Knowledgebase Transformations*

Gösta Grahne†

Department of Computer Science, University of Helsinki, Teollisuuskatu 23, SF-00510, Helsinki, Finland

Alberto O. Mendelzon‡

Department of Computer Science, University of Toronto, Toronto, Canada, M5S 1A4

and

Peter Z. Revesz§

Department of Computer Science, University of Nebraska-Lincoln, Lincoln, Nebraska 68588

Received May 8, 1996

1. INTRODUCTION

It is a fact in database theory that as soon as the data model becomes slightly more general than a simple relational structure—furthermore, if one allows views in addition to stored relations—it becomes difficult to give meaning to updates [BS81, FUV83, FKUV86]. For a typical example, suppose the database is represented by the theory \( \{ A, B, A \land B \rightarrow C \} \). Let the update request be the “insertion” of the sentence \( \neg C \). Then simply adding \( \neg C \) to the theory results in inconsistency. Reasonable ways to incorporate the request could result in \( \{ A, A \land B \rightarrow C, \neg C \} \), \( \{ B, A \land B \rightarrow C, \neg C \} \), \( \{ A, B, \neg C \} \), or the disjunction of the three.

The update problem is not unique to database theory. One also encounters it in artificial intelligence [Rei92] and in belief revision theory [Mak85, Gär88]. The common fundamental question is: What should be the result of changing a theory \( T \) with a sentence \( \phi \)? The departure point of belief revision theory is the rationality postulates proposed by Alchourrón, Gärdenfors and Makinson [AGM85], and colloquially known as the AGM postulates. These are principles that every adequate belief revision operator should be expected to satisfy. For example: the new fact \( \phi \) must be a consequence of the revised theory. And: if the new fact \( \phi \) is consistent with \( T \), then the result should be logically equivalent to \( T \cup \{ \phi \} \).

However, Katsuno and Mendelzon point out in [KM91a] that all of these postulates are not universally desirable for all kinds of belief revision applications. In particular, Katsuno and Mendelzon distinguish two kinds of theory change operations, update and revision. Update consists of bringing the knowledgebase up to date when the world described by it changes. For example, most database updates are of this variety, e.g. “increase Joe’s salary by 5%.” The second type of modification, revision, is used when new information is obtained about a static world. We may for instance be trying to diagnose a faulty circuit and wanting to incorporate into the knowledgebase the results of successive tests, where newer results may contradict old ones. The following is an example of revision:

**Example 1.1.** Suppose two robot vehicles \( V \) and \( W \) are orbiting Venus. We have received the message “I have landed,” but due to noise we could not determine whether
it came from $V$ or $W$. Let $v$ be the proposition “$V$ has landed” and $w \land \neg w$ “$W$ has landed.” After receiving the message, the knowledgebase $T$ is the theory $\{v \land \neg w\} \cup \{\neg v \land w\}$. Suppose we now send the command “Land immediately” to $V$, and then $V$ replies “I have landed.” This change can be modeled by incorporating the sentence $v$ into $T$. Since $v$ is consistent with $T$, the AGM postulate cited above says the result should be equivalent $T \cup \{v\}$, which is equivalent to $\{v \land \neg w\}$. But upon reflection it becomes clear that this is incorrect. After $V$ has landed, all we know is that $V$ has landed; there is no reason to conclude that $W$ has not. The correct answer should be $\{v\}$.

The authors of [KM91a] came to the conclusion that the AGM postulates describe only revision, and gave a modified set of postulates that characterize update operators (the KM postulates).

Suppose now we want to define a language for expressing updates to databases. It seems that in defining such a language we have first to decide whether to use update or revision, or both. But, as it turns out, Gärdenfors [Gär88] has shown that if a logic has a semantics built upon Boolean algebra, then there is no way of defining a language based on revision that does not lead to triviality. On the other hand, Grahne [Gra91] has axiomatized a nontrivial logic in which update is an operator in the object language.

We shall therefore choose update as the notion of change. The KM postulates do not prescribe any particular update operator; they characterize a class of acceptable operators. As Katsuno and Mendelzon show, an operator satisfies the postulates if and only if it has the following behavior. For each model $M$ of the theory to be changed, find the set of models of the sentence to be inserted that are “closest” to $M$. The theory that describes all models obtained in this way is the result of the change operation. Choosing an update operator then reduces to choosing a notion of closeness of models. In this paper, we adopt Winslett’s possible models approach [Win89]. Loosely speaking, a relational database $D_1$ is closer than another database $D_2$ to the initial database $D$ under the Winslett ordering if every tuple that must be inserted or deleted into $D$ to make it identical with $D_1$ must also be inserted or deleted to make $D$ identical with $D_2$.

Note that we are talking about comparing databases rather than theories, equating databases with models of a theory. This is in fact our data model: we define a database to be a finite relational structure, and a knowledgebase to be a finite set of databases on the same schema. We follow the closed world assumption: only the facts that are explicitly stored are true in a database [Rei78].

Having settled on a data model and a notion of update, we propose a language that will allow both queries and updates to be expressed uniformly. In fact, in our language there is no formal distinction between queries and updates; they are both regarded as transformations. The language contains an operator that “inserts” arbitrary first-order sentences into a knowledgebase, producing a new knowledgebase.

**Example 1.2.** Suppose that we have in our knowledgebase a relation containing all the direct flight paths from cities to cities and would like to ask the following query “which cities are reachable directly or indirectly from Toronto via Air Canada?”. This query is expressed by inserting into the knowledgebase a sentence that defines a new relation containing exactly these cities, i.e., the sentence that defines the transitive closure of the given relation. (In the first example of Section 3 we elaborate further on the transitive closure query.) For expressing the update “delete flight AC902” it is enough to insert into the knowledgebase the sentence that denies the existence of this flight.

Note that, for example, positive existential relational calculus formulas are already expressive enough to formulate updates that can have multiple results. As observed in [AbG85], updates with multiple results are the source of indefiniteness in databases. Considerable expressive power is achieved by the combination of first-order logic and the minimization operator implicit in the KM notion of update and the Winslett order. It turns out that, for example, all fixpoint queries [CH82] are expressible in our transformation language. It is well-known that “inserting” a datalog program into an “extensional database” produces a unique minimal model, which model also can be characterized as a least fixpoint of the program. In case a formula has the syntax of a datalog program, or, more generally, is monotone, our update operator also produces that least fixpoint.

The major results of the paper are the following: (1) a definition of a simple and versatile first-order knowledgebase update language (2) proof that the basic update operator in this language satisfies the Katsuno–Mendelzon postulates for updates (3) an analysis of the computational complexity and expressive power of the knowledgebase update language.

In Section 2 of this paper we lay the foundation of knowledgebase transformations. In Section 3 some examples of transformations are exhibited. The computational complexity of transformations is the subject of Section 4, and expressive power is discussed in Section 5.

### 2. THE FRAMEWORK

As a notational convenience, the symbol $\omega$ denotes the set of all natural numbers, while the symbol $\otimes$ denotes the symmetric set difference operation, that is, $A \otimes B = (A \setminus B) \cup (B \setminus A)$.

Consider a first-order function free language $\mathcal{L}$ built from the following components: A set $A = \{a_i : i \in \omega\}$ of domain elements, a set $X = \{x_i : i \in \omega\}$ of variables, a set $R = \{R_i : i \in \omega\}$ of relation symbols, $\land$ (and), $\neg$ (negation),
3 (existential quantifier), = (equality), and the parenthesis symbols.

With each relation symbol \( R \in \mathcal{R} \) we associate the arity \( |R| \). A \( k \)-ary term is a tuple with \( k \) components, each in \( A \supseteq X \). An atomic formula is an expression of the form \( R(x) \) where \( R \) is in \( \mathcal{R} \) and \( x \) is an \( |R| \)-ary term, or an expression of the form \( x_i = x_j \), or an expression of the form \( x_i \in X \), or \( x_i \in A \), where \( \{x_i, x_j\} \subseteq X \) and \( a_i \in A \).

The set of all well formed formulas of \( \mathcal{L} \) is defined in the usual way, and it is denoted \( \Phi \). The subset of sentences in \( \Phi \) is denoted \( \Phi^s \). If \( \phi \) is a formula where variable \( x_i \) occurs free, then \( \phi(x_i/a_j) \) denotes the formula \( \phi \) with each free occurrence of \( x_i \) substituted by \( a_j \).

A database \( db \) is a sequence \( \langle r_{i1}, \ldots, r_{in} \rangle \) of relations, where each \( r_i \) is a finite subset of \( A^{r_i}(x) \). Then the schema of \( db \) is \( \sigma(db) = \{ R_{i1}, \ldots, R_{in} \} \).

For \( \phi \in \Phi \), define the schema \( \sigma(\phi) \) to be the set of all relation symbols appearing in \( \phi \).

The set of all databases is denoted \( \mathcal{D} \). By \( \mathcal{D}^s \), we mean the set of all databases on schema \( s \). Furthermore, if \( B \) is a subset of the domain \( A \), then \( \mathcal{D}^s_B \) denotes the set of all databases on schema \( s \) containing only values in \( B \).

Let \( db_1 \) and \( db_2 \) be databases. Then we say that \( \sigma(db_2) \) dominates \( \sigma(db_1) \), if \( \sigma(db_1) \) is a subset of \( \sigma(db_2) \).

The following is a key definition in this paper. It defines a partial order relation \( \preceq \) with respect to a given database. Intuitively, this relation is used to rank the desirability of including a database within the updated knowledgebase.

**Definition 2.1.** Let \( db_1 = \langle s_{i1}, \ldots, s_{in} \rangle \) and \( db_2 = \langle u_{i1}, \ldots, u_{in} \rangle \) be databases, where \( \sigma(db_1) = \sigma(db_2) \), and let \( db \) be such that \( \sigma(db) \) dominates \( \sigma(db_1) \).

Then define a relation \( \preceq_{db} \) over \( \mathcal{D}^s_{\sigma(db_1)} \), such that \( db_1 \preceq_{db} db_2 \) if and only if either

\[
s_{ij} \cap r_{ij} \subseteq u_{ij} \cap r_{ij} \quad (1)
\]

for all \( r_{ij} \), \( s_{ij} \), and \( u_{ij} \), whose schemas occur in all three databases, or

\[
s_{ij} \cap r_{ij} = u_{ij} \cap r_{ij} \quad (2)
\]

for all \( r_{ij} \), \( s_{ij} \), and \( u_{ij} \), whose schemas occur in all three databases, and

\[
s_n \cap \emptyset \subseteq u_n \cap \emptyset \quad (3)
\]

hold for the rest of the relations \( s_n \) and \( u_n \) in \( db_1 \) and \( db_2 \).

