QUERY PROCESSING IN DEDUCTIVE DATABASES WITH INCOMPLETE INFORMATION

By
Tomasz Imielinski[1,2]

Department of Computer Science
Rutgers University
New Brunswick, New Jersey 08903

Abstract

We study here automated deduction in databases in the presence of various types of inference rules of the form of Horn Clauses with Skolem functions These inference rules are typical for databases with incomplete information We demonstrate a number of results related to processing of conjunctive queries for different types of database intensions In particular, we show that when a database intension is built from possibly cyclic inclusion dependencies and view definitions any conjunctive query can be translated to the an equivalent form which can be evaluated directly over the database extension (disregarding inference rules) We also demonstrate that the complexity of query processing significantly grows when we mix incomplete information with recursive rules In particular, we demonstrate here that even the power of least fixpoint extension of first order logic may be not sufficient to process queries in the presence of incomplete data and recursive rules The same is demonstrated in case disjunctive information is allowed in the database

1 Introduction

From the point of view of logic the database state can be viewed as a collection of formulas belonging to two classes - database *extension* and database *intension* The database intension corresponds to time invariant properties of database and usually consists of integrity constraints and view definitions The database extension reflects the current state of knowledge about the world and is a subject of frequent updates The database extension and database intension may be linked in two different ways In the classic approach based on *Closed World Assumption* [11]

the database extension is viewed as a model and has to satisfy the formulas of database intension It is easy to see that in this case the database intension is irrelevant from the point of view of query processing - we will say it is *passive* In the second approach which is related to the *Open World Assumption* we only require the database extension to be consistent with database intension This point of view is frequently adopted in deductive databases In this approach the database intension becomes *active*, relevant to the query processing and the formulas of the database intension are frequently called *inference rules* (On the contrary to integrity constraints in the previous case)

In this paper we are going to adopt the second point of view and will be interested in the query processing under these circumstances This area is currently a subject of vigorous research ([4]) However most of the papers deal with the case of complete information and database intensions in the form of Horn clauses without functions We are not aware of any study of query processing in deductive databases with incomplete information possibly involving *Horn Clauses with Skolem functions* as inference rules Such situations occur frequently in practice for example when

1 When Inclusion Dependencies become inference rules

2 When updates on views are allowed even when translations of these updates are not currently known

3 In general, when incomplete data is allowed to enter the database

We will demonstrate that all of these situations lead to the occurrence of Horn clauses with Skolem functions in the database intensions and in consequence to the incomplete information in the database

[1]Research supported by NSF grant DCR 85 04140

[2]On leave from Institute of Computer Science, Polish Academy of Sciences, Warsaw 00-901, Poland

In this paper we will investigate the query processing in such incomplete information deductive databases In particular we will be interested in the complexity of query processing in this circumstances What happens to the complexity of query processing when updates on certain families of views are allowed?

How the difficulty of query processing grows when we allow recursive Horn clauses with Skolem functions or Non Horn rules representing disjunctive information ? These are the typical questions we will be investigating here in this paper

Example 1 [Data Dependencies as Inference Rules]

Let our database scheme consists of the following relations

R[Student, Teacher, Class, Department]

S[Student, Teacher, Project]

U[Project, Lab, Student]

W[Project, Teacher, Budget]

P[Teacher, Class, Room]

Where R has the meaning "Student attends the class given by the teacher in the department", S means that a student is taking a project with a teacher, U - Project is beeing done in in the lab by a student, W- project is offered by a teacher with some budget and finally P- a teacher is giving a class in the room

Let the inclusion dependencies have the form

S[Student, Teacher]$\subset$R[Student, Teacher]

P[Teacher, Class]$\subset$R[Teacher, Class]

U[Student, Project]$\subset$S[Student, Project]

W[Project, Teacher]$\subset$S[Project, Teacher]

Notice now how, in our interpretation inclusion dependencies become relevant in query processing Suppose for example that an update to relation S has been made and a new tuple (s,t,p) has been inserted to S Since the corresponding inclusion dependency does not have to be satisfied we do not necessarily have the pair (s,t) represented in the table R If the query asks about the projection of the predicate R on the attributes "Student" and "Teacher" then this tuple (s,t) from S has to be included in the answer for this query This can be done only by application of the inclusion dependency between S and R as the inference rule

Example 2 [Updates on Views]

Let us assume now that our database scheme has a form

R[Supplier, Project, City] , W[Department, Project,], E[Employee, Department]

Additionally let us assume that there are four views defined in the following way

$V_1 = \pi_{Supplier,Project}(R)$

$V_2 = \pi_{Project,\ City}(R)$

$V_3 = \pi_{Department,\ Supplier}(R \bowtie W)$

$V_4 = \pi_{Employee,\ Project}(W \bowtie E)$

These view definitions correspond to the formulas defined by means of existential quantifier and conjunction For example the definition of the view V_3 corresponds to the following formula

