Knowledgebase Transformations

Gösta Grahne, Alberto O. Mendelzon and Peter Z. Revesz

Abstract

We propose a language that expresses uniformly queries and updates on knowledgebases consisting of finite sets of relational structures. The language contains an operator that "inserts" arbitrary first-order sentences into a knowledgebase. The semantics of the insertion is based on the notion of update formalized by Katsuno and Mendelzon in the context of belief revision theory. Our language can express, among other things, hypothetical queries and queries on recursively indefinite databases. The expressive power of our language lies between existential second-order and general second-order queries. The data complexity is in general within polynomial space, although it can be lowered to co-NP and to polynomial time by restricting the form of queries and updates.

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—for example, if one allows views in addition to stored relations—it becomes difficult to give meaning to updates [BS81, FUV83, FKU86]. 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

\footnotesize
*This work was supported by the Institute for Robotics and Intelligent Systems and the Natural Sciences and Engineering Council of Canada. A preliminary version of this paper appears in [GMR92].

\(^1\)Department of Computer Science, University of Helsinki, Teollisuuskatu 23, SF-00510 Helsinki, grahne@cs.helsinki.fi. Work partially performed while visiting the Department of Computer Science, University of Toronto.

\(^2\)Department of Computer Science, University of Toronto, Toronto, Canada, M5S 1A4, mendel@db.toronto.edu

\(^3\)Department of Computer Science, University of Nebraska-Lincoln, Lincoln, NE 68588, revesz@cse.unl.edu Work partially performed while visiting the Department of Computer Science, University of Toronto.
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 \( w \) “\( W \) has landed.” After receiving the message, the knowledgebase \( T \) is the theory \( \{(v \land \neg w) \lor (\neg v \land w)\} \). 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 \( \mathcal{M} \) of the theory to be changed, find the set of models of the sentence to be inserted that are “closest” to \( \mathcal{M} \). 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 AC 902” 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) 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 $\Delta$ denotes the symmetric set difference operation, that is, $A \Delta B = (A \setminus B) \cup (B \setminus A)$.

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), $\exists$ (existential quantifier), $=$ (equality), and the parenthesis symbols.
With each relation symbol \( R_i \in R \) we associate the \textit{arity} \( \alpha(i) \). A \textit{k-ary term} is a tuple with \( k \) components, each in \( A \cup X \). An \textit{atomic formula} is an expression of the form \( R_i(x) \) where \( R_i \) is in \( R \) and \( x \) is an \( \alpha(i) \)-ary term, or an expression of the form \( x_i = x_j \), or an expression of the form \( x_i = a_j \), where \( \{x_i, x_j\} \subseteq X \), and \( a_i \in A \).

The set of all \textit{well formed formulas} of \( \mathcal{L} \) is defined in the usual way, and it is denoted \( \Phi' \). The subset of \textit{sentences} in \( \Phi' \) is denoted \( \Phi \). 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 \( r_{ij} \) is a finite subset of \( A^{\alpha(ij)} \). 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 \( \mathcal{DB} \). By \( \mathcal{DB}_\sigma \) we mean the set of all databases on schema \( \sigma \). Furthermore, if \( B \) is a subset of the domain \( A \), then \( \mathcal{DB}^B_\sigma \) denotes the set of all databases on schema \( \sigma \) containing only values in \( B \).

Let \( db_1 \) and \( db_2 \) be databases. Then we say that \( \sigma(db_2) \) \textit{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 \( \leq \) 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 \( \sigma(db) \) \textit{dominates} \( \sigma(db) \).

Then define a relation \( \leq_{db} \) over \( \mathcal{DB}_{\sigma(db_1)} \), such that \( db_1 \leq_{db} db_2 \) if and only if, either:

\[
s_{ij} \oplus r_{ij} \subseteq u_{ij} \oplus r_{ij},
\]

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

\[
s_{ij} \oplus r_{ij} = u_{ij} \oplus r_{ij},
\]

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

\[
s_{ik} \oplus \emptyset \subseteq u_{ik} \oplus \emptyset,
\]

hold for the rest of the relations \( s_{ik} \) and \( u_{ik} \) in \( db_1 \) and \( db_2 \). \( \Box \)

We give an example of partial order. Let \( db_1 = \langle \{R(a_1a_2), S(a_1a_4)\} \rangle \) and \( db_2 = \langle \{R(a_1a_2), S(a_1a_4), S(a_2a_3)\} \rangle \) and \( db = \langle \{R(a_1a_2)\} \rangle \).\(^1\) Here \( \sigma(db_1) = \sigma(db_2) = \{R,S\} \) and \( \sigma(db) = \{R\} \). Since all three relations agree in \( R \) and \( S \) in \( db_2 \) is a strict subset of \( S \) in \( db_2 \), we have that \( db_1 \leq_{db} db_2 \).

\(^1\)From this example, it can be easily seen that \( \leq_{db} \) is a partial order.

This partial order compares the databases in \( \mathcal{DB}_{\sigma(db_1)} \) with respect to their \textit{closeness} to the database \( db \). If \( \sigma(db) = \sigma(db_1) = \sigma(db_2) \), then we have \( db_1 \leq_{db} db_2 \) if and only if the

---

\(^1\)In this example, we indicated which tuple belongs to which relation by explicitly writing out “R” and “S”, in some examples later when the schema is a singleton we will drop the relation symbol.
symmetric difference between \( db_1 \) and \( db \) is included in the symmetric difference between \( db_2 \) and \( db \), 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 \( r \oplus r = \emptyset \), 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 (1)). If two databases have the same changes to these relations (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 \( r \oplus \emptyset = r \), 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 \( KB \).

The interpretation of a sentence \( \phi \in \Phi \) \( w.r.t. \) a database \( db \) is a relation \( \models \) on \( DB \times \Phi \) defined for \( db \) and \( \phi \) if and only if \( \sigma(db) \) dominates \( \sigma(\phi) \), in which case the recursive definition is:

\[
\begin{align*}
\text{db} \models (a_i = a_j) & \iff i = j \quad (4) \\
\text{db} \models R_i(x) & \iff x \in r_i \quad (5) \\
\text{db} \models (\phi \land \psi) & \iff \text{db} \models \phi \text{ and } \text{db} \models \psi \quad (6) \\
\text{db} \models (\neg \phi) & \iff \text{not db} \models \phi \quad (7) \\
\text{db} \models (\exists x_i \phi) & \iff \text{db} \models \phi(x_i/a_j) \text{ for some } a_j \in A \quad (8)
\end{align*}
\]

By \( [\phi] \) we mean the set of all databases such that \( \models (db, \phi) \) 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 \( [\phi] \cap DB^B_s \).

