(T) = \{v(T)\},
\]

where \( v \) is a valuation.

D. Universal Solution

The notation of homomorphism is used to provide the algebraic specifications of universal solution and core.

Definition 2.2 (Homomorphism): For any two instances \( T_1 \) and \( T_2 \) over any arbitrary schema, where domains of instances are subsets of \( \text{CONSTANT} \cup \text{NULL} \), a homomorphism \( h : T_1 \rightarrow T_2 \) is defined in [3] as a mapping function from \( \text{CONSTANT} \cup \text{NULL} \) to \( \text{CONSTANT} \cup \text{NULL} \) such that:

- For every constant \( c \in \text{CONSTANT} \), \( h(c) = c \).
- For every fact \( P(t) \) of \( T_1 \), there is a fact \( P(h(t)) \) in \( T_2 \), where for any \( t = (x_1, x_2, \ldots, x_n) \), \( h(t) \) is defined as \( (h(x_1), h(x_2), \ldots, h(x_n)) \).

This definition of homomorphism also implies mapping from \( \text{NULL} \) to \( \text{CONSTANT} \cup \text{NULL} \). In order to obtain the same complete instances as \( \text{REP}(T) \), this definition of homomorphism is slightly modified in [8] by assuming the mapping from \( \text{NULL} \) only to \( \text{NULL} \) and then a partial valuation is performed.

Definition 2.3 (Universal Solution): A solution \( T \) for a source instance \( S \) is called a universal solution for \( S \), if for every solution \( T' \) for \( S \), there is homomorphism \( h : T \rightarrow T' \).

The canonical universal solution of a given source instance \( S \) is denoted as \( \text{CANSOL}(T) \). The concept of universal solution suffers from the fact that there may be multiple, non-isomorphic universal solutions for a source instance under a given data exchange setting. Therefore, the notion of cores of universal solutions is defined in [4] and it is denoted as \( \text{CORE}(T) \) for solutions \( T \).

E. Data Exchange Solutions under CWA

In [8], CWA is chosen as the standard assumption for data exchange and CWA-solutions are considered as the solutions. The concept of justification is taken into account in [8, 9] for generating the CWA-solutions. This can easily be verified that CWA-solutions are universal solutions in the terminology of [3]. For every CWA-solution \( T \), the following inclusions hold:
\[ \text{Rep} (\text{Core}(T)) \subseteq \text{Rep}(T) \subseteq \text{Rep} (\text{CanSol} (T)). \]

**F. Certain Answers and Maybe Answers**

The **certain answers** of a query \( Q \) are the tuples that occur in the intersection of all \( Q(T) \) on all the solutions \( T \). If the collection of all solutions for \( S \) under the data exchange setting \( E \) is defined as \( \text{SOLUTION}(E, S) \), the certain answers of \( Q \) on \( T \) with respect to \( E \), denoted \( \text{CERTAIN}(Q, S) \), is a set

\[ \text{CERTAIN}(Q, S) = \cap \{ Q(T) \mid T \in \text{SOLUTION}(E, S) \}. \quad (2) \]

On the other hand, the **maybe answers** of a query \( Q \) are the tuples that occur in the union of all answers of the query \( Q(T) \) on all the solutions \( T \) in \( \text{SOLUTION}(E, S) \). The maybe answers of \( Q \) on \( T \) with respect to \( E \), denoted \( \text{MAYBE}(Q, S) \), is a set

\[ \text{MAYBE}(Q, S) = \cup \{ Q(T) \mid T \in \text{SOLUTION}(E, S) \}. \quad (3) \]

To evaluate \( Q \) on an instance \( T \) with nulls, the set \( \{ Q(R) \mid R \in \text{Rep}(T) \} \) is normally considered. The lower and upper approximations are defined respectively as

\[ \nabla Q(T) = \cap \{ Q(R) \mid R \in \text{Rep}(T) \}, \quad (4) \]
\[ \Delta Q(T) = \cup \{ Q(R) \mid R \in \text{Rep}(T) \}. \quad (5) \]

**G. Semantics of Query Answering**

There are primarily two different ways to obtain the answers to queries over different solutions: (i) by computing the certain answers which are true for all solutions (this semantics used in [2]) and (ii) by collecting tuples true in some solutions. The combination of certain and maybe answers at the levels of individual solutions and all solutions give rise to four reasonable semantics for query answering. For a source instance \( S \) under a data exchange setting \( E \), these are defined in [8] as **certain answers semantics**, **potential certain answers semantics**, **persistent maybe answers semantics** and **maybe answers semantics**.