$V_3(x,y) \Leftrightarrow R(y,z,w) \wedge W(x,z)$

In this way each view can be represented as the formula defining a new predicate This formula is going to be a part of the database intension The requirement that the database extension satisfy this view definitions all the time (i e they become pasive) requires that in fact all information entering the database is complete (i e all translations are known) It seems to be a pretty unrealistic requirement We may replace it by requiring only that the database extension be *consistent* with view definitions meaning that we may accept insertions of new facts into the view predicates (so called updates on views) without necessarily knowing the "translations" ([2]) of them on the level of base predicates From the point of view of logic view updates are simply updates which are nonatomic formulas possibly leading to incomplete data For example we may enter a new "fact" in the form of the tuple <Smith, Databases> into the predicate V_4 meaning that we know that Smith is associated with a Database project even if at the given moment we do not know in which department Smith works This means that we can accept the database extensions which *do not satisfy* the database intension formulas (in this case view definitions) although they are still consistent with them (Indeed, in case of our update the definition of the view V_4 will not be

satisfied since we do not have a corresponding department in the relation W) Notice therefore that making a view definition or inference rule active is simply equivalent to making this view "updatable" while making it passive is equivalent to rejection of updates on this view (unless proper translation is known)

View definitions together with a remaining part of database intension will influence the process of query answering For example if in our database we have the query "Give me all employees who work for *some* departments" - the answer should include Smith although he is not necessarily represented in the table E This information will be derived from database intension which plays the role of *inference rules* rather than integrity constraints in this case

1 1 Query Processing

We will be interested in extending here the two phase approach for query processing introduced by R Reiter in [12] Given a query Q and the database intension Σ we will be interested in translating Q to some other form Q^{Σ} which can be evaluated directly on the database extension disregarding the formulas of database intension As stated in [12] such approach has a number of advantages, the chief one beeing the avoidance of costly backtracking in the second phase of query processing (which is done by the relational algebra since the database extension is relational) Besides, the size of the translation Q^{Σ} is a good measure of complexity of query evaluation Finally, since the extension of the database is a subject of frequent updates , while intension is not, the isolation of"the invariant portion of query evaluation" in a form of Q^{Σ} is helpful when Q is a view definition

A natural question is whether Q^{Σ} is first order definable and if not whether it is least fixpoint definable?

If it is first order definable, then the size of Q^{Σ} does not depend on the size of database, i e the number of join (or union) operations to be performed over the database state in the second phase is not a function of the size of database extension On the other hand, if Q^{Σ} is not first order definable then it is at best "recursive" i e similar to the least fixed point queries which require iterative computation of the same expression over the database a number of times which in general is of the order of database size We will also demonstrate results demonstrating when even such recursive translation is not possible This expressibility questions have clear complexity implications Both data and expression complexity of least fixpoint queries and first order queries are well understood (14]) They give us a preliminary indication as how "hard " the query becomes because of the influence of database intension In this way complexity analysis can be extended for nonatomic, logical databases

In this paper we will focus on the expressibility questions of the conjunctive queries in presence of various families of data dependencies and view definitions

Before proceeding further let us give a few examples of situations in which different types of inference rules arise

Example 3 (Horn Clauses without Skolem functions)

If the set of Horn clauses is nonrecursive we obtain a situation which was previously considered by Reiter This is a very simple case when for any conjunctive query Q it's translation will be a union of expressions built from projections and joins If the set of rules is recursive then no such a first order definable translation exist in general and the corresponding expression is a so called least fixpoint expression [1]

Example 4

In the database from example 1 the query

$Q = \pi_S(R)$

asking about all students who take some classes will be translated to the form

$Q^{\Sigma} = \pi_S(R) \cup \pi_S(S) \cup \pi_S(W)$

Indeed, since the inclusion dependencies play the role of inference rules here we may enter some students into the tables S and W without necessarily introducing them into the table R The membership of these students in the answer for the query is a consequence of application of inference rules (which are inclusion dependencies in this case)

Similarly, in the second example (View Definitions) let our query be

$Q = \pi_{\text{Employee, Supplier}}(R \bowtie W \bowtie E)$

It is easy to see that the required translation has the form

$Q^{\Sigma} = Q \cup \pi_{\text{Employee, Supplier}}(V_4 \bowtie R)$

Indeed tuples from the extension of the view V_4 must be taken under consideration in the processing of the query Despite the fact that we may not have proper translations of these updates on the basic scheme level they influence the answer for the query (It does not matter for the query what is the department of the given employee so, intuitively, it does not matter when it is unknown)

Notice that in both cases above the translations of the queries are first order definable and can be evaluated directly over the extensions of the database predicates disregarding totally the database intensions (in the first case inclusion dependencies, in the second case view definitions)

■

1 2 Our Approach - the alternative treatment for processing incomplete information and updates on views

The two phase translation approach has an important advantage over the standard way of representing incomplete information in databases by means of null values It is reasonable to assume that the only way the incomplete information may enter the database system is through the updates on views Under this assumption, instead of introducing null values on the level of database extension it is much easier to introduce the proper view definition formulas (since they are predefined by the database administrator) as inference rules of database intension and keep the database extension as the ordinary relational database (the database extension will consist now the database predicates and view predicates)[3]) In the process of query answering we have only to produce proper translation of the query (as in the above examples) and evaluate it as the relational algebra expression over the database extension in the well known way It is simple and natural on the contrary to space consuming ways of storing the tuples with null values and necessity of extending relational algebra operations to work on the tables with null entries allowed ([6])

Our approach provides also an alternative treatment of updates on views Instead of looking for the complete translations of updates on views we simply accept them as incomplete data and incorporate this information into the query in the translation process Let us point out here the important role the only if parts of view definitions play in this process Notice for example the role of the 'only if" part of the definition of the view V_3 in the example 2 If a new tuple is inserted into the predicate V_3 then the "only if" part of V_3 definition provides an "incomplete" translation of this update