We give an example of partial order. Let \( db_1 = \langle \{ R(a_1, a_2), S(a_1, a_4) \} \rangle \) and \( db_2 = \langle \{ R(a_1, a_2), S(a_1, a_4), S(a_2, a_4) \} \rangle \) and \( db = \langle \{ R(a_1, a_2) \} \rangle ). \)

Here \( \sigma(db_1) = \sigma(db_2) = \{ R, S \} \) and \( \sigma(db) = \{ R \} \). Since all three relations agree in \( R \), and \( S \) in \( db_1 \) is a strict subset of \( S \) in \( db_2 \), we have that \( db_1 \preceq_{db} db_2 \).

From this example, we can easily see that \( \preceq_{db} \) is a partial order.

This partial order compares the databases in \( \mathcal{D}^s_{\sigma(db_1)} \) with respect to their closeness to the database \( db_1 \). If \( \sigma(db) = \sigma(db_1) = \sigma(db_2) \), then we have \( db_1 \preceq_{db} db_2 \) if and only if the symmetric difference between \( db_1 \) and \( db_2 \) is included in the symmetric difference between \( db_2 \) and \( db_2 \), where the symmetric difference is taken componentwise. This corresponds to the way interpretations are compared in Winslett’s possible models approach [Win89]. If the schemas of \( db_1 \) and \( db_2 \) are proper supersets of \( db \), the comparison will proceed in two stages. First, we try to keep the relations in \( db \) invariant. Since \( r \sqcup r = \emptyset \), for any relation \( r \), condition (1) will guarantee that the databases where the relations in \( db \) are invariant will be closer to \( db \) than other databases. These other databases will be ordered in two stages: First, smaller changes to the relations in the \( db \) are favored (condition (2)), then the other relations are compared w.r.t. the empty set, since these relations are not present in \( db \) (condition (3)). Since \( r \sqcup r = r \), for any relation \( r \), databases with smaller relations will be favored in the order.

A knowledgebase \( kb \) is a finite set of \( db \)’s with the same schema. This schema is also the schema of the knowledgebase. The set of all knowledgebases is denoted \( \mathcal{K} \).

The interpretation of a sentence \( \phi \in \Phi \) w.r.t. a database \( db \) is a relation \( |= \) on \( \mathcal{D} \times \Phi \) defined for \( db \) and \( \phi \) if and only if \( \sigma(db) \) dominates \( \sigma(\phi) \), in which case the recursive definition is

\[
db |= (a_i = a_j) \quad \text{if} \quad i = j 
\]

\[
db |= R(x) \quad \text{if} \quad x \in r \n\]

\[
db |= (\phi \land \psi) \quad \text{iff} \quad db |= \phi \quad \text{and} \quad db |= \psi 
\]

\[
db |= (\neg \phi) \quad \text{if not} \quad db |= \phi 
\]

\[
db |= (\exists x \phi) \quad \text{iff} \quad db |= \phi(x/a_j) \quad \text{for some} \quad a_j \in A 
\]

By \([\phi]\) we mean the set of all databases such that \( |= \) is true. We call \([\phi]\) the models of \( \phi \). We say that \( \phi \) finitely implies \( \psi \), if \( [\phi] \subseteq [\psi] \).

If \( s \) is a schema and \( B \) is a subset of the domain \( A \), then \( [\phi]_B^s \) denotes the set \([\phi] \cap \mathcal{D}^s_B \).

Let \( DB \subseteq \mathcal{D} \) and \( \{ db_1, db_2 \} \subseteq \mathcal{D} \). Then we say that \( db_1 \) is \( \preceq_{db} \)-minimal in \( DB \), if \( db_1 \) is in \( DB \), and if \( db_2 \in DB \) and \( db_2 \preceq_{db} db_1 \) entails \( db_2 = db_1 \).

Now consider the function \( \mu : \Phi \times \mathcal{D} \to \mathcal{K} \), defined as

\[
\mu(\phi, db) = \{ db' \in \mathcal{D}^s_B, db' \text{ is } \preceq_{db} \text{minimal in } [\phi]_B^s \} \quad (9)
\]
where \( B \) is the smallest subset of the domain \( A \), such that \( B \)
contains all values that appear in \( db \) and \( \phi \), and \( \varepsilon \) is the
sigma (\( db \cup \sigma \phi \)).

The function \( \mu \) thus picks the models of \( \phi \) that are closest
to \( db \) among the databases that have no other relations than
those in \( db \) and those mentioned in the sentence \( \phi \), and no other
values than those appearing in \( db \) or in \( \phi \).

Next, consider the (partial) transformation functions
\[ \tau : \Phi \times \mathcal{K} \rightarrow \mathcal{K} ; [\Phi] : \mathcal{K} \rightarrow \mathcal{K} ; [\cdot] : \mathcal{K} \rightarrow \mathcal{K} ; \text{ and} \]
\[ \pi_1, \ldots, \pi_n : \mathcal{K} \rightarrow \mathcal{K} , \text{ where} \]
\[ \tau_a(kh) = \bigcup_{db \in kh} \mu(\phi, db). \]  

and \( [\phi] \) is the componentwise intersection of the databases
in \( kh \), i.e., the \( gh \) of \( kh \) w.r.t. to the Cartesian
generalization of subset inclusion. Likewise, \( [\cdot] \) is the
componentwise union of \( kb \). As an example, let \( kb = \{ \langle a_1, a_2, a_3 \rangle, \langle a_4, a_5, a_6 \rangle \} \). Here the knowledge-
base contains two databases with the same schema. The value of \( [\phi] \) is \( \{ \langle a_1, a_2, a_3 \rangle \} \), and the value of \( [\cdot] \) is \( \{ \langle a_4, a_5, a_6 \rangle \} \).

A function \( \pi \) maps a \( kh \) to the \( kb \) that is obtained by
projecting each \( db \) in the input to the components whose
indices appear in the subscript of \( \pi \).

The function \( \tau \) can be seen as an update function, in that
\( \tau_a(kh) \) “inserts” \( \phi \) into \( kh \), producing a new \( kh \). In the
space example in Section I, the knowledgebase \( T \) would be
\( kh = \{ \langle v \rangle, \langle w \rangle \} \), where \( v \) is a tuple corresponding to the
proposition “\( V \) has landed,” and likewise for \( w \). Let these
tuples be over schema \( R \). Then, we learn that \( V \) has landed
we perform the update
\[ \tau_{R \{ v \}}(kh) = \{ \langle v \rangle, \langle v, w \rangle \} . \]

Note. We do not need to express the traditional projection
operator, because it can be expressed by the above
more general projection and the transformation operators.
For example, suppose that we have a knowledgebase with
schema \( [R] \) where \( R \) is a binary relation symbol and that
the knowledgebase contains a single database in it. Suppose
that we want to express the regular projection operation
onto the first column of \( R \). This can be accomplished by
inserting the sentence.
\[ \pi_{R \{ v \}} \cdot \tau_{R \{ v \}} \cdot \pi_{R \{ v \}}(kh). \]

The output relation \( R \) will be the projection as desired. It
similarly can be checked that the standard selection operation
is also expressible in our transformation language. In
general, it turns out that the following properties, corres-
cponding to the KM-postulates [KM91a] for update, hold.

**Theorem 2.1.** For all schema-wise appropriate \( db, kb, \)
\( kb_1 \), and \( kb_2 \):

(i) \( \tau_a(kh) \subseteq [\phi] \)

(ii) If \( kb \subseteq [\phi] \), then \( \tau_a(kh) = kb \)

(iii) If \( kb \not\subseteq \emptyset \) and \( [\phi]_g \\emptyset \not\subseteq \emptyset \), then \( \tau_a(kh) \not\subseteq \emptyset \)

(iv) \( [\phi] = [\psi] \), then \( \tau_a(kh) = \tau_a(kh) \)

(v) \( \tau_a(kh) \cap [\psi] \subseteq \tau_a(kh) \)

(vi) \( \tau_a(kh) \subseteq [\psi] \), and \( \tau_a(kh) \subseteq [\phi] \), then \( \tau_a(kh) = \tau_a(kh) \)

Proof. (i) Let \( db \in \tau_a(kh) \). From (9) it follows that
\( db \in [\phi]_g \). Since \( [\phi]_g = [\phi]_g \), the claim follows.

(ii) Let \( db' \in \tau_a(kh) \). Then there is a \( db \in kb \), such that
\( db' \) is \( \leq_g \)-minimal in \([\phi]_g \). Since \( kb \subseteq [\phi]_g \), we have
\( db' \subseteq [\phi]_g \). Now \( db \) is the unique \( \leq_g \)-minimal element in
\([\phi]_g \). Thus \( db' = db \), and consequently \( \tau_a(kh) \subseteq [\phi]_g \).

For inclusion in the other direction, let \( db \in kb \). Since
\( kb \subseteq [\phi]_g \), \( db \) is the unique \( \leq_g \)-minimal element in
\([\phi]_g \). Therefore \( db \in \tau_a(kh) \).

(iii), (iv), and (viii) These are immediate consequences of
definitions (9) and (10).

(v) Let \( db' \in \tau_a(kh) \cap [\psi] \not\subseteq \emptyset \). Then there is a \( db \in kb \), such that
\( db' \in \mu(\phi, db) \), and \( db' \in [\phi \land \psi]_g \). Here \( B \) is the set of values appearing in \( db \) or \( \phi \). Suppose now
towards a contradiction that \( db' \) is not in \( \mu(\phi \land \psi, db) \). Then
there must be some database, say \( db'' \), such that \( db'' \leq_g db' \),
\( db'' \not\subseteq db' \), and \( db'' \in [\phi \land \psi]_g \). Here \( B' \) is the set
of values appearing in \( db'' \). Then it follows that \( db'' \) also
is in \( [\phi]_g \). Thus we have a contradiction to the fact
that \( db' \not\in \mu(\phi, db) \).