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

Now consider the function \( \mu : \Phi \times DB \rightarrow KB \), defined as

\[
\mu(\phi, db) = \{db' \in DB^B_s : db' \text{ is } \leq_{ab} \text{-minimal in } [\phi]^B_s \}, \quad (9)
\]

where \( B \) is the smallest subset of the domain \( A \), such that \( B \) contains all values that appear in \( db \) and \( \phi \), and \( s \) is the schema \( \sigma(db) \cup \sigma(\phi) \).

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 \( \tau : \Phi \times KB \rightarrow KB \), \( \sqcap : KB \rightarrow KB \), \( \sqcup : KB \rightarrow KB \), and \( \pi_{i_1, \ldots, i_k} : KB \rightarrow KB \), where

\[
\tau_\phi(kb) = \bigcup_{db \in kb} \mu(\phi, db), \quad (10)
\]

and \( \sqcap(kb) \) is the componentwise intersection of the databases in \( kb \), i.e. the \( glb \) of \( kb \) \( w.r.t. \) to the Cartesian generalization of subset inclusion. Likewise, \( \sqcup(kb) \) is the componentwise
union or lub. As an example, let $kb = \{\{a_1 a_2, a_1 a_4\}, \{a_1 a_4, a_2 a_3\}\}$. Here the knowledgebase contains two databases with the same schema. The value of $\cap(kb)$ is $\{\{a_1 a_4\}\}$ and the value of $\cup(kb)$ is $\{\{a_1 a_2, a_2 a_3, a_1 a_4\}\}$. 

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

The function $\tau$ can be seen as an update function, in that $\tau_\phi(kb)$ “inserts” $\phi$ into $kb$, producing a new $kb$. In the space example in Section 1, the knowledgebase $T$ would be $kb = \{\{v\}, \{w\}\}$, where $v$ is a tuple corresponding to the proposition “$V$ has landed,” and likewise for $w$. Let these tuples be over schema $R_1$. When we learn that $V$ has landed we perform the update

$$\tau_{R_1(v)}(kb) = \{\{v\}, \{v, w\}\}.$$ 

**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 to express the regular projection operation onto the first column of $R$. This can be accomplished by inserting the sentence

$$\pi_{R_1} \tau_{R_1(x) - R(x,y)}(kb).$$

The output relation $R_1$ 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, corresponding to the KM-postulates [KM91a] for update, hold.

**Theorem 2.1** For all schema-wise appropriate $db$, $kb$, $kb_1$ and $kb_2$:

(i). $\tau_\phi(kb) \subseteq [\phi]$

(ii). If $kb \subseteq [\phi]$, then $\tau_\phi(kb) = kb$

(iii). If $kb \neq \emptyset$ and $[\phi]^B \neq \emptyset$, then $\tau_\phi(kb) \neq \emptyset$

(iv). If $[\phi] = [\psi]$, then $\tau_\phi(kb) = \tau_\psi(kb)$

(v). $\tau_\phi(kb) \cap [\psi] \subseteq \tau_{\phi \land \psi}(kb)$

(vi). If $\tau_\phi(kb) \subseteq [\psi]$, and $\tau_\psi(kb) \subseteq [\phi]$, then $\tau_\phi(kb) = \tau_\psi(kb)$

(vii). $\tau_\phi(\{db\}) \cap \tau_\psi(\{db\}) \subseteq \tau_{\phi \vee \psi}(\{db\})$

(viii). $\tau_\phi(kb_1 \cup kb_2) = \tau_\phi(kb_1) \cup \tau_\phi(kb_2)$

**Proof.** (i). Let $db \in \tau_\phi(kb)$. From (9) it follows that $db \in [\phi]^B$. Since $[\phi]^B \subseteq [\phi]$, the claim follows.

(ii). Let $db' \in \tau_\phi(kb)$. Then there is a $db \in kb$, such that $db'$ is $\leq_{db}$-minimal in $[\phi]^B$. Since $kb \subseteq [\phi]$, we have $db \in [\phi]^B$. Now $db$ is the unique $\leq_{db}$-minimal element in $[\phi]^B$. Thus $db' = db$, and consequently $\tau_\phi(kb) \subseteq kb$.

For inclusion in the other direction, let $db \in kb$. Since $kb \subseteq [\phi]$, $db$ is the unique $\leq_{db}$-minimal element in $[\phi]^B$. Therefore $db \in \tau_\phi(kb)$.

(iii), (iv), and (viii) are immediate consequences of definitions (9) and (10).
For (v), let $db' \in \tau_\phi(kb) \cap [\psi] \neq \emptyset$. Then there is a $db \in kb$, such that $db' \in \mu(\phi, db)$, and $db' \in [\phi \land \psi]_B^{\tau_\phi(kb) \cup (\sigma(\phi))} \land B$, where $B$ 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 $db'' \leq db$, $db'' \neq db'$, and $db'' \in [\phi \land \psi]_B^{\tau_\phi(kb) \cup (\sigma(\phi))}$. Here $B'$ is the set of values appearing in $db$, $\phi$, or $\psi$. It now follows that $db''$ also is in $[\phi]_B^{\tau_\phi(kb) \cup (\sigma(\phi))}$. Thus we have a contradiction to the fact that $db' \in \mu(\phi, db)$.

For (vi), we show that, given the assumptions, $\tau_\phi(kb) \subseteq \tau_\psi(kb)$. Let $db' \in \tau_\phi(kb)$. Then there is a $db \in kb$, such that $db' \in \mu(\phi, db)$. We claim that $db' \in \mu(\psi, db)$. 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 $db' \in [\psi]_B^{\tau_\phi(kb) \cup (\sigma(\phi))}$, the set $\mu(\psi, db)$ must be nonempty. Let therefore $db'' \in \mu(\psi, db)$. Then it must be that $db'' \leq db$, and $db'' \neq db'$. But this is not possible, since $db' \in [\phi]_B^{\tau_\phi(kb) \cup (\sigma(\phi))}$, thus yielding a contradiction to the fact that $db' \in \mu(\phi, db)$. The argument for inclusion in the other direction is symmetrical.