The paper is organized as follows In the second section we introduce the basic notions, in the third section we study processing of conjunctive queries when the database intension is built from so called pseudorecursive Horn clauses which include inclusion dependencies and major types of view definitions We demonstrate the proper translation procedures for these cases We also present negative results related to complexity of query processing in the presence of inclusion dependencies and template dependencies Finally, in the last section we deal with the problem of translatability of conjunctive queries when the database intension is built from non-Horn clauses

2 Basic Notions

We will assume that the reader is familiar with basic notions of relational database theory [13] We will alternatively use here a notion of relational algebra and relational calculus We will assume that the database state is a collection of positive atomic formulas (tuples in relations)[4] which form the database extension and nonatomic closed formulas forming the database intension - set of integrity constraints and view definitions Any logical theory can be represented in this way since we can always introduce additional predicates to the language as well as corresponding definitions (formulas) That is for any nonatomic formula F(a) we may introduce additional predicate V and add V(a) to the database extension with the simultaneous addition of the axiom (definition) $V(x) \Leftrightarrow F(x)$ to the database intension In the context of databases it pays to have such a transformation since we usually have a large number of instantiations of the same open formula The formulas of database intension usually come from two classes

- data dependencies
- view definitions

Data dependencies are essentially Horn clauses (implicational dependencies) or Horn clauses of the predicate calculus with equality, called functional dependencies All Horn clauses in database intensions

[3] As we show in the next section it is always possible to compile the database to such a form that the database extension will be atomic (relational

[4] no negative facts

will have exactly one positive literal (i e we do not consider clauses with no positive literals)

By database intension we mean a collection of formulas called database dependencies A dependency ([13]) is a first order sentence

$\forall x_1 \ldots x_m \ (A_1 \wedge \ldots \wedge A_n) \Rightarrow \exists y_1 \ldots \exists y_r \ (B_1 \wedge \ldots \wedge B_s)$ where each A_i is atomic formula of the form $Pz_1 \ldots z_d$ (where P is the name of a d-ary relation and where the z_i's are individual variables If each of the formulas B_i on the right hand side is atomic formula then we call the dependency a tuple generating dependency if all of these formulas are equalities then we call the dependency an equality-generating dependency

inclusion dependencies are another important subclass of data dependencies denoted by $R[X] \subset S[Y]$ where R and S are relation names and X and Y are attribute sets This statement represents the relation

$$\pi_X(R) \subset \pi_Y(S)$$

It is easy to see that inclusion dependencies are in fact data dependencies For example $R[A] \subset S[B]$, where R and S are binary relations over attributes C and A and B and D respectively in the above notation can be expressed as

$$\forall_{x,y} \exists_z (r(xy) \rightarrow s(yz))$$

or after skolemization as the following Horn clause with skolem functions

$$r(x,y) \rightarrow s(y,f(x,y))$$

where f is a skolem function

By template dependencies we mean data dependencies without existential quantifiers and equality predicates Template dependencies form a generalization of multivalued and join dependencies [13] The database intension language denoted by L_i describes which formulas are allowed as part of intension

View definitions have the form of logical equivalences of a view predicate and the formula built from existential quantifiers and conjunctions They can be represented alternatively as a collection of Horn clauses with Skolem functions (corresponding to existential quantifier) By n-Horn clauses we mean clauses with n literals For example inclusion dependencies are 2-Horn clauses (with Skolem functions)

In the paper we will adopt the following convention predicates will be denoted by small letters like p q, w, v and their corresponding extensions (relations in the database which correspond to all instantiations satisfying the predicate in the current state of the world) by corresponding capital letters P, Q, W, V We also will drop parenthesis after predicates and commas between variable names For example we will write $px_1x_2x_3$ instead of $p(x_1, x_2, x_3)$ The same convention will be applied to function symbols

We will always assume that a database extension (denoted by DB) is consistent with a database intension This is a weaker requirement than satisfaction and it follows from Open World Assumption assumed here Formulas of intension simply play a role of (may be incomplete) inference rules For example, inclusion dependencies in our approach do not have to be satisfied by a database extension, a database state can be always "closed" under inclusion dependencies deriving often incomplete data which can be added to a database extension by means of tuples with null values

We assume also that a reader is familiar with the resolution [9]

Our queries will come from sublanguages of predicate calculus, e g sublanguage of conjunctive queries (built only from existential quantifier, conjunction and limited form of selection)

Given a set Σ constituting the database intension, and a query Q we will be interested in *translation* of Q into Q^Σ such that for any extension DB

$$DB \cup \Sigma \models Q(x) \text{ iff } DB \models Q^\Sigma(x)$$

If Q^Σ is first order formula we will talk about first order translatability of Q under Σ, if $Q^\Sigma$is a least fixpoint query ([1], [3]) then we will talk about least fixpoint translatability

Given a query language L_q and a database intension language L_i we will say that L_q is first order translatable modulo L_i iff

For any finite set $\Sigma \subset L_i$ and for any query $Q \in L_q$ there exists a (first order) translation Q^Σ

In an analogous way we could talk about least fixed point definability depending of course on the query language under consideration We may consider two different languages. ***pure*** **least fixpoint queries** - which are extension **of** relational **algebra** simply by

least fixpoint expressions of the form $lfp_g(R)$ = the smallest fixpoint of g containing R which cannot be the arguments of any other expressions and *full* least fixpoint extension in which least fixpoint expressions can be arguments for the other relational operators In this paper we will use least fixpoint definability in the sense of *pure* least fixpoint extension defined above