(vi) We show that, given the assumptions, \( \tau_a(kh) \subseteq \tau_a(kh) \).
Let \( db' \in \tau_a(kh) \). Then there is a \( db \in kb \), such that
\( db' \in \mu(\phi, db) \). We claim that \( db' \not\in \mu(\psi, db) \). Suppose the
contrary. We can assume \( w.l.o.g. \), that \( \sigma(\phi) = \sigma(\psi) \), and that
the values appearing in \( \phi \) and \( \psi \) are the same. Let \( B \) be the
set of values in \( db \) and \( \phi \). Since \( db' \in [\phi \land \psi]_g \), the set
\( \mu'(\phi, db') \) must be nonempty. Let therefore \( db'' \in \mu(\phi, db) \).
Then it must be that \( db'' \leq_g db' \), and \( db'' \not\subseteq db' \). But this is
not possible, since \( db'' \in [\phi \land \psi]_g \), thus yielding a contradiction
to the fact that \( db' \not\in \mu(\phi, db) \). The argument for
inclusion in the other direction is symmetrical.

(viii) Let \( db' \in \mu(\phi, db) \cap \mu(\psi, db) \). Again, can assume
w.l.o.g. that \( \sigma(\phi) = \sigma(\psi) \). Let \( B \) be the set of values appearing
in \( db \) or \( \sigma \), and \( B' \) the set of those appearing in \( db \) or \( \psi \).
From the assumption it follows that \( \mu(\phi \lor \psi, db') \not\subseteq \emptyset \). If
\( db' \not\in \mu(\phi \lor \psi, db) \), there must be a \( db'' \in \mu(\phi \lor \psi, db) \), such that
\( db'' \leq_g db', \) and \( db'' \not\subseteq db' \). It now follows that \( db'' \) is in
\([\phi \lor \psi]_g \), or in \([\psi]_g \). In the first case we have a
contradiction to the fact that \( db' \in \mu(\psi, db) \), and in the second case to the fact that \( db' \in \mu(\psi, db) \).

The next lemma illustrates an elementary property of the transformation language. It shows that updates do not commute with \( g \) and \( l \).

**Lemma 2.1.** There are knowledgebases and sentences such that \( \bigcap (\tau_d(kb)) \neq \tau_d(\bigcap (kb)) \), and \( \bigcap (\tau_d(kb)) \neq \tau_d(\bigcup (kb)) \).

**Proof.** Let \( kb = \{ \langle a_1a_2a_3 \rangle, \langle a_1a_2a_4 \rangle \} \), that is, two databases, each with a single tuple over schema, say, \( R_1 \). Then the value of

\[
\bigcap (\tau_{v_{y_{i,j}}}R_{y_{i,j}}x_{i,j}y_{i,j} - R_2 ) (kb)
\]

is \( \{ \langle G, \{ a_1 \} \rangle \} \), where the second relation is on schema \( R_2 \).

On the other hand, commuting \( \bigcap \) and \( \tau \) produces \( \{ \langle G, \emptyset \rangle \} \), or \( \{ \langle a_1a_2 \rangle \} \), \( \{ \langle a_2a_3 \rangle \} \), with schema \( R_1 \). Then the value of

\[
\tau_{v_{y_{i,j}}}R_{y_{i,j}}x_{i,j}y_{i,j} - R_2 ) (kb)
\]

is \( \{ \langle a_1a_2, a_2a_3 \rangle, \{ a_1a_2, a_2a_3, a_1a_3 \} \} \), while commuting the operators results in \( \{ \langle a_1a_2, a_2a_3 \rangle, \{ a_1a_2, a_2a_3, a_1a_3 \} \} \).

By composing the transformation functions in the obvious way, and by using schemas as parameters, transformation expressions can be formed. For example, \( \pi_1(\tau_{v_{a_1}}(\bigcap (\tau_d(R_1, R_2)))) \) is a transformation expression. The value of this expression, when applied to a knowledgebase, says \( \{ \langle r_1, r_2 \rangle, \langle s_1, s_2 \rangle \} \), is the value of the transformation obtained by substituting the parameters by the knowledgebase, that is \( \pi_1(\tau_{v_{a_1}}(\bigcap (\tau_d(\{ \langle r_1, r_2 \rangle, \langle s_1, s_2 \rangle \})))) \). Transformation expressions will be denoted by \( \Theta \), \( \Theta' \), ..., while \( \Theta \) denotes the set of all transformation expressions. In the sequel we shall leave out extra parenthesis symbols whenever there is no risk of confusion.

### 2.1. Comparison with Related Work

At this point it is possible to compare our approach with some previous work on update languages.

Abiteboul and Vianu [AV87, AV88] define a class of non-deterministic transformations on databases that they call *updates*. This class is similar to our transformations in that it includes queries and modifications of the database state as special cases. They define an update as a relation between instances of a fixed schema \( \sigma \) and another fixed schema \( \tau \) that is recursively enumerable and \( C \)-generic for some finite \( C \). (Recall that a binary relation \( r \in A \times B \) is \( C \)-generic, for \( C \) a subset of \( A \cup B \), if for every bijection \( \rho \) over \( A \cup B \) which is the identity on \( C \), \( (i, j) \in r \) if and only if \( (\rho(i), \rho(j)) \in r \).) If we restrict our attention to the case where the input knowledgebases are singletons, i.e. databases, it is clear that the transformations defined by \( \Theta \) expressions are updates in the sense of Abiteboul and Vianu.

In fact, they are within the subclass that they call *finitely nondeterministic updates*, those in which the set of all output databases related to each input database is finite. They also consider *deterministic updates*, in which the relation is a function. In Section 5, we consider a class of \( \Theta \) expressions that falls within this subclass: those expressible by a transformation on the form \( \Pi \bigwedge \tau \), where each \( b \) is one of \( \bigcap \) or \( \bigvee \).

The work of Fagin et al. [FUV83, FKUV86] considers updates of logical databases, where the database is described by a set of sentences. The essential idea in [FUV83] is to consider all maximal subsets of the set of sentences in the database that is consistent with the sentence that is inserted. Although this definition seems intuitive, we do not follow this because it does not satisfy the Katsuno–Mendelzon postulates for updates. The essential problem is that it does not satisfy the principle of the irrelevance of syntax, i.e., the results of the update should not depend on the exact syntax of the set of sentences in the knowledgebase, but only on their semantics. This is in fact one of the harder postulates to satisfy and in [KM91a, KM91b] it is shown that several other update operator proposals also do not satisfy this principle. Another difference between the update operator in [FUV83] and in this paper is that in the former priorities, that is integer numbers, are assigned to sentences and these are also taken into account in the definition of maximum consistent subset. Furthermore, [FUV83] considered inserting only a single sentence into the database. This limitation was removed by [FKV86] that defined the flock semantics for updates that allows insertion of a group of sentences. Our update operator also allows the simultaneous insertion of a group of sentences by treating each group of sentences as the conjunction of the sentences. It is well-known that if the Horn clause form of logic programs is relaxed, then there might be several least fixpoints of a program. In this case our update operator produces all least fixpoints w.r.t. a generalization of the usual subset inclusion into the partial order based on Boolean sum. This is in contrast to the more common approach to designate one of the models, or some other relational structure, as the "intended model" of the program. As is demonstrated in [IN88] such intended models might force the programmer to express his or her intentions in a cumbersome way. We do however note that the *iterative fixpoint* [ABW88] of a stratified program can be obtained in our language by sequentially updating the database with the strata of the program in their hierarchical order.

It is also interesting to note that *hypothetical queries* [Bon88, Gab85] and *queries on recursivey indefinite databases* [Mey90] can be expressed through updates. The
connection between hypothetical queries and updates is explored in [GM95].

3. SAMPLE TRANSFORMATION

In this section we present seven example transformations. First we show that the common queries of transitive closure (Example 1) and parity (Example 6) can be expressed. We also show that the robots query described in the introduction can also be expressed (Example 4). The remaining examples express increasingly harder queries on graphs. In the presentation, the examples used here also illustrate the high degree of modularity inherent in our transformation language. In the presentation, the examples are ordered in increasing level of difficulty.

**Example 1.** Let \( r \) be a relation such that \( \sigma(r) = R_1 \), and let \( \phi \) be the sentence

\[
\forall x_1 x_2 x_3 : (R_2 x_1 x_2 \land R_1 x_2 x_3) \lor R_1 x_1 x_3 \rightarrow R_2 x_1 x_3.
\]

**Explanation.** \( \pi_2 \tau_{\phi, \delta}(\{\langle r_1 \rangle\}) = \{\langle s \rangle\} \), where \( s \) is the transitive closure of \( r \) and \( \sigma(s) = R_2 \). To see that this is indeed the case, note that by definition (10), the databases returned by \( \tau_{\phi} \) are models of sentence \( \phi \). For a database to be a model, it has to satisfy both of the conditions

\[
\forall x_1 x_2 x_3 : (R_2 x_1 x_2 \land R_1 x_2 x_3) \lor R_1 x_1 x_3 \rightarrow R_2 x_1 x_3.
\]

By the first condition, \( s \) must contain \( r \). By the second condition, for any path in \( r \), that is, any set of tuples \( \{a_1 a_2, ... a_{k-1} a_k\} \subseteq r \), there must be an edge \( a_1 a_2 \in s \). This can be proven by induction. Hence, the relation \( s \) must contain also the transitive closure of \( r \). By definition (9), the relation that results from applying the transformation must be minimal. In addition, the definition of minimality ensures that the argument relation \( r \) is not altered while an \( s \) can be found such that \( \langle r, s \rangle \) is a model of \( \phi \). Clearly, for any \( r \) we can always find a transitive closure \( s \), hence \( r \) will never need to be changed according to the minimality condition. Therefore, after the projection operation, the result will be exactly the transitive closure of \( r \).

**Example 2.** To transform a directed graph \( r_1 \) into the set of its transitive reductions, let \( \psi \) be the following sentence:

\[
\forall x_1 x_2 : R_2 x_1 x_2 \rightarrow R_1 x_1 x_2.
\]

Also, let \( \chi \) be the conjunction of sentences

\[
\forall x_1 x_2 x_3 : (R_2 x_1 x_2 \land R_1 x_2 x_3) \lor R_1 x_1 x_3 \leftrightarrow R_3 x_1 x_3,
\]

\[
\forall x_1 x_2 x_3 : (R_3 x_1 x_2 \land R_2 x_1 x_3) \lor R_2 x_1 x_3 \leftrightarrow R_3 x_1 x_3.
\]

**Explanation.** \( \pi_2 \tau_{\phi, \delta}(\{\langle r_1 \rangle\}) = \{\langle r_2 \rangle, ... \langle r_5 \rangle\} \), where each \( r_5 \) is a transitive reduct of \( r_1 \). Recall that a binary relation \( r_2 \) is a transitive reduction of \( r_1 \) if and only if \( r_2 \) is an antitransitive subset of \( r_1 \), and the transitive closure of \( r_2 \) is the same as the transitive closure of \( r_1 \). (Here antitransitivity means that for any \( a_1 a_2 \) in \( r_2 \), there is no \( a_2 \) such that both \( a_1 a_2 a_3 \) and \( a_2 a_3 a_1 \) are also in \( r_2 \).) Note also that by definition (9), the relation \( r_2 \) does not change if \( r_5 \) and \( r_1 \) can be found. This will indeed be the case.