For (vi), 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 $\mu(\phi \lor \psi, db) \neq \emptyset$. If $db' \notin \mu(\phi \lor \psi, db)$, there must be an $db'' \in \mu(\phi \lor \psi, db)$, such that $db'' \leq db$, $db'' \neq db'$. It now follows that $db''$ is in $[\phi]_B^{\tau_\phi(kb) \cup (\sigma(\phi))}$, or in $[\psi]_B^{\tau_\phi(kb) \cup (\sigma(\psi))}$. In the first case we have a contradiction to the fact that $db' \in \mu(\phi, db)$, 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 update does not commute with glb and lub.

**Lemma 2.1** There are knowledgebases and sentences such that $\cap(\tau_\phi(kb)) \neq \tau_\phi(\cap(kb))$, and $\cup(\tau_\phi(kb)) \neq \tau_\phi(\cup(kb))$.

**Proof.** Let $kb = \{\langle\{a_1a_2a_3\}, \{a_1a_2a_4\}\rangle\}$, that is, two databases, each with a single tuple over schema, say, $R_1$. Then the value of $\cap(\tau_{\forall a_1a_2}R_{a_1a_2}(kb))$ is $\{\emptyset, \{a_1\}\}$, where the second relation is on schema $R_2$. On the other hand, commuting $\cap$ and $\tau$ produces $\{\emptyset, \emptyset\}$.

For the second part of the lemma, let $kb = \{\{a_1a_2\}, \{a_2a_3\}\}$, with schema $R_3$. Then the value of $\tau_{\forall a_1a_2}R_{a_1a_2}(\cup(kb))$ is $\{\{a_1a_2a_2a_3\}, \{a_1a_2a_2a_3\}\}$, while commuting the operators results in $\{\{a_1a_2a_2a_3\}, \{a_1a_2a_2a_3\}\}$.

By composing the transformation functions in the obvious way, and by using schemas as parameters, transformation expressions can be formed. For example, $\pi_1(\tau_\psi(\cap(\tau_\phi(R_1, R_2))))$ is a transformation expression. The value of this expression, when applied to a knowledgebase, say $\{\langle r_1, r_2\rangle, \{s_1, s_2\}\}$, is the value of the transformation obtained by substituting the parameters by the knowledgebase, that is $\pi_1(\tau_\psi(\cap(\{\langle r_1, r_2\rangle, \{s_1, s_2\}\})))$. Transformation expressions will be denoted by $\theta, \theta', \ldots$, while $\Theta$ denotes the set of all transformation expressions. In the sequel we shall leave out extra parenthesis symbols wherever 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 \( s \) and another fixed schema \( t \) that is recursively enumerable and \( C\)-generic for some finite \( C \). (Recall that a binary relation \( r \subseteq A \times B \) is \( C\)-generic, for \( C \) a subset of \( B \cup B \), 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 non-deterministic 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 of the form \((\pi B \cup \pi T)^*\), where each \( b \) is one of \( \cap \) or \( \cup \).

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 [FKUV86] 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 \( w.r.t. \) to 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 re-
cursively indefinite databases [Mey90] can be expressed through updates. The connection between hypothetical queries and updates is explored in [GM95].

3 Sample Transformations

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 particular transformations for transitive reductions (Example 2), edges belonging to every transitive reduction (Example 3), monochromatic triangle (Example 5), and maximal clique (Example 7) are given. Since the harder graph queries build on the results of the earlier ones, 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

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

Explanation: \( \pi_2 \tau_{\phi}(\{\langle r \rangle \}) = \{\langle s \rangle \} \), 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 \( \forall x_1 x_2 x_3 R_1 x_1 x_3 \rightarrow R_2 x_1 x_3 \) and \( \forall x_1 x_2 x_3 (R_2 x_1 x_2 \wedge R_1 x_2 x_3) \rightarrow R_2 x_1 x_3 \). By the first condition, \( s \) must contain \( r \). By the second condition, for any path in \( r \), that is, any set of tuples \( \{a_1 a_2 \ldots a_{k-1} a_k\} \subseteq r \), there must be an edge \( a_1 a_k \in s \). 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:

\[
\begin{align*}
\forall x_1 x_2 x_3 : & (R_3 x_1 x_2 \wedge R_1 x_2 x_3) \lor R_1 x_1 x_3 \leftrightarrow R_3 x_1 x_3, \\
\forall x_1 x_2 x_3 : & (R_3 x_1 x_2 \wedge R_2 x_2 x_3) \lor R_2 x_1 x_3 \leftrightarrow R_3 x_1 x_3.
\end{align*}
\]

Explanation: \( \pi_2 \tau_{\psi \chi}(\{\langle r_1 \rangle \}) = \{\langle r_2 \rangle, \ldots, \langle r_2 \rangle \} \), where each \( r_2 \) 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 \( a_1 a_3 \) in \( r_2 \), there is no \( a_2 \) such that...
both \(a_1a_2\) and \(a_2a_3\) are also in \(r_2\).) Note also that by definition (9), the relation \(r_1\) does not change if suitable \(r_2\) and \(r_3\) can be found. This will indeed be the case.

At first we check that \(r_2\) satisfies the second requirement to be a transitive reduction of \(r_1\). By the first part of sentence \(\chi\), the relation \(r_3\) 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 \(\{a_1a_2, a_2a_3, a_1a_3\}\). Clearly, the relation \(r_2 \setminus \{a_1a_3\}\) 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 databases* [Mey90]. The query can also be formulated as a transformation expression. Suppose that the relation \(r_3\) describes the set of edges in question. Let the sentence \(\zeta\) be:

\[
\forall x_1x_2 : (R_3x_1x_2 \rightarrow R_2x_1x_2) \rightarrow R_4.
\]

**Explanation:** The transformation \(\pi_4\tau_\zeta\theta(\{\langle r_1, r_3 \rangle \})\), where \(\theta = \pi_{2,3} \sqcap \tau_{\psi_{\psi_{\chi}}}(\{\langle r_1, r_3 \rangle \})\) will yield one zeroary relation \(r_4\) that will contain the empty tuple if and only if \(r_3 \subseteq r_2\) where \(r_2\) 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 \(\theta(\{\langle r_1, r_3 \rangle \}) = \{\langle r_2, r_3 \rangle \}\) is as in the previous example, except that \(r_3\) is present and preserved unchanged by \(\theta\), and we take by \(\sqcap\) the common set of edges in \(r_2\).

In the \(\tau_\zeta\) 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 \(r_3\) is not a subset of \(r_2\).