3 Horn Clauses with Skolem Functions and Conjunctive Queries

As we have said before, situations when database intension is a collection of Horn Clauses with Skolem functions are frequent in practice They arise in the context of inclusion dependencies (which are 2-Horn clauses with skolem functions), even more importantly they also arise when we allow database updates through views defined by Projection and Join Therefore, they basically capture all types of incomplete data we would like to represent

Let us now more precisely define the formulas we are going to deal with

3 1 PJ-rules and pseudorecursive Horn clauses

We are going to define here the class of formulas which comprise both inclusion dependencies and major class of view definitions defined by means of projection and join

Definition

A formula is called a Projection-Join Rule (shortly - PJ-rule) iff it is of the form

$$\forall(x_1 \ldots x_n)\exists(z_1 \ldots z_l)\ R(x_4 \ldots x_m) \Rightarrow (P_1(y_{1,1}, \ldots y_{1,k_1}) \wedge \ldots P_m(y_{m,1}, \ldots y_{m,k_m}))$$

where $y_{i,j}$ is either one of x-variables or one of z-variables In other words the variables on the right hand side of implication either occur on the left hand side of the implication and therefore are universally quantified or are existentially quantified The important restriction is that variables on the left hand side of the implication are universally quantified PJ-rules can be viewed as these data dependencies which have only one occurrence of the literal on the left hand side of the implication ■

It is easy to see that PJ-rules are generalizations of inclusion dependencies Indeed, we obtain inclusion dependencies when we simply limit ourselves to one occurrence of the predicate on the right hand side (i e m=1))

It is easy to see that any set of PJ-rules can be transformed to a set of 2-Horn clauses with Skolem functions Moreover such a set have important property

Proposition 1

Let the set of 2-Horn clauses Σ correspond to the set of PJ-rules If any Skolem function occurs in two clauses c_1 and c_2 then they must share the same negative literal

Proof

The only way two clauses may share the same Skolem functions is when they correspond to the same PJ-rule (different "parts" of it) ■

More importantly PJ-rules correspond to the "Only if" parts of view definitions using only the existential quantifier and conjunction (or projection and join) This is why they are called PJ-rules (Projection-join rules)

Therefore the PJ-rules capture the inclusion dependencies and the "only if" parts of major view definitions This will turn out to be a particularly important property - since the "only if" parts of view definitions play the major role in the updates on views

In this section we are interested in the database intensions corresponding to nonrecursive view definitions and possibly recursive (cyclic) families of inclusion dependencies

Definition

The set of Horn Clauses (possibly with Skolem functions) is called pseudorecursive iff in its connection graph there are no cycles in which the n-Horn clauses with $n \geq 3$ would participate

In other words pseudorecursive set of formulas allows recursion (or cycles) but only on 2-Horn clauses For example cyclic inclusion dependencies form a perfect example of a pseudorecursive set It is also easy to see that any set of inclusion dependencies and nonrecursive view definitions is a pseudorecursive set (Only cycles are between inclusion dependencies) In fact we can transform such a database intension to the form which will contain only PJ-rules (inclusion dependencies and the "only if" parts of view definitions) and Horn clauses without functions (the "if" parts of view definitions)

We are going to demonstrate now the first of the

main results of the paper We demonstrate that even when PJ-rule part of database intension is "recursive" the conjunctive queries have first order definable translations The above theorem will be followed by the translation procedure for conjunctive queries modulo the pseudorecursive formulas

Theorem 1

Conjunctive queries are first order translatable modulo pseudorecursive rules

Before proving the theorem (which will be constructive since it will also show a procedure of generation of a translation of a query) we will demonstrate a translation procedure in case when breadth first resolution tree for $\neg Q$ and the database intension (no database extension) is finite What we have to do in order to compute a translation is to keep resolving a negated query predicate with clauses of a database intension until we eventually reach a finite failure (only clauses from the database intension will participate) Generally speaking some of clauses obtained in this way will correspond to subqueries of Q^{Σ} (the translation Q^{Σ} will be a union of the relational expressions corresponding to these subqueries) The special nodes of the resolution tree which correspond to relational algebra expressions (subqueries of Q^{Σ} are called Relational nodes and are characterized as follows

1 They consists of only negative literals

2 The corresponding substitution (the composition of most general unifiers generated in the process of generation of this node) does not substitute Skolem functions (or constants) for free variables in the query

3 There are no Skolem functions in the clause

Intentionally, the relational node corresponds to the nonempty subquery of the translation Q^{Σ} Indeed, a relational node may be successfully resolved further directly with atomic formulas of database extension If any of the literals in the clause would be positive no such successful resolution could be performed since there are no negative literals in the database extension (this is our assumption in this paper) The requirement about the lack of Skolem functions substitutions comes from the fact that we do not want Skolem functions in the answer for the query (i e when they are substituted for free variables of the query) Finally, we do not want Skolem functions to occur in the relational nodes because they cannot be resolved further with the constants in the database Again therefore (as in case of positive literal in the clause) a clause with Skolem functions could not be resolved with atomic formulas in the database and therefore would not contribute to the subquery of Q^{Σ} It does not mean of course that such a clause cannot contribute in the process of resolution since it may simply be further resolved and lead to a relational node It only means that it cannot be a "terminal" clause which can be resolved directly with the formulas of database extension

Relational clauses correspond naturally to relational algebra expressions in the following way

Given a relational node n satisfying the above criteria and given the set of variables V(c) of the clause c of n which are substituted for free variables of Q we have the following relational algebra expression corresponding to n

$$E(n) = \pi_{V(c)}(E(c))$$

where E(c) is a relational algebra expression corresponding to clause c This can be obtained by first negating c and then generating a relational algebra expression corresponding to $\sim c$ in the well known way (see [13])