**H. Representation of Maybe Answers**

In [8], a finite representation of maybe answers \( \Delta Q(T) \) is defined using a different valuation, termed as **strict valuation**, which states 1-to-1 mapping from the set of nulls in a tuple of a solution \( T \) to \( \text{CONSTANT} \) such that no value of strict valuation occurs as a constant in \( T \). For a \( T \) and a query \( Q \), a table \( W \) is termed as **fair representation** of \( \Delta Q(T) \), if

\[ \cup \{ \text{Rep} \left[ \{ \bar{t} \} \mid \bar{t} \in W \} = \Delta Q(T). \quad (6) \]

Here, \( \text{Rep} \left[ \{ \bar{t} \} \right] = \{ \nu(\bar{t}) \} \), where \( \nu \) stands for strict valuations.

It is further stated in [8] that if \( \{ a_{ij}, \bot, a_{ij}, \bot \} \) is in a fair representation of \( \Delta Q(T) \), then for every pair \( \{ a_{ij}, a_{ij} \} \) of constants not present in \( T \), the tuple \( \{ a_{ij}, a_{ij}, a_{ij} \} \) is in \( Q(R) \) for some \( R \in \text{Rep}(T) \).

It can be verified that such a simple table \( W \) is not enough to hold all representative information of \( \Delta Q(T) \). Again, by applying valuation with any combination of constants does not always give maybe answers satisfying the predicate.

**III. PROPOSED METHOD**

In query answering scenarios, two extreme semantics: **certain answers semantics**, \( \text{CERTAIN}(Q, S) \) and **maybe answers semantics**, \( \text{MAYBE}(Q, S) \) are used. Since both can be computed over \( \text{CanSol}(S) \) [8], query answering can be done for simple relational queries. Simple positive SQL queries considered for the experiment are of the following general format:

**SELECT** **Attribute-list**
**FROM** **Relation-list**
**WHERE** **Predicate**.

**Relation-list** consists of any \( n \) relations \( R_i \) with \( 1 \leq i \leq N \). **Attribute-list** has one or more of \( A_j(s) \), where \( A_j \) stands for the \( j \)-th attribute from \( R_i \). The value of \( j \) depends on the arity of the associated relation \( R_i \) and it may be different for different relations, i.e., for different values of \( i \). **Predicate**, denoted by \( P \) consists of \( p_1 \land p_2 \land \cdots \land p_m \), i.e., conjunction of \( m \) atomic expressions of the form \( p \colon X \text{ op } Y \), where \( \text{op} \) is any binary operator from the set \( \{ =, <, >, \geq, \leq, \neq \} \). One of the two operands \( X \) and \( Y \) of \( p \colon X \text{ op } Y \) must be an attribute from any \( R_i \). In order to distinguish between projection and predicate attributes, a superscript is used: \( s \) for projection-attributes (attributes in **SELECT** clause) and \( w \) for predicate-attributes (attributes in **WHERE** clause). The other operand may be either any constant value or another attribute from any \( R_i \). When both operands are of the form \( A_j^s \), they might be either from the same relation or two different relations.

**A. Computing Certain Answers**

When a positive relational query \( Q \) is posed over a schema \( \tau \), the certain answers \( \nabla Q(T) \) are computed using the naive evaluation method [6] on the canonical solution \( T \) over \( \tau \). The null variables are treated as constants and general query evaluation is applied to \( T \) in order to get \( Q(T) \). Finally, the tuples with nulls are discarded to get \( \nabla Q(T) \).

**B. Computing Maybe Answers**

The system is initialized with a canonical solution and a pre-processing metadata (the pre-processing is explained later). All the attributes stored in the relational databases are of type **text** (or varchar). The constants are inserted in its original form in texts. Each null is
represented by distinct text with a common prefix pattern:

$n$.