**Example 4.** Transformation expressions can also describe hypothetical, or *subjunctive* queries. Recall the space example from Section 2. Let the knowledgebase be \(kb = \{\{v\}, \{\{w\}\}\}\), 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

\[
\sqcup \tau_{R_1(v)}(kb).
\]

does not contain \(w\). Since \(\sqcup \tau_{R_1(v)}(kb) = \{\{v, w\}\}\) 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 and to any right-nested \((A > (B > C))\)\ldots counterfactual query. These can be expressed by nested transformations \(\tau_A(\tau_B(\tau_C))\)\ldots

**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 \(u\) be the sentence:

\[
\forall x_1x_2 : R_1x_1x_2 \rightarrow R_2x_1x_2 \lor R_3x_1x_2.
\]

Let \(\rho\) be the conjunction of sentences:

\[
\begin{align*}
\forall x_1x_2x_3 & : R_2x_1x_2 \land R_2x_2x_3 \rightarrow \neg R_2x_1x_3, \\
\forall x_1x_2x_3 & : R_3x_1x_2 \land R_3x_2x_3 \rightarrow \neg R_3x_1x_3, \\
\forall x_1x_2 & : R_1x_1x_2 \leftrightarrow R_1x_2x_1, \\
\forall x_1x_2 & : R_2x_1x_2 \leftrightarrow R_2x_2x_1, \\
\forall x_1x_2 & : R_3x_1x_2 \leftrightarrow R_3x_2x_1.
\end{align*}
\]

Let \(\zeta'\) be the sentence \(R_6 \leftrightarrow \forall x_1x_2 \neg R_5x_1x_2\), and let the operation \(\tau_{\zeta}\) denote copying the relation \(r_1\) into a relation \(r_4\), while \(\tau_{\zeta}\) is denoting assigning the value of \(r_4 \setminus r_1\) into relation \(r_5\). (From looking at the previous examples, the last two are easily expressible as a transformation.)

**Explanation:** The result of the transformation \(\cup \tau_{\zeta'}\theta\), where \(\theta\) is the subexpression \(\pi_5\pi_6\tau_{\nu \cup \rho} \tau_{\eta}(R_1)\), has in \(r_6\) the empty tuple if and only if \(r_1\) has the described partition.

Consider at first the sentence \(u\). It formalizes the fact that \(r_2\) and \(r_3\) form a partition of \(r_1\). The disjointness of \(r_2\) and \(r_3\) is enforced by equation (3) in the definition of the \(\tau\)-operation. That is because if we had \(a_i \in r_2 \cap r_3\), then by taking \(r_2 \setminus \{a_i\}\) instead of \(r_2\) and the same \(r_3\), we could also satisfy the sentence \(u\). The choice between these two databases depends on equation (3) because the relation \(r_2\) is not an input relation. Equation (3) clearly prefers the second database.

Next see that the sentence \(\rho\) says that \(r_2\) and \(r_3\) are antitransitive, and that each of the relations \(r_1, r_2, r_3\) are symmetric.

After the second \(\tau\) operation we have all possible partitions in the knowledgebase, but we may have some undesirable partitions, that is, partitions that change the initial relation \(r_1\). Since we have a copy of the initial relation in \(r_4\), we can check whether there are any partitions that are desirable. Therefore, after \(\theta\), a required partition exists if and only if \(r_5\) is empty in some of the databases.

The transformation \(\cup \tau_{\zeta'}\) checks exactly if \(r_5\) is empty in some of the databases in \(\theta(\{\langle r_1 \rangle \})\). If \(r_5\) is indeed empty in some of the databases, then \(r_6\) will have in it the empty tuple. Otherwise \(r_6\) will be the empty relation by the minimization requirement. Hence we see that the transformation expression is a yes or no query corresponding to the monochromatic triangle problem.

**Example 6.** The *parity* problem is, does a given unary relation \(r_1\) have an even number of elements.
Let \( v' \) be the sentence
\[
\forall x_1 : R_1x_1 \rightarrow (R_2x_1 \lor R_3x_1).
\]
Let \( \varphi \) be the sentence
\[
\forall x_1x_2 : (R_2x_1 \land R_3x_2) \rightarrow R_4x_1x_2.
\]
Let \( \varsigma \) be the conjunction of sentences:
\[
\forall x_1x_2x_3 : (R_4x_1x_2 \land R_4x_1x_3) \rightarrow x_2 = x_3,
\]
\[
\forall x_1x_2x_3 : (R_4x_2x_1 \land R_4x_3x_1) \rightarrow x_2 = x_3.
\]
Let \( \rho \) be the sentence
\[
\forall x_1x_2 : (R_4x_1x_2 \lor R_4x_2x_1) \rightarrow R_5x_1.
\]
Let \( \tau_\ell \) denote the transformation that assigns \( r_1 \setminus r_5 \) to \( r_6 \). Let \( \theta \) be the expression
\[
\pi_{1,5} \tau_\ell \theta(\{ \langle r_1 \rangle \})\).
\]

**Explanation:** The transformation \( \pi_6 \tau_\ell \theta(\{ \langle r_1 \rangle \})\) results in a knowledgebase that has a database in which the relation \( r_6 \) is empty if and only if the parity 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 \( r_3 \) form a partition. Now \( r_2 \) and \( r_3 \) have an equal number of elements if and only if there is a function from \( r_2 \) to \( r_3 \) 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 \( \varsigma \) will eliminate all but those databases where \( r_4 \) is a bijection of \( r_2 \) and \( r_3 \).