At first we check that \( r_5 \) satisfies the second requirement to be a transitive reduction of \( r_1 \). By the first part of sentence \( \chi \), the relation \( r_5 \) is the transitive closure of \( r_1 \), because it is just like in Example 1, except for the bidirectionality of the implication. By sentence \( \psi \), \( r_2 \) must be a subset of \( r_1 \), and by the second part of sentence \( \chi \), this subset must have the same transitive closure as \( r_1 \) has. Here we need the bidirectionality.

Second, for \( r_2 \) to be a transitive reduction of \( r_1 \), it also has to be antitransitive. This condition is satisfied by the minimality requirement. For suppose that \( r_2 \) is not antitransitive. Then \( r_2 \) must contain a certain subset, say \( \{a_1 a_2, a_2 a_3, a_1 a_3\} \). Clearly, the relation \( r_2 \setminus \{a_1 a_2\} \) has the same transitive closure as \( r_2 \) has, but it has fewer tuples, and will hence by \( \mu \) be preferred over \( r_2 \).

Since the two conditions are satisfied, after operation \( \pi_2 \) the knowledgebase will indeed contain the set of transitive reducts of the original graph.

**Example 3.** Suppose now that we would like to know whether a certain set of edges belongs to every transitive reduction of a graph. This query can be expressed in the language of recursively indefinite database [Mey90]. The query can also be formulated as a transformation expression. Suppose that the relation \( r_4 \) describes the set of edges in question. Let the sentence \( \zeta \) be

\[
\forall x_1 x_2 : (R_3 x_1 x_2 \rightarrow R_2 x_1 x_2) \rightarrow R_4.
\]

**Explanation.** The transformation \( \pi_2 \tau_{\theta, \delta}(\{\langle r_1, r_2 \rangle\}) \), where \( \theta = \pi_2 \land \tau_{\phi, \delta}(\{\langle r_1, r_2 \rangle\}) \) will yield one zeroary relation \( r_4 \) that will contain the empty tuple if and only if \( r_5 \subseteq r_3 \) where \( r_5 \) stores the set of edges that belong to all transitive reductions of graph \( r_1 \), and \( r_3 \) is the given set of edges.

To see this, note that \( \theta(\{\langle r_1, r_2 \rangle\}) = \{\langle r_2, r_1 \rangle\} \) is as in the previous example, except that \( r_1 \) is present and preserved unchanged by \( \theta \), and we take \( \land \) the common set of edges in \( r_2 \).

In the \( \tau_{\phi} \) operation we added to our sentence \( \psi \) an implication for relation \( r_4 \). Here the minimality requirement assures that \( r_4 \) will be empty if and only if \( r_5 \) is not a subset of \( r_2 \).

**Example 4.** Transformation expressions can also describe hypothetical, or subjective queries. Recall the space
example from Section 2. Let the knowledgebase be \( kb = \langle \{ v \} \rangle , \langle \{ w \} \rangle \), and consider the query “if \( V \) had landed, would \( W \) be necessarily still orbiting \( ? \)”. The answer to this query would be “yes” if and only if the resulting singleton knowledgebase of the transformation

\[ \bigcup \tau_{R(r,v)}(kb). \]

does not contain \( w \). Since \( \bigcup \tau_{R(r,v)}(kb) = \langle \{ v, w \} \rangle \) this is not the case.

**Note.** The above example expresses a type of hypothetical queries called **counterfactual** queries. A counterfactual query, denoted as \( A > B \), has two parts an antecedent (\( A \)) and a consequent (\( B \)), and the antecedent is known to be false. A counterfactual query is true whenever “if the antecedent were true then the consequent would be also true”. It is possible to generalize from this example to any right-nested \((A > (B > C))\) \cdots** counterfactual query. These can be expressed by nested transformations \( \tau_{r} (\tau_{B}(\tau_{C})) \) \cdots.

**Example 5.** Now consider the **monochromatic triangle** problem, that is, the problem of deciding whether an undirected graph \( r_{1} \) has a partition into two graphs \( r_{2} \) and \( r_{3} \) such that both \( r_{2} \) and \( r_{3} \) are antitransitive.

Let \( v \) be the sentence

\[ \forall x_1 x_2 : R_1 x_1 x_2 \rightarrow R_2 x_1 x_2 \lor R_3 x_1 x_2. \]

Let \( p \) be the conjunction of sentences

\[ \forall x_1 x_2 : R_2 x_1 x_2 \land R_2 x_2 x_3 \rightarrow \neg R_2 x_1 x_3, \]

\[ \forall x_1 x_2 : R_3 x_1 x_2 \land R_3 x_2 x_3 \rightarrow \neg R_3 x_1 x_3, \]

\[ \forall x_1 x_2 : R_4 x_1 x_2 \leftrightarrow R_2 x_1 x_2, \]

\[ \forall x_1 x_2 : R_3 x_1 x_2 \leftrightarrow R_2 x_2 x_1, \]

\[ \forall x_1 x_2 : R_4 x_1 x_2 \leftrightarrow R_3 x_2 x_1. \]

Let \( \varphi \) be the expression \( \forall x_1 x_2 : R_2 x_1 x_2 \lor R_3 x_1 x_2 \). Let \( \varphi \) be the expression \( \forall x_1 x_2 : (R_2 x_1 x_2 \land R_3 x_1 x_2) \rightarrow R_4 x_1 x_2. \)

Let \( \zeta \) be the conjunction of sentences

\[ \forall x_1 x_2 x_3 : R_4 x_1 x_2 \land R_4 x_1 x_3 \rightarrow x_2 = x_3, \]

\[ \forall x_1 x_2 x_3 : (R_2 x_1 x_2 \land R_4 x_1 x_3) \rightarrow x_2 = x_3. \]

Let \( \zeta \) be the expression \( \forall x_1 x_2 : (R_2 x_1 x_2 \lor R_4 x_1 x_2) \rightarrow R_4 x_1 x_2. \)

Let \( \tau \) denote the transformation that assigns \( r_{1} \) to \( r_{6} \). Let \( \theta \) be the expression \( \pi_{1,5} \pi_{6} \pi_{5} \pi_{6} \tau_{\langle \{ r_{1} \} \rangle} (R_{0}). \)

**Explanation.** The transformation \( \pi_{6} \pi_{5} \theta (\langle \{ r_{1} \} \rangle) \) results in a knowledgebase which has a database in which the relation \( r_{6} \) is empty if and only if the partition of \( r_{1} \) is even.

Note that \( r_{1} \) has an even number of elements if and only if \( r_{1} \) can be partitioned into two unary relations \( r_{2} \) and \( r_{3} \) of the same cardinality. This serves as the basic intuition behind the transformation expression. In other words, we have here an expression that results in a knowledgebase containing a single database that contains the relation \( r_{6} \) as empty if and only if \( r_{1} \) has the desired partition.
Clearly, the sentence $v'$ is similar to $v$ in Example 5 and says that $r_2$ and $r_1$ form a partition. Now $r_2$ and $r_1$ have an equal number of elements if and only if there is a function from $r_2$ to $r_1$ that is bijective. In our example, the relation $r_4$ is such a bijection.

The knowledgebase after the insertion of $v'$ will contain a set of databases each of which specifies a partition of $r_1$ into an $r_2$ and $r_3$. Next the insertion of $\varphi$ into the knowledgebase will result in the addition of an $r_4$ that is the Cartesian product of $r_2$ and $r_3$ within each of the databases. The insertion of $\zeta$ will eliminate all but those databases where $r_4$ is a bijection of $r_2$ and $r_3$.

The insertion of $\varphi$ adds a new relation $r_4$ that will contain all the elements occurring in $r_2$ (either as first or second argument) in each database. We claim that $r_4$ will be equal to $r_1$ in a database if and only if $r_1$ has an even number of elements.

To prove if: if $r_1$ has an even number of elements, then there is clearly a partition into $r_2$ and $r_1$ that have the same number of elements and to there an $r_4$ that is a bijection. The first arguments of $r_4$ will be equal to the elements of $r_2$ and the second arguments of $r_4$ will be equal to $r_1$, hence the union of these which will be output as $r_5$ must be the same as $r_1$.

To prove only if: if there is no database with $r_4$ equal to $r_1$, then in any database there must be some item $a$ in $r_1$ that does not occur as either the first or the second argument of $r_4$ after the insertion of $\zeta$. We know that $a$ must be in either $r_2$ or $r_3$. Let us suppose without loss of generality that $a$ is in $r_2$. Then since $r_2$ and $r_3$ is a partition, it cannot belong to $r_3$ (partitions are disjoint and the disjointness is enforced by the minimality condition as in Example 5). Since $a$ belongs to $r_2$ and it is not paired by any element of $r_3$ there must be an odd number of elements. (Note that $r_3$ cannot also have an element $b$ that is not paired with any element in $r_2$ because that would contradict the minimality condition, i.e., from the Cartesian product of $r_2$ and $r_3$ the relation $r_4 \cup \{(ab)\}$ would be closer than the relation $r_3$ obtained after insertion of $\zeta$.) Hence $r_1$ must have an odd number of elements.

Therefore it is clearly enough to check that in the knowledgebase that results after performing $\theta$ one of the databases contains an $r_1$ that is equal to $r_1$. We can use here the transformation $r_2$ similarly to the $r_2$ transformation in Example 5. Then $r_2$ will be the empty relation in one of the databases if and only if the initial $r_1$ had an even number of elements.

Example 7. The maximal clique problem asks for a graph whether the largest clique or maximal complete subgraph has exactly size $k$.