Finally, the translation of the query Q, Q^{Σ} will be obtained as the union of relational algebra expressions corresponding to the relational nodes

Now we can come back to the proof of our theorem

Proof of the theorem 1

We will prove out theorem only for the case when the database intension is built purely from PJ-rules The generalization for the arbitrary pseudorecursive set of rules is immediate - indeed it is sufficient to consider additionally nonrecursive Horn clauses without functions

We will construct a resolution tree as above, the only problem is that it may be and in general it will be infinite (if there are cyclic inclusion dependencies) We will demonstrate however that only finite part of this tree is relevant from the point of view of each individual query This will be done by demonstrating a proper "termination" condition which will specify when no further extension of a resolution tree will lead to a generation of new relational nodes

We will deal with three types of nodes Relational nodes, Nonrelational nodes with Skolem functions and Nonrelational nodes with Skolem substitutions for free

variables in the query (as it is easy to see there will be no nodes with positive literals besides original clauses of database intension) First of all it is easy to see that we may simply disregard the nodes of the last category Indeed neither them nor any nodes (clauses) which can be generated from them will be relational nodes since once a Skolem function substitution has been made for a free variable of the query Q we have no chance to reverse it Hence, the nodes of the last category can be simply pruned

The bigger problem is due to the second category of nodes - nodes with Skolem functions occurring in them Although such nodes are not relational they may contribute to the future generation (through resolution) of relational nodes Therefore, they cannot simply be pruned, at least before deciding whether such relational nodes may be produced in the subtree rooted in such a node The relevant question is therefore whether we can perform such a test of "usefulness" of such node in some bounded time The following lemma indicates that this is in fact the case

Lemma 1 Given a resolution tree of the conjunctive query and set of clauses corresponding to PJ-rules It is decidable whether a nonrelational node with Skolem functions is a root of a resolution subtree containing relational nodes

Proof

Without loss of generality let us assume that a clause corresponding to the negation of the query is built from one connected component (each clause can be viewed as a graph in which each node corresponds to an individual literal of the clause and two literals are connected directly if they share a common variable) We will partition variables in the query clause into free and bounded according to the way it is stated in the query The only way Skolem functions may occur in the clause is when as the result of resolution of the original query clause with one of the clauses of the original PJ-rule intension a Skolem function is substituted for a bounded variable x in $\neg Q$ Let n be a node resulting in such resolution step Let p be a predicate name imported to the clause of this node by this resolution step Let l_1 l_k be all the other literals in the clause of this node containing occurrences of variable x In the result of above resolution step each of these literals will contain an occurrence of a Skolem term Each of l_i , i=1, k must be now resolved upon, since we want to get rid of Skolem functions In each case it must be the same predicate p as before which will be imported to the clause as result of resolution This directly follows from the proposition 1 In our case all of the literals l_1 l_k contain the same Skolem terms as the one which was introduced In case we cannot find proper clauses which would have the same predicate p as the negative predicate we fail and the node n can be pruned because no chance of generating a relational node from it exists Otherwise, the next resolution steps are made which possibly may result in the introduction of new Skolem terms into the other literals In all cases however the same predicate p will be imported and all introduced Skolem terms will depend on variables occurring in the imported predicate p If the original query consists of n literals, then after at most n-1 resolution steps we will either get rid of all Skolem functions introduced in the consequence of the first resolution step or we will fail because of the negative occur check Indeed, this would happen when the attempt to substitute a Skolem term in the imported predicate were made Since all Skolem terms depend on all variables in p such an attempt would be equivalent to substituting for a variable a term containing an occurrence of this variable Therefore our procedure will either fail or succeed in at most n-1 resolutions It may succeed earlier if we get rid of all Skolem functions earlier In the positive case - a proper relational node will be generated In the negative case our original node will be pruned In case of several connected components of Q we have to consider each of the connected components separately (by trying to get rid of Skolem terms component after component) ■

Now, let us discuss relational nodes

We will say that two relational nodes $n_1 = <c_1,\Theta_1>$ and $n_2 = <c_2,\Theta_2>$ are equivalent iff there is an isomorphism i from the set of variables occurring in c_1 to a set of variables of c_2 such that

$$c_2 = ic_1$$

$$\Theta_2 = i\Theta_1$$

where i is naturally extended over clauses

It is easy to see that two relational nodes n_1 and n_2 are equivalent iff they correspond to the same subquery of Q^{Σ}

We now have to notice an important property of a resolution tree for 2-Horn clauses, namely that the size of each clause in this tree is bounded by the number of literals in Q It follows easily from the examination of resolution

Therefore, there is only a finite number of possible nonequivalent relational nodes in the tree which can be generated

Moreover the node equivalence relation is preserved by a resolution step i e

$n_1 \equiv n_2 \Rightarrow r(n_1) \equiv r(n_2)$ where

r is a resolution step (by resolving a clause $c(n_1)$ with any other clause) From this we immediately conclude that if a node is generated which is equivalent to a node which has been already produced this node need not have to be further expanded (by resolving it further) since no new queries are going to be generated This can be called a *quasi finite failure* and it guarantees a termination for generation of relational nodes This together with our lemma 1 guarantee that after finite number of resolution steps a query Q^{Σ} can be generated proving our theorem The expression Q^{Σ} is formed as the union of relational expressions corresponding to relational nodes of the tree