The insertion of \( \rho \) adds a new relation \( r_5 \) that will contain all the elements occurring in \( r_4 \) (either as first or second argument) in each database. We claim that \( r_5 \) 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 \( r_3 \) 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 \( r_3 \), 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 \( r_5 \) 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 \( \varsigma \). 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 \( r_4 \) 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 \( r_5 \) that is equal to \( r_1 \). We can use here a the transformation \( r_6 \) similarly to the \( r_5 \) transformation in Example 5. Then \( r_6 \) 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:

\[
\begin{align*}
\forall x_1 \exists x_2 : & \quad R_2 x_1 \rightarrow R_5 x_1 x_2, \\
\forall x_1 \exists x_2 : & \quad R_4 x_1 \rightarrow R_5 x_2 x_1, \\
\forall x_1 x_2 x_3 : & \quad R_5 x_2 x_1 \land R_5 x_3 x_1 \rightarrow x_2 = x_3, \\
\forall x_1 x_2 x_3 : & \quad R_5 x_1 x_2 \land R_5 x_1 x_3 \rightarrow x_2 = x_3, \\
\forall x_1 x_2 : & \quad R_4 x_1 \land R_4 x_2 \land x_1 \neq x_2 \rightarrow R_1 x_1 x_2.
\end{align*}
\]

**Explanation:** Transformation \( r_6 \) 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 \( r_5 \) 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, \( r_5 \) is a bijection from \( r_2 \) to \( r_4 \). Hence the size of \( r_2 \) and \( r_4 \) 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 \( r_1 \) there is an edge in the graph. In other words, the 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 \( r_3 \) of size \( k+1 \).

\[
\begin{align*}
\forall x_1 \exists x_2 : & \quad R_3 x_1 \rightarrow R_6 x_1 x_2, \\
\forall x_1 \exists x_2 : & \quad R_7 x_1 \rightarrow R_6 x_2 x_1, \\
\forall x_1 x_2 x_3 : & \quad R_6 x_2 x_1 \land R_6 x_3 x_1 \rightarrow x_2 = x_3, \\
\forall x_1 x_2 x_3 : & \quad R_6 x_1 x_2 \land R_6 x_1 x_3 \rightarrow x_2 = x_3, \\
\forall x_1 x_2 : & \quad R_7 x_1 \land R_7 x_2 \land x_1 \neq x_2 \rightarrow R_1 x_1 x_2.
\end{align*}
\]

The above transformation will either not change \( r_1 \) and \( r_3 \) and find a clique of size \( k+1 \) and place it in \( r_7 \), or changes \( r_1 \) 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 or 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>((\tau, \pi))</td>
<td>(\in \text{co-NP})</td>
<td>(\in \text{co-\text{NEXPTIME}})</td>
</tr>
<tr>
<td>(\Theta)</td>
<td>(\in \text{PSPACE})</td>
<td>(\in \text{EXPSPACE})</td>
</tr>
</tbody>
</table>

In this table \((\tau, \pi)\) denotes the class of all single transformations other than \(\cup\) or \(\cap\), and \(\Theta\) 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 \(\Theta\) is \(\notin \text{NP} \cup \text{co-NP}\) and its expression complexity is \(\notin \text{NEXPTIME} \cup \text{co-NEXPTIME}\), assuming the standard hypothesis in complexity theory that \(\text{NP}\) and \(\text{NEXPTIME}\) are not closed under complement [GJ79]. We also show complexity results for quantifier-free transformation expressions, which we refer to as \(\Theta_0\).

### 4.1 Data Complexity

By the data complexity of \(\Theta\) with respect to an expression \(\theta\) we mean the complexity of deciding membership in the set

\[ C_{\theta} = \{ (db, kb) \in DB \times KB : db \in \theta(kb) \}. \]

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

**Theorem 4.1** For any \(\theta \in (\tau, \pi)\), \(C_{\theta}\) is in co-NP.

**Proof.** The case where \(\theta\) is of the form \(\pi_{i_1, \ldots, i_k}\) is obvious.

In the other case \(\theta\) is of the form \(\tau_\phi\), for some \(\phi \in \Phi\). Consider the complement of \(C_{\tau_\phi}\), that is, deciding whether \(db \notin \tau_\phi(kb)\). From the definition of the \(\tau\) operator, it follows that there are two ways that this can be the case. Either \(db\) is not a model of \(\phi\), or for each \(db_1 \in kb\) there is a \(db_2\) that is a model of \(\phi\), and such that \(db_2 <_{db_1} db\).

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 \texttt{PTIME} procedure for any fixed first-order formula.

For the second case, for each \(db_1\) in \(kb\) we guess a \(db_2\) and again we verify both of the facts that \(db_2\) is a model of \(\phi\) and that \(db_2 \prec_{db_1} db\). In other words, the symmetric difference between \(db_2\) and \(db_1\) must be less than the symmetric difference between \(db_1\) and \(db\). Since the symmetric difference between \(db_1\) and \(db\) is at most the union of the two databases, the size of each \(db_2\) guessed should not be greater than the input size. Hence whether \(db_2\) is a model of \(\phi\) can be also decided within \texttt{PTIME}. Finally, checking the condition \(db_2 \prec_{db_1} db\) can obviously also be carried out in \texttt{PTIME} for each \(db_2\) guessed.

Since we need to guess only as many \(db_2\)'s as many databases the \(kb\) contains, we could decide in \(\text{NP}\) whether \(db \notin \tau_{\phi}(kb)\).

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

**Theorem 4.2** There is a \(\theta\) in \((\tau, \pi)^*\), such that \(C_{\theta}\) is not in \(\text{NP} \cup \text{co-NP}\), unless \(\text{NP} = \text{co-NP}\).

**Proof.** We will show a particular yes or no reduction from the 3CNF formula satisfiability problem. Let the given 3CNF formula \(\phi\) be \(c_1 \land \ldots \land c_n\), where each \(c_i\) is of the form \(l_1 \lor l_2 \lor l_3\), with each \(l_i\) being a literal. We will show that there is an expression of the form \(\pi(\tau_{\psi}(\))\) that gives a yes or no query.

In the reduction we will use relations \(r_1\) (representing the clauses), \(r_2\) (representing a consistent and complete truth assignment), and \(r_3\) (representing the clauses not satisfied by the particular assignment).

Let \(kb\) be \{\(\langle r_1 \rangle\}\}, where \(r_1\) has a tuple \(t_i\) for each clause \(c_i\) of \(\psi\). For instance, if clause \(c_i\) is \(x_1 \lor \neg x_5 \lor x_8\), then \(t_i\) is \(\langle i, 1, 1, 5, 0, 8, 1\rangle\). Note that the third, fifth, and the eighth elements of the tuple \(t_i\) denote by value 1 or 0 whether the literal immediately preceding them occurs positively or negatively.

Consider now the following formulas:

\[
\forall_{x_1x_2x_3x_4x_5x_6x_7} : R_1x_1x_2x_3x_4x_5x_6x_7 \rightarrow ((R_2x_50 \lor R_2x_21) \land
(R_2x_40 \lor R_2x_41) \land (R_2x_00 \lor R_2x_01))
\]

\[
\forall_{x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}} : (R_1x_1x_2x_3x_4x_5x_6x_7 \land R_2x_2x_8 \land R_2x_4x_9 \land R_2x_6x_{10} \land
x_3 = x_8 \land x_5 \neq x_9 \land x_7 \neq x_{10} ) \rightarrow R_3
\]

The sentence \(\psi\) is the conjunction of the above two sentences.