Let $r_1$ be the set of edges of a graph. Let $r_2$ be any set with exactly $k$ elements and $r_3$ be any set with exactly $k+1$ elements. Let $\phi$ be the following conjunction of sentences:

\[
\forall x_1 \exists x_2 : R_2 x_1 \rightarrow R_2 x_1 x_2,
\]

\[
\forall x_1 \exists x_2 : R_3 x_1 \rightarrow R_3 x_1 x_1,
\]

\[
\forall x_1 x_2 x_3 : R_3 x_2 x_1 \land R_3 x_3 x_1 \rightarrow x_2 = x_3,
\]

\[
\forall x_1 x_2 x_3 : R_3 x_1 x_2 \land R_3 x_1 x_3 \rightarrow x_2 = x_3,
\]

\[
\forall x_1 x_2 : R_4 x_1 \land R_4 x_2 \land x_1 \neq x_2 \rightarrow R_1 x_1 x_2.
\]

Explanation. Transformation $r_2$ can be used to check whether the graph has a clique of size $k$.

If the graph has a clique of size $k$, then the vertices of one such clique will be placed in the unary relation $r_4$. To see that, note that besides $r_4$, an $r_4$ relation has to be found. Since $r_5$ is a new relation, it must be a minimal-size relation. Also by the first four lines, $r_4$ is a bijection from $r_2$ to $r_4$. Hence the size of $r_4$ and $r_2$ will be the same. Hence the size of $r_4$ will be $k$ as required. The last line assures that between each distinct pair of elements in $r_4$ there is an edge in the graph. In other words, the elements of $r_4$ are vertices and form a clique.

If the graph does not have a size $k$ clique, then either of the two input relations $r_1$ or $r_2$ will be changed. By making copies of these relations before the above transformation and comparing them to the values of $r_1$ and $r_2$ after the transformation we can test whether the graph has a size $k$ clique.

Note that we need to test not only that the graph has a clique of size $k$ but also that it is maximal. Clearly, if $k$ is maximal, then the graph has no clique of size $k + 1$. To test that, we can reuse the above query, after an appropriate renaming of the relations. We also have to use here the input relation $r_1$ of size $k + 1$.

\[
\forall x_1 \exists x_2 : R_3 x_1 \rightarrow R_3 x_1 x_2,
\]

\[
\forall x_1 \exists x_2 : R_2 x_1 \rightarrow R_2 x_1 x_2,
\]

\[
\forall x_1 x_2 x_3 : R_3 x_2 x_1 \land R_3 x_3 x_1 \rightarrow x_2 = x_3,
\]

\[
\forall x_1 x_2 x_3 : R_3 x_1 x_2 \land R_3 x_1 x_3 \rightarrow x_2 = x_3,
\]

\[
\forall x_1 x_2 : R_4 x_1 \land R_4 x_2 \land x_1 \neq x_2 \rightarrow R_1 x_1 x_2.
\]

The above transformation will either not change $r_1$ and $r_3$ and find a clique of size $k + 1$ and place it in $r_2$, or change $r_1$ and $r_3$ when there is no clique of that large size. By using again copies of $r_1$ and $r_3$, we can tell which of the two cases occurred and construct a query that answers either true of false as required.

**4. Computational Complexity**

In this section we will examine the computational complexity of transformation expressions. We will consider both
data and expression complexities in separate subsections. In a third subsection we also consider the special case of transformation expressions built from quantifier free formulas.

The main complexity results of this section are summarized in the following table:

<table>
<thead>
<tr>
<th>Transformation class</th>
<th>Data complexity</th>
<th>Expression complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>(τ, π)</td>
<td>∈co-NP</td>
<td>∈co-NEXPTIME</td>
</tr>
<tr>
<td>θ</td>
<td>∈PSPACE</td>
<td>∈EXPSPACE</td>
</tr>
</tbody>
</table>

In this table (τ, π) denotes the class of all single transformations other than [ ] or [ ], and θ denotes the class of all transformations. For the second case we also have some lower bounds. Namely, we can prove that the data complexity of θ is not NP ∪ co-NP and its expression complexity is not NEXPTIME ∪ co-NEXPTIME, assuming the standard hypothesis in complexity theory that NP and NEXPTIME are not closed under complement [GJT97]. We also show complexity results for quantifier-free transformation expressions, which we refer to as θ₀.

4.1. Data Complexity

By the data complexity of θ with respect to an expression we mean the complexity of deciding membership in the set

\[ θ₀ = \{ \langle db, kb \rangle \in \mathcal{D} \times \mathcal{K} \subseteq kb : db ∈ \theta(kb) \}. \]

For the case of only one update operation the upper bound for data complexity is:

**Theorem 4.1.** For any θ ∈ (τ, π), θ₀ is in co-NP.

**Proof.** The case where θ is of the form π₁,...,πₙ is obvious.

In the other case θ is of the form τψ₀ for some ψ ∈ Φ. Consider the complement of θ₀, that is, deciding whether db ∉ τψ₀(kb). From the definition of the τ operator, it follows that there are two ways that this can be the case. Either db is not a model of ψ, or for each db₁ ∈ kb there is a db₂ that is a model of ψ, and such that db₁ ∉ db₂.

We guess which of these two cases holds. For the first case the verification can be done in PTIME, since deciding whether a database is a model of a given first-order formula can be done in time polynomial in the size of the database. This is because for the domain of variables B we have to take the constants that appear in either the database or the formula. Hence if we follow equations (4–8) that define models of first-order formulas, at each existential quantifier we have to test at most cardinality of B number of cases, which is linear in the size of the database. This yields a PTIME procedure for any fixed first-order formula.

For the second case, for each db₁ ∈ kb we guess a db₂ and again we verify both of the facts that db₂ is a model of ψ and that db₂ <ₚ db₁. In other words, the symmetric difference between db₁ and db₂ must be less than the symmetric difference between db₁ and db. Since the symmetric difference between db₁ and db is at most the union of the two databases, the size of each db₂ guessed should not be greater than the input size. Hence whether db₂ is a model of ψ can be also decided within PTIME. Finally, checking the condition db₂ <ₚ db can obviously also be carried out in PTIME for each db₂ guessed. Since we need to guess only as many db’s as many databases the kb contains, we could decide in NP whether db ∉ τψ₀(kb).

For unrestricted composite expressions we have the following lower bound characterization.

**Theorem 4.2.** There is a θ in (τ, π)*, such that θ₀ is not in NP ∪ co-NP, unless NP = co-NP.

**Proof.** We will show a particular yes or no reduction from the 3CNF formula satisfiability problem. Let the given 3CNF formula ψ be c₁ ∧ ⋯ ∧ c₈, where each cᵢ is of the form lᵢ₁ ∨ lᵢ₂ ∨ lᵢ₃, with each lᵢᵢ being a literal. We will show that there is an expression of the form π(τψ₀) that gives a yes or no query.

In the reduction we will use relations r₁ (representing the clauses), r₂ (representing a consistent and complete truth assignment), and r₃ (representing the clauses not satisfied by the particular assignment).

Let kb be \{ \langle rᵢ \rangle \}, where rᵢ has a tuple tᵢ for each clause cᵢ of ψ. For instance, if clause cᵢ is x₁ ∨ ¬x₃ ∨ x₈, then tᵢ is \langle i, 1, 1, 5, 0, 8, 1 \rangle. Note that the third, fifth, and the eight elements of the tuple tᵢ denote by value 1 or 0 whether the literal immediately preceding them occurs positively or negatively.

Consider now the following formulas:

\[
 \forall x₁ x₂ x₃ x₄ x₅ x₆ x₇ : R₁ x₁ x₂ x₃ x₄ x₅ x₆ x₇ \\
 \rightarrow ((R₂ x₁ x₅ \lor R₂ x₂ x₃ x₆ x₇) \\
 \land (R₃ x₄ x₁ \lor R₃ x₃ x₈ x₅ x₆ x₇)) \\
 \land (R₄ x₅ x₆ x₇) \\
 \lor R₁ x₁ x₂ x₃ x₄ x₅ x₆ x₇ \\
 \rightarrow (R₁ x₆ x₇ x₃ x₄ x₅ x₆ x₇)
\]

The sentence Ψ is the conjunction of the above two sentences.
Now it can be seen that if $\phi$ is satisfiable, then
$$\pi_i(\tau_\phi(kb)) = \{\langle \mathcal{G} \rangle, \langle \mathcal{G} \rangle \}.2$$
If $\phi$ is not satisfiable, then
$$\pi_i(\tau_\phi(kb)) = \{\langle \mathcal{G} \rangle \}.$$

Suppose that $\mathcal{G}[r_\mathcal{G}]$ is in co-NP. Then it would be possible to solve the 3CNF satisfiability problem in co-NP. That is, we could verify in NP for each $kb$ if there is no solution, by testing whether $\langle \mathcal{G} \rangle \neq \pi_i(\tau_\phi(kb))$. Conversely, suppose that $\mathcal{G}[r_\mathcal{G}]$ is in NP. Then again the 3CNF problem could be solved in co-NP. This time, we would test whether $\langle \mathcal{G} \rangle \in \pi_i(\tau_\phi(kb))$.

We know that 3CNF satisfiability problem is an NP-complete problem. Now the existence of an NP-complete problem that is in co-NP entails NP = co-NP.

For the upper bound of $\mathcal{G}[r_\mathcal{G}]$ we have:

**Lemma 4.1.** For any $\theta$ without $\mathbin{|}$ and $\mathbin{[}$ operators, the set $\mathcal{G}[r_\mathcal{G}]$ is in PSPACE.

**Proof.** WLOG, we assume that $\theta$ is of the form $\tau_\phi(\cdots(\tau_{\phi_1}(\cdots))).$ Let $\theta_{kb}$ be the input knowledgebase, and let $kb_1$, be the knowledgebase after operation $\tau_\phi$ is performed. In particular let $kb_0$ be the output of the whole query, i.e., let $kb_0 = \theta(kb_1)$.

Suppose that for some database $db_n$ we know that $db_0 \in \theta(kb_0)$ and we need to verify that this condition holds. We can do that by finding a $db$ in each knowledgebase $kb_i$, such that $db_i \in \tau_\phi(db_{i-1})$. In other words we must nondeterministically guess a chain of databases from an initial $db_{i-1}$ to $db_i$ and verify that in that chain each successive database $db_i$ is a closest model of $\phi_i$ with respect to the previous database $db_{i-1}$.