B.1. Pre-processing

In order to reduce the work load during the computation of maybe answers, pre-processing can be applied to a canonical solution $T$. Pre-processing may include identification of null/constant presences for attributes, computation of the common attributes in relations etc. Pre-processing on the attributes uses two Boolean parameters, hasNull and hasConstant, confirming whether a specific attribute has null values and constant values, respectively. Note that, out of the four combinations for the possible values of these two parameters, (TRUE, TRUE) is most common in data exchange. The combination (FALSE, FALSE) is an impossible one and hence is never used. The attributes with no nulls, having parameters value (FALSE, TRUE) and with only nulls with parameters value (TRUE, FALSE) are vital in data exchange.

B.2. Query Rearrangement

As $Q$s are simple relational algebra queries, they can easily be rearranged to obtain better performance in join operation. In order to reduce the storage complexity during the join operations, the projection can be pushed in, i.e., projection is performed before join operation. The concept of restriction will be useful during the query rearrangement process.

**Definition 3.1 (Restriction):** The set of all atomic expressions $p_r$ with $1 \leq r \leq m$ is called the restriction of $P$ to $R$, if all the attributes used in the atomic expressions are only from $R$.

Restriction of $P$ to $R$ is denoted by $\text{REST}_p(R)$. This definition can be extended for a relation-pair.

**Definition 3.2 (Extended Restriction):** The extended restriction of $P$ to a relation-pair $(R, R')$ with $i \neq j$ is the set of all atomic expressions $p_{ij}$ in $P$ that include only attributes of $R \cup R_j$. When a query $Q$ is posed to any $n$ relations of an instance $T$ with $N$ relations $R_1, R_2, \ldots, R_N$,

- First projection operation is performed on each of the $n$ relations using $A_{r1} \cup A_{r2} \cup A_{rk}$, where $A_{ri}$ are the attributes in $R_i$ that also present in at least one of the $(n-1)$ relations (superscript $c$ stands for common attributes in different relations). Note that three different $j, k, l$ are used to avoid ambiguity, but they can also represent the same value(s).
- Then, the restrictions of $P$ to relations and extended restrictions of $P$ to relation-pairs are identified. Finally, each of the $p \in \text{Rest}_p(R)$ is applied to $R_i$ using valuation to compute the tuples incrementally upon which $p \in \text{Rest}_p(R)$ holds.

B.3. Join Operations

The basic idea for the null variables in the join operation as follows:

- Nulls of different attribute domains are different i.e., $\bot_i \neq \bot_j$ for any $i \neq j$ and any $k, l$.
- Nulls of the same attribute domain are same i.e., $\bot_i = \bot_j$ for any $i \neq j$ and any fixed $k$.

Now join operation, referred to as join around nulls, in short $\text{JAN}$ and denoted by $\otimes$ can be described as (i) to perform the Cartesian product on the targeted relations, and (ii) to discard the tuples with different constant values for the common attribute(s).

B.4. Compact Representation

After performing query rearrangement and, then applying join operation and extended restrictions of $P$ to relation-pairs iteratively, a combined relation is obtained. Finally by performing projection operation using projection attributes on the combined relation, an intermediate representation, $W$ (with another table $C$ for indicating the conditions) of maybe answers $\Delta Q(T)$ is obtained. Finally, $\Delta Q(T)$ can be expressed as

$$\cup \{ \text{REP}_{i}(\vec{t}) | \vec{t} \in W \} = \Delta Q(T).$$

Here, $\text{REP}_{i}(\vec{t})$ is obtained using Eq. (1) by satisfying conditions in $C$.

B.5. Putting All Together

First, the query $Q$ is analyzed and decomposed to get $A_1^c, A_2^c, A_k^c$ of all $n$ query-relations.

Next, for each relation $R_i$ of the $n$ query-relations, where $1 \leq i \leq N$, do the following:

- Perform projection operation on $R_i$ using $A_{r1}^c \cup A_{r2}^c \cup A_{rk}^c$, where $j, k, l$ are integers.
- Identify the restriction of $P$ to $R_i$, i.e., $\text{REST}_p(R_i)$ and apply each atomic expression $p \in \text{REST}_p(R_i)$ on $R_i$ using valuation.
- Identify the extended restrictions of $P$ to relation-pairs.