■

Notice that the size of Q^{Σ} depends both on the number of literals in Q and on the structure of Σ (However it does not depend on the size of the database extension) In general the length of the query Q^{Σ} will depend exponentially on the length of the original query Q In the worst case the length of Q^{Σ} will also be the exponential function of the size of the database intension (but not of the database extension) Notice also that, as we pointed out before, introduction of nonrecursive Horn clauses does not change the "finitness" property of our resolution tree, hence the theorem easily follows if we add arbitrary nonrecursive set of formulas to the set of PJ-rules

We will now sketch a procedure for generating Q^{Σ} which would include elimination of redundant subqueries

Procedure for Generating Q^{Σ}

1 Build the Breadth first SLD tree by resolving ~Q with the elements of the set Σ

2 Generate a new clause by resolving the goal (query predicate) with the clause from Σ

3

a If the generated node is relational then check if the node - <clause, Θ substitution> is equivalent to a node already generated If so terminate this path and backtrack to another nonequivalent node Check whether a new relational node is not subsumed by some other node, if it is do not expand this node by building the corresponding relational algebra expression Otherwise apply resolution again (if possible) to this new node and repeat the above procedure

b If the generated node is not relational and a corresponding substitution is substituting a Skolem constant for a free variable of the query - prune this node Otherwise, if the node consist of occurrences of Skolem constants perform a test as the described by lemma 1 If the test is negative prune this node, otherwise generate a proper relational node and go to (a)

Our procedure obviously terminates by Theorem 1 since there is only a finite number of nonequivalent relational nodes which can be generated

Remark

This is perhaps a good moment to interpret our result in some wider context Regardless of whether the expression Q^{Σ} is actually going to be constructed explicitly or if we simply run a resolution theorem prover to the end (i e representing database extension in the form of atomic formulas) our termination conditions are useful In the latter case, when we do not construct Q^{Σ} explicitly, they simply become termination conditions for resolution

Finally let us point out that Klug and Johnson ([8]) were considering interaction of conjunctive queries and inclusion dependencies They were however interested in the different problem, namely in the *optimization* of conjunctive queries modulo inclusion dependencies while we are interested in the *translation* of conjunctive queries modulo database intensions in such a way that the database intension may be disregarded We also consider here a wider family of formulas (possibly cyclic PJ-rules and nonrecursive Horn clauses)

Example 7

Let our database intension have a form Σ = ID $\cup$ DEF, where ID is a set of inclusion dependencies and DEF is a set of view definitions

Let a database scheme have a form P[ABC], S[DEF], W[GHI], R[KL] and let inclusion dependencies be

$P[BC] \subseteq S[DE]$

S[EF] $\subseteq$ W[GH]
W[HI] $\subseteq$ P[AB]
S[D] $\subseteq$ R[K]

Notice that the set of inclusion dependencies is cyclic

Finally, let DEF part consists of the following view definition

(*) $v[x,y,z] \leftrightarrow \exists_w \exists_u s(x,y,u) \wedge s(u,w,z)$

Let our query Q be

$Q(x,y) = \{x,y|\exists_u S(x,y,u)\}$

Since the predicate v(x,y,z) does not occur in Q we can disregard the if-part of (*) We will now resolve the query predicate with the individual clauses of the database which have the following form (starting from the inclusion dependencies)

(1) $\{\sim p(xyz), s(yz,fxyz)\}$

(2) $\{\sim s(x_1y_1z_1), w(y_1z_1,gx_1y_1z_1)\}$

(3) $\{\sim w(x_2y_2z_2), p(y_2z_2,hx_2y_2z_2)\}$

(4) $\{\sim s(x_3y_3z_3), r(x_3,kx_3y_3z_3)\}$

(5) $\{\sim v(x_4y_4z_4), s(x_4y_4,lx_4y_4z_4)\}$

(6) $\{\sim v(x_5y_5z_5), s(lx_5y_5z_5, mx_5y_5z_5,z_6)\}$

where the last two clauses correspond to the PJ-rule which is the "only if" part of the view definition (Notice that a skolem function l occurs in both (5) and (6)) We skipped the parenthesis when presenting function symbols

The negation of the query has the following form

(7) $\{\sim s(\underline{x'y'}u)\}$

where the underlined variables correspond to the free variable in the query

We can start our resolution now by resolving (7) with (1) to get the clause

(8) $\{\sim p(xyz)\}$

with the substitution $\Theta=\{ x' \leftarrow y, y' \leftarrow z\ u \leftarrow fxyz\}$ corresponding to the node generated by this resolution step Notice that resolution of (8) with (3) will not be useful (also is possible) since in effect it will map a skolem function $(hx_2y_2z_2)$ to the free variable of the query (y') (through the substitution to z) The only other resolution which we can perform is (7) with (5) (the resolution of (7) with (6) is possible but does not lead to the relational node from the same reasons as above) Therefore we obtain the node (9) which has the form

(9) $\{\sim v(x_4y_4z_4)\}$ and the corresponding substitution has the form $\Theta_1 = \{x' \leftarrow x_4, y' \leftarrow y_4, u \leftarrow lx_4y_4z_4\}$

In this way we obtain the finite failure (or quasi finite failure as we called before No more "useful" resolutions with the formulas of database intension can be performed What is left now is to take the relational nodes of the tree and transform them to the relational algebra expressions