To verify that a $db_i \in \tau_\phi(db_{i-1})$ we do the following. First we find the domain $B$ that contains all the constants in either $db_{i-1}$ or $\phi_i$. Then we list all possible tuples that can be constructed from $B$ and the relations in $\theta$. The number of these tuples is polynomial in the size of $B$ and $db_{i-1}$. Note that $db_i$ must be a subset of these tuples, hence its size is polynomial in the size of $db_{i-1}$ and by induction it is also polynomial in the size of $db_0$ and $kb_0$. We have to check against each database $db$ (that is a subset of the listed tuples) that $db \leq_{db_{i-1}} db$. To do that we add a one-bit tag initialized to zero to each tuple. A tag value of zero indicates that the tuple is not part of the database, while a value of one indicates that it is. We also fix some arbitrary ordering of the tags. Taking in that fixed order the tags form a counter. For each database that we get we check in polynomial time (and space) whether it satisfies $\phi_i$. If it does we have to verify that the symmetric difference of $db$ and $db_{i-1}$ is not less than the symmetric difference of $db_i$ and $db_{i-1}$. Clearly that also can be done in PSPACE.

The above gives a NFSPACE procedure for testing whether $db_0 \in \theta(kb_0)$. By Savitch’s theorem, we have that NFSPACE$(n) \subseteq$ PSPACE$(n^2)$. Hence we can also do the test in deterministic PSPACE.

**Theorem 4.3.** For any $\theta \in \Theta$, the set $\mathcal{G}[r_\mathcal{G}]$ is in PSPACE.

**Proof.** Note that the relation symbols in $\theta$ come from a fixed set, say $\{R_1, R_2, ..., R_n\}$, and that $\theta$ and the initial knowledgebase $kb_0$ have a finite set, say $B$, of domain elements in them. Let $\alpha(i)$ be the arity of relation $R_i$, and let $a$ be the maximum arity. Suppose we want to decide whether $db_0$ is in the knowledgebase $\theta(kb_0)$.

When each operation in $\theta$ is applied, the size of each database in the current knowledgebase can never be more that $\ell(B^a)$. This is because there can be at most $B^{\alpha(i)}$ tuples over a domain $B$ in a relation over $R_i$, and we have $n$ relations. Note that $\ell(B^a)$ is only a polynomial in the size of the input database and knowledgebase. Therefore, it is possible to use the technique described in Lemma 4.1 to cycle through all possible databases in PSPACE.

We claim that we can test whether $db_n$ is in $\theta(kb_0)$ within PSPACE for any $\theta$. To prove that we use induction. For the base case we may take any formula for $\theta$ that has in it only $\pi$ and $\tau$ operations. Then by Lemma 4.1 it is possible to verify if $db_0$ is in $\theta(kb_0)$. To check that $db_n$ is in $\theta(kb_0)$, we use the fact that PSPACE is closed under complement.

Now suppose that $\theta$ is of the form $\mathbin{[} \theta \mathbin{]}$ where $\theta'$ has in it only $\pi$ and $\tau$ operations. Then we create each possible tuples using $B$ and the relation symbols in $\theta$. With each of these tuples, we use the technique of Lemma 4.1 again to cycle through all possible databases. As we cycle through the databases, we test using induction, whether it is in $\theta'(kb_0)$.

If yes, then we read out each tuple of the database and as we read each tuple from it, we increment the counter of the corresponding tuples that we created. As we do this we also count the number of databases that are found to be in $\theta'(kb_0)$. Finally, to obtain the output, we simply delete all the counters and tuples we created except those tuples whose counter value at the end of the cycling is the same as the number of databases we found to be in $\theta'(kb_0)$. Clearly this procedure can be done in PSPACE in the size of the domain and the initial input.

It is easy to see that we can evaluate each $\mathbin{[}$-operation similarly to the above and that we can continue to evaluate each successive transformation operation by cycling through again the set of possible databases and calling recursively the evaluation routine for an expression with one less operations in it. Hence we can decide in PSPACE whether $db_n \in \theta(kb_0)$ for any $db_n$ and $kb_0$. 

\[ \text{2 Or } \{\langle \mathcal{G} \rangle\}, \text{ if } \phi \text{ was a tautology.} \]
4.2. Expression Complexity

By the expression complexity of $\Theta$ with respect to a pair $\langle db, kb \rangle \in \mathcal{B} \times \mathcal{A}$ we mean the complexity of deciding membership in the set
\[
\{ \phi \in \Theta : db \in \emptyset(kb) \}.
\]

**Theorem 4.4.** For any $\phi \in \Phi$ and any fixed $\langle db, kb \rangle \in \mathcal{B} \times \mathcal{A}$, we can test in $\text{co-NEXPTIME}$ whether $db \in \tau_s(kb)$.

**Proof.** We follow the same procedure as in Theorem 4.1 to test whether $db \in \tau_s(kb)$. Here the size of $\phi$ is variable. The formula can have at most $|\phi|$ existentials quantifiers, where $|\phi|$ denotes the length of the formula $\phi$. Hence the verification procedure may need to branch $|\phi|$ times in $B$ ways, where $B$ is the domain. (Note that $B$ could vary with the size of $\phi$.) Therefore $O(B^{|\phi|})$ is the worst-case time complexity, which is exponential in the size of $\phi$.

**Theorem 4.5.** There is a $\langle \phi, db, kb \rangle$ such that deciding for arbitrary $\emptyset \in \Theta$ whether $db \in \emptyset(kb)$ is not in $\text{NEXPTIME} \cup \text{co-NEXPTIME}$, unless $\text{NEXPTIME}$ is closed under complement.

**Proof.** In this proof we will simulate a nondeterministic exponential time bounded Turing machine. Let the nondeterministic exponential time Turing machine be $\mathcal{T} = \langle K, \sigma, \delta, s_0 \rangle$, where $K$ is the set of states of the machine, $\sigma$ is the alphabet, $\delta$ is the transition function, and $s_0$ is the initial state. Let the input tape have size $n$, and the computation be bounded by $2^n$ steps. To simplify some of the notation, throughout this proof an overline (as in $\bar{x}$ for example) will denote binary vectors of length $n$.

First we use a $n + 1$-ary relation called $T$ to describe the initial content of the tape. We create $n$ facts $T_i, c_o$, one for each $i \leq n$, where $i$ is given in binary notation as $i$. If $i > n$, then content of the $i$th tape cell will be a special tape symbol # denoting that it is blank. We write a sentence $\forall \langle i \neq 0 \and i \neq 1 \and \ldots \and i \neq n \rangle \rightarrow T_i, \#$, where $0, 1, \ldots, n$ are binary expressions of the numbers from 0 to $n$. Let $\phi_1$ be the conjunction of all the $n$ facts and the sentence. The size of $\phi_1$ is $O(n^2)$.

Second we use a $5$-ary relation called $D$ to describe the transition function $\delta$ of $\mathcal{T}$. We create for each possible machine input state $s_m$, input tape symbol $c_i$, output state $s_{out}$, tape symbol $w$, and movement indicator $m$ (being $n$, 1, or 0) a fact that $Ds_m, c_i, s_{out}, w, m$ if according to $\delta$ when the machine is in state $s_m$ and pointing to $c_i$, then either (1) the machine may go to state $s_{out}$ and point to symbol $w$ after writing it on the tape and the move $m$ is $n$, that is, no move, or (2) the machine may move one tape cell to the left and $m$ is 1, or right and $m$ is 0. (Note that if $m$ is 1 or 0, then $w$ can be any tape symbol; we will not use its value anywhere later in the reduction.) Let $\phi_2$ be the conjunction of all the facts described above. The size of $\phi_2$ is $O(k^2\bar{y})$ where $k$ is the number of machine states and $l$ is the number of different tape symbols.

Third we use a $2n + 1$-ary relation called $C$ to describe the configuration of the machine. The relation $C_i, t, s$ describes that at time step $t$ the machine is in state $s$ and is pointing to tape position $i$. We can assume that the Turning machine is pointing at time zero to the first tape cell. Therefore we create a fact $C_0, 0, 1, s_0$. Let $\phi_3$ denote this single fact. The size of $\phi_3$ is $O(n)$.

Fourth we use the $2n + 1$-ary relation $R$ to denote the sequence of nondeterministic transitions of the machine. We will express the sequence of transitions of the machine by relation $R_{a_1}, \ldots, a_{n+1}, \ldots, a_{2n}, c$ in such a way that the relation records the fact that at time $i$, encoded by the binary sequence $a_1, \ldots, a_s$, the $j$th tape cell, where $j$ is encoded by the binary sequence $a_{s+1}, \ldots, a_{2n}$, contains the tape symbol $c$. To initialize $R$ we write a sentence: $\forall x_0, x_1, \ldots, x_n, i, 0, 1, \ldots, n \rightarrow R_{i, x_0, x_1, \ldots, x_n, i, 0, 1, \ldots, n}$. Let $\phi_4$ denote this sentence. The size of $\phi_4$ is $O(n)$. (Here $\phi_4$ is only for the initialization, but the expression of $R$ will continue in $\phi_5$ below.)

Fifth we use the $2n$-ary relation $S$ to describe the successor function limited to binary numbers of size $n$ bits. That is, we want $Si, \bar{a}$ to be true if and only if the binary number $\bar{a}$ is the successor of $i$, in shorthand $i + 1 = \bar{a}$. The successor function can be expressed by a sentence of size $O(n)$ as described in [FR79]. Similarly, we use the $2n + 1$-ary relation $M$ to describe the next tape position after the machine moves one tape cell in direction $m$. That is we want $Mi, t, n$ be true for each $0 \leq t \leq 2^n$, and we want $Mi, \bar{a}, \bar{t}$ to be true for each $0 \leq t \leq 2^n$ and $\bar{a}$ successor of $i$, and we want $M(i, \bar{a}, 1)$ be true for each $0 \leq t \leq 2^n$ and $\bar{a}$ successor of $\bar{i}$. This can be expressed using the successor function by a sentence. Let the conjunction of the two sentence be $\phi_5$. The size of $\phi_5$ is $O(n)$.