A set of $n$ relations is produced with possibly less arity, where all tuples of a relation are satisfied by corresponding restriction of that relation. Then any two of the $n$ relations are combined into a single relation, $r_c$ by applying $\text{JAN}$ and each of the atomic expression $p$ of the restrictions of $P$ to the relation-pair is checked on $r_c$ using valuation. The conditions in $p$ are stored in a table $C$. Then, $\text{JAN}$ is again applied to $r_c$ and another relation from among $n-2$ relations and extended restrictions of a relation-pair are checked on $r_c$ if the relation-pair is a subset of already combined relations. Finally, table $W$ is obtained by performing projection operation on the combined relation using $\cup \{ A_{rk}^c \}$ of all relations $R_i \in n$ - query-relations. The combined $(W, C)$ provides compact representation.
representation of the maybe answers $\Delta Q(T)$. The table $C$ can only be omitted in special cases. It is claimed in [8] that, a \textit{fair representation} of $\Delta Q(T)$ can be constructed in polynomial time for a positive relational algebra query. Initial results using queries consist of only equality (i.e., $=$) on attributes with nulls and others operators (e.g., $>$, $<$, $\geq$, $\leq$) on attributes without nulls justify this claim.

But this claim fails in case of queries including $>$, $<$, $\geq$, $\leq$ on attributes with nulls. The main reason of this failure is that a simple table is not enough to represent maybe answers in such situations. Hence, the theorem can be modified as follows:

**Theorem 3.1:** If a positive relational query $Q$ is executed on a canonical solution $T$, an \textit{intermediate representation}, $W$ of $\Delta Q(T)$ can be constructed in polynomial time, where $Q$ consists of only equality ($=$) on attributes with nulls and others operators ($>$, $<$, $\geq$, $\leq$) on attributes without nulls. •

In the algorithm, the query rearrangement phase and joining phase just rearrange and organize the query-relations in order to improve the query evaluation. In valuation phase, answers are obtained by assigning constant to null (if only one of the operands of '=' is null) and renaming two nulls to a new null variable (if both operands are null). The main idea behind this operation is that the infinitely many maybe answers form equivalence-classes. The implemented algorithm returns each of these equivalence-classes only once, as nulls are renamed considering equivalent (i.e., using the isomorphic property). Based on the work done in this paper, it can be stated that

**Theorem 3.2:** If a generalized relational query with inequalities $Q$ is executed on a canonical solution $T$, an \textit{intermediate representation}, $(W,C)$ of $\Delta Q(T)$ can be constructed in polynomial time. •

IV. EXPERIMENTAL RESULT

The proposed algorithm is implemented in a restricted setting. It is assumed that all the SQL queries posed to the system are syntactically correct. The operation process of the described algorithm is experimented using different scenarios of data exchange setting. The system is experimented using simple positive queries as well as queries with inequalities. Couples of the scenarios are explained below:

**Example 4.1 (People-Person):** Consider a canonical solution is given in Table I. Table II shows the result of the preprocessing. If the following SQL statement

\[
\text{SELECT person.name, people.name}
\]

FROM person, people
WHERE person.name = people.name;

is executed on the canonical solution, output (in Table III) is produced after applying query rearrangement and join operation using valuation. Note that new null variable (i.e., _n_name_3) is generated by renaming nulls in the last tuple of Table III.

**Example 4.2 (Tabulation):** Consider another simple canonical solution given in Table IV, which shows marks obtained by students in a particular course. The result of the preprocessing is given in Table V. Now “to list the students who have obtained at least 27 in the first test”, the SQL expression, $Q_2$ is written as

\[
\text{SELECT marks.roll, marks.test1 WHERE marks.test1 } \geq 27.
\]

As it is a single relation, query rearrangement only drop attributes of other than the set \{roll, test2\} and no join operation is needed.
It is clear from Table IV that the first tuple has a constant (i.e., 29) which satisfies the predicate and hence it is included in the representation. The second and third tuples have marked null variables (_n_test1_1 and _n_test1_2 respectively) and applying valuation, sometimes constants are mapped which satisfy the predicate (for constant values greater than or equal to 27) and sometimes does not satisfy (for constant values less than 27). A simple marked null variable cannot be used in the representation and for every valuation, correct answer is not obtained.