Now it can be seen that if \(\phi\) is satisfiable, then

\[\pi_{\psi}(\tau_{\psi}(kb)) = \{\langle \emptyset\rangle, \langle \emptyset\rangle\}\].

If \(\phi\) is not satisfiable, then

\[\pi_{\psi}(\tau_{\psi}(kb)) = \{\langle \emptyset\rangle\}\].

Suppose that \(C_{\psi_{\psi}}\) is in \(\text{co-NP}\). Then it would be possible to solve the 3CNF satisfiability problem in \(\text{co-NP}\). That is, we could verify in \(\text{NP}\) for each \(kb\) if there is no

\[\text{Or } \{\langle \emptyset\rangle\}, \text{ if } \phi \text{ was a tautology.}\]
solution, by testing whether \( \langle \emptyset \rangle \not\in \pi_3(\tau_\psi(kb)) \). Conversely, suppose that \( C_{\tau_3\tau_\psi} \) is in \( \text{NP} \). Then again the 3CNF problem could be solved in \( \text{co-NP} \). This time, we would test whether \( \langle \emptyset \rangle \in \pi_3(\tau_\psi(kb)) \).

We know that the 3CNF satisfiability problem is an \( \text{NP} \)-complete problem. Now the existence of an \( \text{NP} \)-complete problem that is in \( \text{co-NP} \) entails \( \text{NP} = \text{co-NP} \).

For the upper bound of \( C_\theta \) we have:

**Lemma 4.1** For any \( \theta \) without \( \sqcup \) and \( \sqcap \) operators, the set \( C_\theta \) is in \( \text{PSPACE} \).

**Proof:** W.l.o.g. we assume that \( \theta \) is of the form \( \tau_{\phi_1}(\ldots(\tau_{\phi_1}(\ldots))\ldots) \). Let \( kb_0 \) be the input knowledgebase, and let \( kb_i \) be the knowledgebase after operation \( \tau_{\phi_i} \) is performed. In particular let \( kb_n \) be the output of the whole query, i.e., let \( kb_n = \theta(kb_0) \).

Suppose that for some database \( db_n \) we know that \( db_n \in \theta(kb_0) \) and we need to verify that this condition holds. We can do that by finding a \( db_i \) in each knowledgebase \( kb_i \) such that \( db_i \in \tau_{\phi_i}(db_{i-1}) \). In other words we must nondeterministically guess a chain of databases from an initial \( db_0 \in kb_0 \) to \( db_n \) 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 \( db_i \in \tau_{\phi_i}(db_{i-1}) \) 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 \( db_i \leq_{db_{i-1}} db \). 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. Taken in that fixed order the tags form a counter. We cycle through all possible databases by making a binary addition of one to the 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 \( \text{PSPACE} \).

The above gives a \( \text{NPSPACE} \) procedure for testing whether \( db_n \in \theta(kb_0) \). By Savitch's theorem, we have that \( \text{NPSPACE}(n) \subseteq \text{PSPACE}(n^2) \). Hence we can also do the test in deterministic \( \text{PSPACE} \).

**Theorem 4.3** For any \( \theta \in \Theta \), the set \( C_\theta \) is in \( \text{PSPACE} \).

**Proof:** Note that the relation symbols in \( \theta \) come from a fixed set, say \( \{R_1, R_2, \ldots, R_n\} \), 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 \( db_i \) 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 than \( O(B^{n^a}) \). 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 \( O(B^{n^a}) \) 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 \( \text{PSPACE} \).
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 \( db_n \) is in \( \theta(kb_0) \). To check that \( db_n \) is not in \( \theta(kb_0) \), we use the fact that PSPACE is closed under complement.

Now suppose that \( \theta \) is of the form \( \bigcap \theta' \) 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 create a counter initialized to zero. Independently from these set of 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 \( \sqcup \)-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 \( kb_0 \).

### 4.2 Expression Complexity

By the expression complexity of \( \Theta \) with respect to a pair \( \langle db, kb \rangle \in DB \times KB \) we mean the complexity of deciding membership in the set

\[
C_{\langle db, kb \rangle} = \{ \theta \in \Theta : db \in \theta(kb) \}.
\]

**Theorem 4.4** For any \( \phi \in \Phi \) and any fixed \( \langle db, kb \rangle \in DB \times KB \), we can test in co-NEXPTIME whether \( db \in \tau_\phi(kb) \).

**Proof:** We follow the same procedure as in Theorem 4.1 to test whether \( db \in \tau_\phi(kb) \). Here the size of \( \phi \) is variable. The formula can have at most \( |\phi| \) existential 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 \( C_{\langle db, kb \rangle} \) such that deciding for arbitrary \( \theta \in \Theta \) whether \( db \in \theta(kb) \) is not in NEXPTIME \( \cup \) co-NEXPTIME, unless 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 \( T = \langle K, \sigma, \delta, s_0 \rangle \), 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 $\overline{r}$ 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 $T_\tau, c_\tau$, one for each $i \leq n$, where $i$ is given in binary notation as $\tau$. 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 $\forall \tau (\tau \neq 0 \land \tau \neq 1 \land \ldots \land \tau \neq 2^n) \rightarrow T_\tau, \#$, where $0, 1, \ldots, 2^n$ 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 $\delta$ of $T$. We create for each possible machine input state $s_{in}$, input tape symbol $c_{in}$, output state $s_{out}$, tape symbol $w$, and movement indicator $m$ (being 0, 1, or $\#$) a fact $Ds_{in}, c_{in}, s_{out}, w, m$ if according to $\delta$ when the machine is in state $s_{in}$ and pointing to $c_{in}$, 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 $\#$. (Note that if $m$ is 1 or $\#$, 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 $O(k^2l^2)$ 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 $C\overline{r}, \overline{\tau}, s$ describes that at time step $\overline{r}$ the machine is in state $s$ and is pointing to tape position $\overline{\tau}$. We can assume that the Turing machine is pointing at time zero to the first tape cell. Therefore we create a fact $C0, \ldots, 0, \overline{r}, s_0$. 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 $Ra_1, \ldots, a_n, a_{n+1}, \ldots, a_{2n}, c$ in such a way that the relation records the fact that at time $t$, encoded by the binary sequence $a_1, \ldots, a_n$, the $j$th tape cell, where $j$ is encoded by the binary sequence $a_{n+1}, \ldots, a_{2n}$, contains the tape symbol $c$. To initialize $R$ we write a sentence: $\forall x_1, \ldots, x_n, y R0, \ldots, 0, x_1, \ldots, x_n, y \leftrightarrow T x_1, \ldots, x_n, y$. 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 $\phi_6$ 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 $S\overline{r}, \overline{s}$ to be true if and only if the binary number $\overline{s}$ is the successor of $\overline{r}$, in shorthand $\overline{s} = 1 + \overline{r}$. 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 $M\overline{r}, \overline{s}$, $0$ be true for each $0 \leq r \leq 2^n$, and we want $M\overline{r}, \overline{s}$, $r$ to be true for each $0 \leq r \leq 2^n$ and $\overline{s}$ successor of $\overline{r}$, and we want $M\overline{r}, \overline{s}$, $1$ be true for each $0 \leq r \leq 2^n$ and $\overline{s}$ successor of $\overline{r}$. This can be expressed using the successor function by a sentence. Let the conjunction of the two sentences 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.