The corresponding expression Q^Σ is a union of expressions corresponding to individual nodes of the above tree

$$Q^\Sigma = \pi_{1,2}(S) \cup \pi_{2,3}(P) \cup \pi_{1,2}(V)$$

In a similar way we can get that for a query Q

$Q(x_1,x_2, x_5) = \{x_1,x_2,x_5|\exists_{x_6x_3x_4} S(x_1,x_2,x_3) \wedge S(x_3,x_4,x_5) \wedge R(x_3,x_6)\}$

$$Q^\Sigma = \pi_{1,2,6}\, \sigma_{3=4}(S \bowtie S) \cup V$$

Notice that all clauses in our proof are 2-Horn ∎ The natural question which arises now is whether we can do anything about query answering in the general context of n-Horn clauses with Skolem functions The answer for this question is important if we want to consider the general deductive databases with incomplete information with possibly recursive view definitions (so far we have only restricted ourselves to the nonrecursive views) We already know that in general, even without Skolem functions the translations of Q will not be first order definable but rather be least fixpoint queries (see e g [1] It turns out that introduction of Skolem constants will make even least fixpoint definability impossible

Theorem 2

Projection queries (P-queries) are not least fixpoint translatable modulo arbitrary Horn clauses with Skolem functions

In other words there exists a family Σ of 3-Horn

Clauses (which is not pseudorecursive) with Skolem functions and a query Q such that Q^{Σ} is not definable even by least fixpoint queries

Proof

In our proof we will use the result of Hull [5] about nonspecifiability of projections of template dependencies Hull in [5] constructs a family Σ of three template dependencies and demonstrates that there is no finite family Σ' of template dependencies such that

$$\pi_x\ (Sat(\Sigma)) = Sat(\Sigma')$$

where $X = \{A,B,C,D\}$ is a subset of the set of attributes of the relational scheme p[A,B,C,D,E] Here is his set of rules which we will denote by Σ

Example 8 [5]

Let the rules have the following form

$$r_1 = P(a,b',c',d',e),P(a,b,c,d,e')\rightarrow P(a,b,c,d,e)$$

$$r_2 = P(a',b,c',d',e),P(a,b,c,d,e')\rightarrow P(a,b,c,d,e)$$

$$r_3 = P(a',b',c,d,e),P(a,b,c,d',e)\rightarrow P(a,b,c,d,e)$$

where a,b,c,d,e,a',b',c',d',e' are all variables Let us consider the projection π_{ABCD} Hull demonstrated that the set of all relations which are the projections on ABCD of the relations satisfying this set of rules is not finitely specifiable by a set of Horn clauses, that is

$$\pi_X\ (Sat(\Sigma)) =\ Sat(\Sigma')$$

for no Σ' beeing a set of full template dependencies

and

$$\pi_X(Sat(\Sigma)=\{\pi_X(R)\ \ R\in Sat(\Sigma)\}$$

Let now $\Sigma^* = \Sigma\cup\{u(x,y,z,v)\rightarrow p(x,y,z,v,w)\}$ The latter formula is an "Only If" part of the definition of the view u which is defined in terms of projection and can be independently updated We will demonstrate that the query $Q = \pi_{1,2,3,4}(P)$ does not have a least fixpoint definable translation

First of all let us notice that for any relation q over the set of attributes X there exists a smallest q^* such that

(i) $q^*\supseteq q$

(ii) $q^*\in\pi_X(Sat(\Sigma))$

Let our current database state has a form $<U,\emptyset>$, where U is an extension of u and extension of P is empty Such a database state may be a result of the "massive" view update of the predicate u, while the extension of the "basic" database predicate is empty Such a database state obviously does not satisfy the formulas of the database intension (the view definition), but is still consistent with them (which is what we require here) The answer for our query in this case will have a form U^*, where U^* is the smallest relation as define by (i) and (ii) Indeed, formally the answer for our query can be expressed as $\cap\{\pi_X(P)\ \ P\in Sat(\Sigma)$ and $\pi_X(P)\supset U\}$ This is however equal to U^* Let us suppose that there is a least fixpoint query T built from projection, join and possibly union such that $T(U^*)=U^*$ and that U^* is a least fixpoint of T containing U This would mean however that

$\pi_X(Sat(\Sigma))=\{U^*$ X is the set of attributes of $U\}=\{U\ \ T(U)=U\}$ $=Sat(\Sigma')$

where Σ' is a set of template dependencies (Horn clauses) corresponding in a standard way (see [3] to the least fixpoint operator T contrary to the Hull's result

We believe that our argument is still valid even when we allow arbitrary monotone expression built from least fixpoint operators and monotone relational algebra operations (i e all operations but difference) as possible translations of our query Q Indeed, notice that since the extension of P is empty the hypothetical translation of Q would have to be generated only by the relational symbol U (i e totally in terms of the predicate u) This can be proved impossible using arguments analogous to [5]

The above result demonstrates how the complexity of query processing grows when we admit arbitrary recursive Horn clauses in the context of incomplete information Not only a first order translation cannot be constructed here, but also no iterative program similar to the one computing the least fixpoint queries can be used as translation when we mix incomplete information with recursive rules Notice that this could be considered a price for the incompleteness of data since if the data is complete and rules are recursive such iterative program can be constructed for any query We will demonstrate now that the introduction of general disjunctive information leads to a similar growth of complexity

4 Query Processing with Non Horn Clauses

Given the conjunctive query Q and the set of formulas Σ, if the resolution tree is finite then Q^{Σ} always exists and can be computed as expression corresponding to resolution tree in a way described before This certainly does not depend on the syntax of formulas from Σ , these formulas may as well not be Horn Clauses However, if additionally to a disjunctive information we allow also recursive rules (possibly making a resolution tree infinite) then in some cases it is impossible even to find a least fixpoint translation of conjunctive queries This is obviously a consequence of the previous theorem which demonstrated a similar fact for more restricted class of rules