Sixth we write a sentence $\phi_6$ that expresses the requirements for a valid nondeterministic computation of the machine:

$$\forall l, i, p, s_m, s_{out}, c_i, w, m : \quad Cli + 1, i, s_{out} \land R_i + 1, i, w \leftrightarrow m = n \land Cli, i, s_m \land R_i, i, c_i \land Ds_m, c_i, s_{out}, w, m \land$$

$$\land Cl_1 + 1, \bar{a}, s_{out} \leftrightarrow Mi, \bar{a}, m \land m \neq n \land Cli, i, s_m \land R_i, i, c_i \land Ds_m, c_i, s_{out}, w, m \land$$

$$\land Ri + 1, \bar{p}, c_p \leftrightarrow R_i, \bar{p}, c_p \land Cli, i, s_m \land (i \neq \bar{p} \lor Ds_m, c_i, s_{out}, w, n).$$
This sentence says that if in time $t$ the machine $T$ is pointing to the $r$th position and is in state $s_a$, and the content of the $r$th cell is $c_t$, and the transition specifies either a write or a move, then the configuration and the tape contents in the next time step will be as expected. The third part of the sentence says that the tape symbol never changes in any position (even at the current position) unless it is explicitly overwritten by the machine. The size of $\phi_t$ is $O(n)$.

Seventh we write a sentence $\phi_t$ that asserts that at time $2^n$ the machine is in the halting state $h$. The sentence will be $\exists \phi (C_1 \wedge \ldots \wedge C_l \wedge h)$. Thus all valid relations for $R$ are restricted to those that lead to an accepting configuration. The size of $\phi_t$ is $O(n)$.

Let $\theta_1$ be the transformation $\tau_{\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5}$, and let $\theta_2$ be the transformation $\tau_{\phi_6 \wedge \phi_7}$. Applying $\theta_1$ to an empty initial knowledgebase will create the relations $T, D, S, M$ and initialize $C$ and $R$ as required.

Let $\theta_i$ be the transformation that copies the four relations $T, D, S, M$ into a set of four new relations that do not occur within either $\theta_1$ or $\theta_2$. We can express $\theta_1$ by a sentence of length $O(n)$. Apply $\theta_1$ after $\theta_1$.

By the minimization requirement and definition (9), whenever it is possible applying $\theta_2$ after $\theta_1$ will result in a valid $R$ and $T, D, S, M$ and the initialization of $R$ and $C$ unchanged, otherwise either one or more of these four relations will change or the initial value of $R$ or $C$ will change.

Similarly to Example 5, after performing $\theta_2$ we can use another transformation $\theta_4$ that makes a binary output relation $r_0$ the empty relation if and only if there are no changes to $T, D, S, M$ and the initialization of $R$ and $C$, otherwise $r_0$ will be the relation with the empty tuple. We can write $\theta_4$ such that it projects out any other relations beside $r_0$, so that only $r_0$ remains. The size of $\theta_4$ is $O(n)$.

Let $\theta_5$ denote the complete transformation expression, that is, $\theta_5(\theta_2(\theta_1(t))))$. The size of $\theta_5$ is $O(n^2 + k^2 t^2)$.

Suppose then that the language $\mathcal{L}_{\langle \phi, kb \rangle}$ is in $\text{NEXPTIME}$ for any $kb$ and every database $\phi$. We fix $kb_0$ to be the empty knowledgebase. Let $db_0$ be the database with the only relation $r_0$ and $r_0$ being the empty relation. Then using the transformation $\theta_5$, we could decide in $\text{NEXPTIME}$ in the size of $\theta_5$, whether $db_0 \in \mathcal{L}_{\langle \phi, kb_0 \rangle}$. Now let $db_1$ be the database that contains the only relation relation $r_0$, and $r_0$ contains the empty tuple. We could now decide in $\text{NEXPTIME}$ whether $db_1 \in \mathcal{L}_{\langle \phi, kb \rangle}$.

Note that we can describe by any fixed $\text{NEXPTIME}$ bounded Turing machine and any variable input string of length $n$ by some $\theta_1$ of length polynomial in $n$. Therefore, if the language $\mathcal{L}_{\langle \phi, kb \rangle}$ is in $\text{NEXPTIME}$ for every $db$ and every $kb$, then the question of whether the input tape is accepted by an $\text{NEXPTIME}$ bounded Turing machine is in $\text{co-NEXPTIME}$. This in turn implies that $\text{NEXPTIME}$ is closed under complement. Hence unless $\text{NEXPTIME}$ is indeed closed by complement, the language $\mathcal{L}_{\langle \phi, kb \rangle}$ is not in $\text{NEXPTIME}$. We can argue similarly that the language is also not in $\text{co-NEXPTIME}$ using the fact that we have an if and only if transformation.

For the upperbound we have in general that:

**Theorem 4.6.** For any $\langle db, kb \rangle \in \mathcal{L} \times \mathcal{L} \times \mathcal{L}$, the set $\mathcal{L}_{\langle db, kb \rangle}$ is in $\text{EXPSPACE}$.

**Proof.** The maximum size of any database in the current knowledgebase during testing will be bounded as in Theorem 4.3. In the present case the domain $B$ is fixed, but the maximum arity $a$ and the number of relations $n$ in the databases are variable. Hence the expression complexity will be the $\text{EXPSPACE}$, and we can show that by using the same algorithm as in Theorem 4.3 together with the fact that $\text{EXPSPACE}$ is closed under complement.

### 4.3. Some Special Cases

In this section we consider two special cases of the transformation language: (1) quantifier-free transformations and (2) Datalog-restricted transformations. By quantifier-free transformations we mean expressions in which all sentences used to update the knowledgebase are boolean combinations of ground atomic formulas, i.e., formulas in which each argument of each relation is a constant. By Datalog-restricted transformations we mean transformation expressions in which all sentences are conjunctions of function-free Horn clauses.

**Theorem 4.7.** If $\theta$ is quantifier free, then $\mathcal{L}_{\langle \theta, \phi \rangle}$ is in $\text{PTIME}$.

**Proof.** Since each sentence of $\theta$ is quantifier free, we need to make a fixed number of transformations of the form $\bigcup$, $\bigcap$, or $\tau_\phi$ where $\phi$ is a quantifier free sentence. Transformations $\bigcup$ and $\bigcap$ can clearly be done in linear time in the size of the knowledgebase using the same procedure as in Theorem 4.3. For performing $\tau_\phi$ we have to test whether $\phi$ contains a minimal models. Each of these models will be a database which is the union of the ground facts that occur only in the input database and some ground atoms in $\phi$ assigned to be either false (i.e., not occur in the corresponding relation) or true (i.e., occur in the corresponding relation). Since the number of programs that need to be considered is fixed and bounded by the size of $\phi$, all possible minimal databases can be found and tested whether they are models of $\phi$ in $\text{PTIME}$.

For the data complexity of transformations with the second restriction we have:

**Theorem 4.8.** If $\theta$ is a Datalog-restricted transformation, then $\mathcal{L}_{\langle \theta, \phi \rangle}$ is in $\text{PTIME}$.

**Proof.** Here we need to make a fixed number of transformations of the form $\bigcup$, $\bigcap$, or $\tau_\phi$ where $\phi$ is a Datalog program. Again, transformations $\bigcup$ and $\bigcap$ can clearly be done in linear time in the size of the knowledgebase. For
showing that performing \( r_e \) can be done in \( \text{PTIME} \) it is enough to recall that Datalog programs have a unique least model that can be computed using naive evaluation in \( \text{PTIME} \).

For the expression complexity, when \( \theta \) is quantifier free the transformation language has the following bound:

**Theorem 4.9.** \( \theta(r_0, kb) \) is not in \( \text{NP} \cup \co\text{NP} \) unless \( \text{NP} \) is closed under complement, even if \( \theta \in \Theta_q \) and is quantifier free.

**Proof.** This follows by a reduction from the problem of satisfiability of propositional formulas [GJ79]. Take the case when \( db \) and \( kb \) both contain a single zero-ary relation \( r_0 \) that is assigned to be true (i.e., contains the empty tuple). Any propositional formula can be expressed by a sentence \( \phi' \) using zero-ary relation symbols different from \( R_0 \). Then let \( \phi \) be the sentence \( \phi' \rightarrow R_0 \).

Since \( r_0 \) is an input relation, \( r_0 \) will not be changed unless necessary, which occurs if and only if \( \phi' \) has no model. Hence the formula \( \phi' \) is satisfiable if and only if after the projection \( \pi_{\phi} \), the relation \( r_0 \) still contains the empty tuple, i.e., \( db \in \pi_{\phi} r_0 \ kb \) as required. Similarly to Theorem 4.5 this leads to the conclusion that the problem is not in \( \text{NP} \cup \co\text{NP} \) unless \( \text{NP} \) is closed under complement.

**Remark.** Grahne and Mendelzon [GM95] have studied the complexity of evaluating subjunctive queries in a propositional language, and found the data complexity of \( \text{co-NP} \) unless \( GM95 \).

revision and updates we refer the reader to \([\text{EG982, EG93, GM95}]\). For other complexity theoretic issues in belief in it, otherwise it would be a proper superset of the language language does not have subjunctive implication operators \( PSPACE \).

The transformation language has the following bound:

\[
\boxed{\text{Theorem 5.1. } SF \subseteq ST'.}
\]

**Proof.** Without loss of generality let \( \bar{x} \exists n_{n+1} \phi(\bar{x}) \) be any existential second order query. Let \( a \) be the arity of \( R_{n+1} \) and let the set of relation symbols in \( \phi \) be \( \{ R_1, \ldots, R_n, R_{n+1} \} \). (Note that \( a \) is also the size of \( \bar{x} \). The size of \( \bar{x} \) could be smaller if we added extra projection operations.) By the standard definition [Var82] the image of this query under a database \( \langle D, r_1, \ldots, r_n \rangle \) is \( \{ d \in D^n \mid \exists a \text{ such that } \phi(d) \text{ is true in } \langle D, r_1, \ldots, r_n \rangle \} \).

Instead of a fixed domain as in [Var82], we take \( D \) to be the set of constants in \( \phi \) and in the input relations \( r_1, \ldots, r_n \). Therefore the possible values of \( r_{n+1} \) are finite and can be listed. There would be exactly \( 2^{|\phi|} \) number of possibilities for \( r_{n+1} \). Therefore we can construct a knowledgebase \( kb \) that has in it exactly that many databases with each database containing \( r_1, \ldots, r_n \) and one of the possible \( r_{n+1} \).