\[
\forall \overline{r} \overline{r} \overline{k} \overline{s}_{in} \overline{s}_{out} \overline{c} \overline{w} \overline{m} \cdot
\]
$C \bar{t} + 1, \bar{r}, s_{out} \wedge R \bar{t} + 1, \bar{r}, w \leftrightarrow m = n \wedge C \bar{t}, \bar{r}, s_{in} \wedge R \bar{t}, \bar{r}, c_{r} \wedge D s_{in}, c_{r}, s_{out}, w, m$
\[\wedge\]

$C \bar{t} + 1, \bar{r}, s_{out} \leftrightarrow M \bar{r}, \sigma, m \wedge m \neq n \wedge C \bar{t}, \bar{r}, s_{in} \wedge R \bar{t}, \bar{r}, c_{r} \wedge D s_{in}, c_{r}, s_{out}, w, m$
\[\wedge\]

$R \bar{t} + 1, \bar{r}, \sigma \sigma \leftrightarrow R \bar{t}, \bar{r}, \sigma \sigma \wedge C \bar{t}, \bar{r}, s_{in} \wedge (\bar{r} \neq \bar{r} \vee D s_{in}, c_{r}, s_{out}, w, n)$.

This sentence says that if in time $\bar{t}$ the machine $T$ is pointing to the $\bar{r}$th position and is in state $s_{in}$, and the content of the $\bar{r}$th cell is $c_{r}$, 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 overwitten by the machine. The size of $\phi_{o}$ 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 $\exists \bar{r} C 1, \ldots, 1, \bar{r}, h$. 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 $T_{\phi_{1} \wedge \phi_{2} \wedge \phi_{3} \wedge \phi_{4} \wedge \phi_{5}}$ and let $\theta_{2}$ be the transformation $T_{\phi_{6} \wedge \phi_{7}}$. Applying $\theta_{2}$ to an empty initial knowledgebase will create the relations $T, D, S, M$ and initialize $C$ and $R$ as required.

Let $\theta_{3}$ 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 $\theta_{3}$ by a sentence of length $O(n)$. Apply $\theta_{3}$ after $\theta_{1}$.

By the minimization requirement and definition (9), whenever it is possible applying $\theta_{2}$ after $\theta_{3}$ 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 $5_{n}$, 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 were 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 projectd out any other relations beside $r_{0}$, so that only $r_{0}$ remains. The size of $\theta_{4}$ will be $O(n)$.

Let $\theta_{5}$ denote the complete transformation expression, that is, $\theta_{4}(\theta_{3}(\theta_{1}()))$. The size of $\theta_{5}$ is $O(n^{2} + k^{2}l^{2})$.

Suppose then that the language $C_{(db,kb)}$ is in NEXPTIME for any $k$ and every database $db$. We fix $k$ and $k$ to be the empty knowledgebase. Let $db_{0}$ to 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 NEXPTIME in the size of $\theta_{5}$, whether $db_{0} \in \theta_{5}(kb_{0})$. Now let $db_{1}$ be the database that contains the only relation $r_{0}$, and $r_{0}$ contains the empty tuple. We could now decide in NEXPTIME whether $db_{1} \in \theta_{5}(kb_{0})$.

Note that we can describe any fixed NEXPTIME bounded Turing machine and any variable input string of length $n$ by some $\theta_{5}$ of length polynomial in $n$. Therefore, if the language $C_{(db,kb)}$ is in NEXPTIME for every $db$ and every $kb$, then the question of whether the input tape is accepted by an NEXPTIME bounded Turing machine is in co-NEXPTIME. This in turn implies that NEXPTIME is closed under complement. Hence unless NEXPTIME
is indeed closed by complement, the language $C_{(db, kb)}$ is not in NEXPTIME. We can argue similarly that the language is also not in 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 $(db, kb) \in DB \times KB$, the set $C_{(db, kb)}$ is in 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 in EXPSPACE, and we can show that by using the same algorithm as in Theorem 4.3 together with the fact that 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 $C_{\theta}$ is in PTIME.

**Proof:** Since each sentence of $\theta$ is quantifier free, we need to make a fixed number of transformations of the form $\sqcup$, $\sqcap$, or $\tau_\phi$ where $\phi$ is a quantifier free sentence. Transformations $\sqcup$ and $\sqcap$ 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 all truth assignments to $\phi$ that can be 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 grounds atoms 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 PTIME. □

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

**Theorem 4.8** If $\theta$ is a Datalog-restricted transformation, then $C_{\theta}$ is in PTIME.

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

For the expression complexity, when $\theta$ is quantifier free the transformation language has the following bound:
Theorem 4.9 \( C_{(db,kb)} \) is not in \( \text{NP} \cup \text{co-NP} \), unless \( \text{NP} \) is closed under complement, even if \( \theta \in \Theta_0 \) 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 \( \pi_0 \), the relation \( r_0 \) still contains the empty tuple, i.e., \( db \in \pi_0 \tau_\phi(kb) \) as required. Similarly to Theorem 4.5 this leads to the conclusion that the problem is not in \( \text{NP} \cup \text{co-NP} \) unless \( \text{NP} \) is closed under complement. \( \square \)

Remark: Grahne and Mendelzon [GM95] have studied the complexity of evaluating subjective queries in a propositional language, and found the data complexity of such a language to be in PTIME and the expression complexity to be in PSPACE. The quantifier-free case of our language does not have subjective implication operators in it, otherwise it would be a proper superset of the language in [GM95]. For other complexity theoretic issues in belief revision and updates we refer the reader to [EG92, EG93, GM95].