Theorem 3

Conjunctive queries are not least fixpoint translatable modulo Non-Horn database intensions

There exists a non-horn database intension Σ and a query Q such that no least fixpoint query will correspond to Q^{Σ} ∎

Let us then demonstrate a proper example Let

$$\Sigma = \{r(x_1, x_6) \rightarrow p(x_1, x_2, x_3) \vee p(x_4, x_5, x_6),$$

$$p(u,x',y'') \wedge p(u',x,y') \wedge p(u'',x'',y) \rightarrow p(u, x, y)\}$$

(which is a template dependency)

be two formulas of the database intension and let

$$Q(x,v) = \{x,v \mid \exists_{y,z}\ p(x,y,z) \wedge w(y,z,v)\}$$

be a query We will demonstrate that there is no lfp operator which would compute Q^{Σ} for any DB In order to see this let us observe that an arbitrary long disjunction of the form

$$p(u_1,x_1,y_1) \vee \quad \vee\ p(u_n,x_n,y_n)$$

can be produced as a consequence of Σ depending on the extension of the database

Indeed, depending on the size of instance of r a number of disjunction of the form $p(t_1) \vee p(t_2)$ can be produced If we transform a conjunction of these disjunction to the DNF and apply our rule (cartesian product) to each of the conjuncts of DNF and finally transform the formula back to CNF then we can get arbitrary long disjunctions

Let us choose R (extension of r) in such a way that all first components and all the fourth components of tuples in R are equal to the constant d Let $p(d, x_1,y_1) \vee p(d, x_n,y_n)$ be the longest disjunction which can be derived form our extension (using intension rules) such that no shorter disjunction can be obtained Let the instance of W has the form

		1	2	3
W	=	x_1	y_1	c
		x_2	y_2	c
		x_n	y_n	c

It is easy to see that Q(d,c), which is the logical consequence of this database, depends on no less than n tuples in the instance of W (n-tuples in W are necessary to derive Q(d,c)) We can make n arbitrary large such that a derivation of even one single tuple in Q depended on the arbitrary large number of tuples of database extension and could not be done in any number of "smaller" steps Since in derivation of lfp query each step (iteration) depends on some fixed, bounded number of tuples, our hypothetic translation of Q cannot be expressed by lfp query Intuitively speaking in our case each hypothetic step of the computation of the answer for the query may have to take under consideration arbitrary large (even the whole database) number of tuples (for example the whole extension of the relation W)

Conclusions

In this paper we have demonstrated a new alternative approach to query processing in the presence of view updates and generally incomplete information in a database In the positive results of the paper we have showed that conjunctive queries can be always translated to the first order form which can be directly evaluated over the database extension if the database intension is built from pseudorecursive rules (which contain possibly cyclic inclusion dependencies and major nonrecursive view definitions) We have described the translation procedure in this case

In the negative results of the paper we have showed how quickly the complexity of the queries grow if we allow both recursive rules and incompletely specified data In particular we demonstrated that

(iii) Projection queries are not even least fixpoint definable for arbitrary n-Horn clauses with skolem functions when $n \geq 3$

(iv) Conjunctive queries are not least fixpoint definable in nonhorn database intensions The last was demonstrated by showing that there are the cases in

which we cannot express the queries as iterative loops inside which some finitary (depending on the small number of tuples) computation takes place The situation becomes therefore much more complex than in the analogous recursive case for complete data

The interesting research direction is to consider an influence of functional dependencies in the cases investigated above This is a subject of the paper [7], see also [10]

Our results regarding translatability of queries, as we have mentioned in the introduction, can be further interpreted in a spirit of data and expression complexity [14] For example first order queries have data complexity in LOGSPACE and expression complexity in PSPACE while least fixpoint queries have data complexity in PTIME and expression complexity in EXPTIME

References

[1] Aho,A ,Ullman,J D
Universality of data retrieval languages
In *ACM Symposium on Principles of Programming Languages* 1979

[2] Bancilhon,F and Spyratos,N
Update Semantics of Relational Views
ACM Transactions on Database Systems 6, 1981

[3] Chandra, A and Harel, D
Horn clauses and the fixpoint hierarchy
In *ACM PODS* ACM, 1982

[4] Gallaire H , Minker J , Nicolas,J M
Logic and Databases
ACM Computing Surveys , 1984

[5] Hull,R
Finitely Specifiable Implicational Dependency Families
JACM 31, April, 1984

[6] Imielinski, T , Lipski, W
Incomplete Information in Relational Databases
JACM 31(4) 761-791, October, 1984

[7] Imielinski,T
Abstraction in Query Processing
1985

[8] Johnson,D and Klug,A
Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies
Journal of Computer System Sciences , 1984

[9] Loveland,D
Automated Theorem Proving
Springer Verlag, 1979

[10] Maier,D and Vardi,M and Ullman,J D
On the Foundations of Universal Relation Model
ACM TODS , 1984

[11] Reiter, R
On closed world databases
Logic and databases
Plenum Press, 1978

[12] Reiter,R
Deductive Query Answering on Relational Databases
Logic and Databases
Plenum Press, 1978

[13] Ullman,J
Principles of Database Systems
Computer Science Press, 1982

[14] Vardi M
The Complexity of Relational Query Language
In ACM FOCS, 1982