Then the query can be expressed as a transformation as follows. We create a new relation \( r_{n+2} \) to present the output of the query. Then we write

\[
\pi_{n+2} \sqcup \tau_{\phi}(r_{n+2}(\bar{x})) \quad (kb).
\]

This transformation always takes in set of databases, with each database having schema \( \{ R_1, \ldots, R_{n+1} \} \). The input database is never changed because the implication \( \forall \bar{x} \phi(\bar{x}) \rightarrow R_{n+2}(\bar{x}) \) can always be satisfied by adding the required tuples for \( \bar{x} \) to \( r_{n+2} \). By the minimality requirement, \( r_{n+2} \) will always have only those tuples in it which satisfy \( \phi(\bar{x}) \). If no tuples satisfy \( \phi(\bar{x}) \), then \( r_{n+2} \) will be an empty relation. Also by the minimality requirement, none of the other relations are changed. The projection \( \pi_{n+2} \) simply returns the desired output, that is, the set of tuples that satisfy the existential second order formula, for some value of \( r_{n+1} \).

By Theorem 5.1 any \( SF \) query can be expressed by a transformation expression of the form \( \pi \sqcup \tau \). An interesting 5. EXPRESSIVE POWER

Let \( YF \), \( SF \), and \( SO \) be the class of all transformations from databases to databases expressible, respectively, by fixpoint queries, existential second-order queries, and second-order queries, as defined in [CH82, Var82]. It is well-known that \( YF \) is properly included in \( SF \) and that \( SF \) is also included in \( SO \) [Var82].

In order to relate our language to the above classes of transformations, we shall restrict ourselves to the case where all input knowledgebases are singletons, i.e., databases, and we will restrict the language so that all output knowledgebases are also singletons. From this restriction follows that the expressive power results that are lower-bounds in this section will carry over to the general case, but the significance of the upper bounds is primarily in the comparison with other languages. Let \( ST \) be the class of all transformations from singleton knowledgebases to singleton knowledgebases, expressible by a transformation in \( \Theta \) of the form \( (\pi b)^{n} \), where each \( b \) is one of \( \sqcup \) or \( \sqcap \).

As we noted in Section 2, the transformations described by expressions in this class fall within the class of determinist updates defined by Abiteboul and Vianu. It follows immediately that every query in \( ST \) is expressible in their languages \( \text{detTL} \) and \( \text{detDL} \), which are shown in [AV88] to express all determinist updates. It follows from Theorem 5.2 below that this inclusion is proper, since \( ST \) does not go beyond the second order queries \( SO \).

However, \( ST \) does include all the existential second-order queries; let \( ST' \) denote the subset of \( ST \) expressions that use at most \( i \) compositions of subexpressions of the form \( \pi b \). Then all existential second-order queries can be expressed within \( ST' \).

**Theorem 5.1.** \( SF \subseteq ST' \).

**Proof.** Without loss of generality let \( \exists n_{n+1} \phi(\bar{x}) \) be any existential second order query. Let \( a \) be the arity of \( R_{n+1} \) and let the set of relation symbols in \( \phi \) be \( \{ R_1, \ldots, R_n, R_{n+1} \} \). (Note that \( a \) is also the size of \( \bar{x} \). The size of \( \bar{x} \) could be smaller if we added extra projection operations.) By the standard definition [Var82] the image of this query under a database \( \langle D, r_1, \ldots, r_n \rangle \) is \( \{ d \in D^n \mid \exists a \text{ such that } \phi(d) \text{ is true in } \langle D, r_1, \ldots, r_n \rangle \} \).

Instead of a fixed domain as in [Var82], we take \( D \) to be the set of constants in \( \phi \) and in the input relations \( r_1, \ldots, r_n \). Therefore the possible values of \( r_{n+1} \) are finite and can be listed. There would be exactly \( 2^{|\phi|} \) number of possibilities for \( r_{n+1} \). Therefore we can construct a knowledgebase \( kb \) that has in it exactly that many databases with each database containing \( r_1, \ldots, r_n \) and one of the possible \( r_{n+1} \).

Then the query can be expressed as a transformation as follows. We create a new relation \( r_{n+2} \) to present the output of the query. Then we write

\[
\pi_{n+2} \sqcup \tau_{\phi}(r_{n+2}(\bar{x})) \quad (kb).
\]

This transformation always takes in set of databases, with each database having schema \( \{ R_1, \ldots, R_{n+1} \} \). The input database is never changed because the implication \( \forall \bar{x} \phi(\bar{x}) \rightarrow R_{n+2}(\bar{x}) \) can always be satisfied by adding the required tuples for \( \bar{x} \) to \( r_{n+2} \). By the minimality requirement, \( r_{n+2} \) will always have only those tuples in it which satisfy \( \phi(\bar{x}) \). If no tuples satisfy \( \phi(\bar{x}) \), then \( r_{n+2} \) will be an empty relation. Also by the minimality requirement, none of the other relations are changed. The projection \( \pi_{n+2} \) simply returns the desired output, that is, the set of tuples that satisfy the existential second order formula, for some value of \( r_{n+1} \).

By Theorem 5.1 any \( SF \) query can be expressed by a transformation expression of the form \( \pi \sqcup \tau \). An interesting
question is what can be gained in expressive power by the composition of πβτ transformations. The following theorem gives a partial answer to that question, namely that the composition cannot increase the expressive power beyond SO.

**Theorem 5.2.** ST ⊆ SO.

**Proof.** Clearly the theorem is proved by demonstrating there is a logspace reduction from ST to an equivalent transformation in SO.

First we consider the case where θ is of the form πa ⊔ τφ, where j ∈ {1, ..., n}, σ = {db} = {Ri, ..., Rj}, and {Ri} ⊆ σ(φ) ⊆ {Ri, ..., Rj}. In the second-order syntax we therefore sometimes write φ as φ(Ri, ..., Rj).

Suppose that the arity of Rj is k, and for each i let Ri and Si be rational variables of the same arity as Rj. The corresponding second-order query would then be

\[ x_1, x_2, ..., x_k, \exists R_{i_1} \ldots \exists R_{i_m} \forall S_{i_1} \ldots \forall S_{i_n} \phi(x_1, x_2, ..., x_k, R_{i_1}, ..., R_{i_m}, S_{i_1}, ..., S_{i_n}) \]

where φ is the formula

\[ R_{i_1}(x_1, x_2, ..., x_k) \land \phi(R_{i_1}, ..., R_{i_m}) \land \min(φ, R_{i_1}, ..., R_{i_m}, S_{i_1}, ..., S_{i_n}) \]

Here min(φ, ...) is an abbreviation of the formula

\[ \left( φ(S_{i_1}, ..., S_{i_n}) \land \bigwedge_{j=1}^n \left( S_{i_j} \subseteq R_{i_j} R_{i_j}' \right) \right) \to \left( \bigwedge_{j=1}^n \left( R_{i_j}' \subseteq R_{i_j} S_{i_j} \right) \right) \]

where Si ⊆ Rj Rj' abbreviates

\[ \forall x_{i_1} \ldots x_{i_n} : \left( S_{i_1} x_{i_1} x_{i_2} \ldots x_{i_n} \land \neg R_{i_1} x_{i_1} x_{i_2} \ldots x_{i_n} \right) \lor \left( R_{i_1} x_{i_1} x_{i_2} \ldots x_{i_n} \land \neg S_{i_1} x_{i_1} x_{i_2} \ldots x_{i_n} \right) \to \left( R_{i_1}' x_{i_1} x_{i_2} \ldots x_{i_n} \land \neg R_{i_1} x_{i_1} x_{i_2} \ldots x_{i_n} \right) \lor \left( R_{i_1} x_{i_1} x_{i_2} \ldots x_{i_n} \land \neg S_{i_1} x_{i_1} x_{i_2} \ldots x_{i_n} \right) \]

and likewise for Rj' ⊆ Rj Sj.

It can now be verified that for any db ∈ Θ, the transformation expression θ and the above second-order query return the same result when applied to {db}.

If θ is of the form πa ⊓ τφ, then the corresponding second-order query is

\[ x_1, x_2, ..., x_k, \forall R_{i_1}' \ldots \forall R_{i_m}' : \left( \phi(R_{i_1}', ..., R_{i_m}') \land \min(φ, ...) \right) \to R_{i_1}' x_1 x_2 \ldots x_k \]

Note the similarity between these second-order queries and circumstance [McC80].

To proceed, if θ is of the form πa ⊓ τφ where σ(db) = {Ri, ..., Rj}, and {Ri} ⊆ σ(φ) ⊆ {Ri, ..., Rj}, then we apply a slight variation of the basic reduction including the necessary modification to the formula Sj ⊆ Rj Rj' (cf. definitions (1)-(3)).

Furthermore, if the projection in θ is on several component relations, we define a vector of second-order queries.

Finally, if θ is a composition of several πβτ-expressions, the corresponding second-order query will be obtained by composing the more elementary queries.

Since SO = QPHIER, the existence of a logspace reduction for unrestricted expressions in θ to queries in SO would place the θ transformations within the polynomial hierarchy.

Abiteboul, Simon and Vianu [ASV90] have studied restrictions of the update languages that we mentioned at the beginning of this section and characterized their expressive power in terms of complexity classes. Their approach is to transform non-deterministic languages into deterministic ones by either taking the union of all the possible output databases computed for each input database (which they call the possibility semantics) or taking the intersection (certainty semantics). Since the \| operator corresponds naturally to possibility semantics and \( to certainty, it seems that the languages of [ASV90] should be closely connected to various subclasses of ST transformations, but we have not yet explored this in detail.

6. CONCLUSIONS AND OPEN PROBLEM

We propose in this paper a simple and versatile language that unifies queries and updates. There are a few other proposals in that direction, but our language has to its advantage that its basic operator for update satisfies all the Katsuno–Mendelson postulates, which capture intuitive requirements on the notion of update.

Our work leaves open the precise computational complexity and expressive power of the transformation language. Our conjecture is that if we restrict to a constant the number of nested transformations in Lemma 4.1 then we are likely to end up in the polynomial hierarchy. It would be also good to get tighter lower and upper bounds for the data complexity of transformation expressions that we know lies between the upper bound of PSPACE and the lower bound of not in NP ∪ co-NP. To tighten this and other bounds would require a deeper understanding of the nature of the \( and \| operators which are unique to this paper.

Another interesting direction of research would be to investigate the relationship between hypothetical queries and the proposed transformation language. Finally, it
would be challenging to look for suitable specific application areas, and perhaps tailor the transformation language to those by adding application-specific operators.

REFERENCES