Using the concept of adding condition to the conditional-table [6], a slightly modified null variable, termed as weighted marked null variable is used instead in \( W \) (see Table VI(a)). A separate table \( C \) given in Table VI(b) is used to add the condition i.e., define the range of the weighted marked null variable present in the representation. In Table C, each tuple contains a condition, where value can be either a constant or an attribute-name. If it is an attribute-name, it means there has to be a comparison between the values of this attribute and the attribute in the ‘attribute’ field. When a weighted marked null variable is encountered in \( W \) during valuation, first the condition is taken from \( C \) for that attribute and then the condition is checked for all weighted marked null variable. If it is satisfied, the tuple is included as a maybe answer.

Again, “to view the students who have obtained anything other than 27 in first test”, the SQL expression \( Q_{\neq 27} \) is written as

\[
\text{SELECT marks.roll, marks.test1}
\from marks
\where marks.test1 \neq 27.
\]

The representation of the maybe answers is given in Table VII. Again, for getting the students who have got different marks in the first and second tests, the SQL query expression \( Q_{\text{test1} \neq \text{test2}} \) is written as

\[
\text{SELECT marks.roll, marks.test1, marks.test2}
\from marks
\where marks.test1 \neq marks.test2,
\]

the representation is obtained as given in Table VIII. Here, ‘test2’ in the value field indicates an operation between the values of ‘test1’ and ‘test2’. As the nulls are different on the last tuple of \( W \) in Table VIII, the valuation will be different. It can be remembered that for an equality (=), a new null is generated by renaming the nulls (see Example 4.1).

**Example 4.3 (Recruitment):** Consider a canonical solution is given in Table IX, where the term ‘cid’ means candidate-id, and the terms ‘exp-salary’ and ‘str-salary’ mean expected-salary and starting-salary respectively. Table X shows the result of the preprocessing. If the following SQL statement

\[
\text{SELECT candidate.cid, candidate.age, candidate.post, candidate.exp-salary}
\from candidate, position
\where candidate.age <= 28
\and position.str-salary >= candidate.exp-salary;
\]

is executed on the canonical solution, representation of the maybe answers is obtained as given in Table XII.
Here the condition candidate.age <= 28 is a restriction to relation candidate. Applying this condition will drop the last tuple from relation candidate. The first tuple is added as a certain tuple, while the inner two are maybe tuples with a condition in table C. The other condition is an extended restriction to the relation-pair {candidate, position}. After applying JAN to relations candidate and position, this condition is applied to the combined relation. This will also add a condition to table C.

V. PERFORMANCE ANALYSIS

Implemented system is evaluated using a combination of goal-based evaluation and IT-system as such [2]. Using a combination of criteria-based evaluation and IT-system as such [2], it is ensured that no invalid criterion is assumed. The results of each phases like query rearrangement, join operation etc are compared with the definitions and each time expected outcomes are obtained. The proposed algorithm generated a finite intermediate representation W of this infinite object (with C), which is defined in Eq. [8].

\[ \bigcup \{ \text{REP}(t) \mid t \in W \} = \Delta Q \left( \text{CAN}\text{SOL}\left( S \right) \right). \] (8)

The experimental results show that the implemented algorithm is complete, produces all the maybe answers. The implementation uses no relational database software-specific macro; hence it can be implemented on any system.

VI. FUTURE WORKS

The restricted implementation setting under CWA is tested with simple positive relational queries and generalized relational queries with inequalities. This can be extended for generalized queries with joins, e.g., queries on self-joined relations. More generalized data exchange setting with target dependencies can also be used to get all possible answers.

VII. CONCLUSION

Maybe answers play a vital role in the study of data exchange. The maybe answers semantics provide an upper approximation for the answers to the queries. In this paper, the algorithm proposed in [11] is modified and implemented to compute the maybe answers incrementally under CWA. The results show that the algorithm generates all possible answers for generalized positive queries as well as queries with inequalities.

ACKNOWLEDGEMENT

I am grateful to Leonid Libkin for his initial concept and guideline. I also thank David Kensche for his helpful suggestions.

REFERENCES