5 Expressive Power

Let \( \text{YF}, \text{SF}, \) and \( \text{SO} \) 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 \( \text{YF} \) is properly included in \( \text{SF} \) and that \( \text{SF} \) is also included in \( \text{SO} \) [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 \( \text{ST} \) be the class of all transformations from singleton knowledgebases to singleton knowledgebases, expressible by a transformation in \( \Theta \) of the form \( (\pi \beta \tau)^* \), where each \( b \) is one of \( \cap \) or \( \cup \).

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

However, \( \text{ST} \) does include all the existential second-order queries: let \( \text{ST}^i \) denote the subclass of \( \text{ST} \) expressions that use at most \( i \) compositions of subexpressions of the form \( \pi \beta \tau \). Then all existential second-order queries can be expressed within \( \text{ST}^1 \).

Theorem 5.1 \( \text{SF} \subseteq \text{ST}^1 \).
Proof: Without loss of generality let $\exists R_{n+1} \phi(\overline{v})$ 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 $\overline{v}$.) The size of $\overline{v}$ 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 $\{\overline{a} \in D^{\vec{v}} : \text{there is a relation } r_{n+1} \subseteq D^a \text{ such that } \phi(\overline{a}) \text{ is true in } (D, r_1, \ldots, r_{n+1})\}$.

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 $2^{|D|^a}$ 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 represent the output of the query. Then we write

$$\pi_{n+2} \uplus \pi'_{n+2}(\tau_{\exists \phi(\overline{v}) \to R_{n+1}(\overline{v})})(kb)$$

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 $\forall \overline{v} \phi(\overline{v}) \rightarrow R_{n+2}(\overline{v})$ can always be satisfied by adding the required tuples for $\overline{v}$ to $r_{n+2}$.

By the minimality requirement, $r_{n+2}$ will always have only those tuples in it which satisfy $\phi(\overline{v})$. If no tuples satisfy $\phi(\overline{v})$, 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 $\pi \uplus \tau$. An interesting question is what can be gained in expressive power by the composition of $\pi \tau$ 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** $ST \subseteq SO$.

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 $\theta$ is of the form $\pi_{i_1} \uplus \tau_{i_2}$, where $j \in \{1, \ldots, n\}$, $
\sigma(db) = \{R_{i_1}, \ldots, R_{i_n}\}$, and $\{R_{i_j}\} \subseteq \sigma(\phi) \subseteq \{R_{i_1}, \ldots, R_{i_n}\}$. In the second-order syntax we therefore sometimes write $\phi$ as $\phi(R_{i_1}, \ldots, R_{i_n})$.

Suppose that the arity of $R_{i_j}$ is $k$, and for each $i$ let $R_i'$ and $S_i$ be relational variables of the same arity as $R_i$. The corresponding second-order query would then be

$$x_{1}x_{2} \ldots x_{k} : \exists R_{i_1}' \ldots \exists R_{i_n}' \forall S_{i_1} \ldots \forall S_{i_n} : \phi(x_{1}x_{2} \ldots x_{k}, R_{i_1}, \ldots, R_{i_n}, R_{i_1}', \ldots, R_{i_n}'),$$

where $\phi$ is the formula

$$R_{i_j}(x_{1}x_{2} \ldots x_{k}) \land \phi(R_{i_1}', \ldots, R_{i_n}') \land \min(\phi, R_{i_1}, \ldots, R_{i_n}, R_{i_1}', \ldots, R_{i_n}).$$

Here $\min(\phi, \ldots)$ is an abbreviation of the formula

$$(\phi(S_{i_1}, \ldots, S_{i_n}) \land (\bigwedge_{j=1}^n (S_{i_j} \leq R_{i_j} R_{i_j}'))) \to (\bigwedge_{j=1}^n (R_{i_j}' \leq R_{i_j} S_{i_j})).$$
where $S_{ij} \leq_{R_{ij}} R_{ij}'$ abbreviates
\[
\forall_{x_1x_2...x_k} : \\
((S_{ij} x_1x_2...x_k \land \neg R_{ij} x_1x_2...x_k) \lor (R_{ij} x_1x_2...x_k \land \neg S_{ij} x_1x_2...x_k)) \rightarrow \\
((R_{ij}' x_1x_2...x_k \land \neg R_{ij} x_1x_2...x_k) \lor (R_{ij} x_1x_2...x_k \land \neg R_{ij}' x_1x_2...x_k)),
\]
and likewise for $R_{ij}' \leq_{R_{ij}} S_{ij}$.

It can now be verified that for any $db \in DB$, the transformation expression $\theta$ and the above second-order query return the same result when applied to $\{db\}$.

If $\theta$ is of the form $\pi_{i_1} \cap \tau_{\phi}$, then the corresponding second-order query is
\[
x_{1}x_{2}...x_{k}\forall R_{i_{1}}' \ldots \forall R_{i_{n}}' : (\phi(R_{i_{1}}', \ldots, R_{i_{n}}') \land \min(\phi, \ldots)) \rightarrow R_{ij}' x_{1}x_{2}...x_{k}.
\]

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

To proceed, if $\theta$ is of the form $\pi_{i_{n+1}} \cup \tau_{\phi}$ where $\sigma(db) = \{R_{i_{1}}, \ldots, R_{i_{n}}\}$, and $\{R_{i_{n+1}}\} \subseteq \sigma(\phi) \subseteq \{R_{i_{1}}, \ldots, R_{i_{n}}, R_{i_{n+1}}\}$, then we apply a slight variation of the basic reduction including the necessary modification to the formula $S_{ij} \leq_{R_{ij}} R_{ij}'$ (cf. definitions (1)-(3)).

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

Finally, if $\theta$ is a composition of several $\pi br$-expressions, the corresponding second-order query will be obtained by composing the more elementary queries. $\Box$

Since $S0 = QPHIER$, the existence of a logspace reduction for unrestricted expressions in $\Theta$ to queries in $S0$ would place the $\theta$ 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 $\sqcup$ operator corresponds naturally to possibility semantics and $\sqcap$ 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 Problems

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-Mendelzon 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 \textsc{PHIER}. 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 \textsc{PSPACE} and the lower bound of not in \textsc{NPH-co-NP}. To tighten this and other bounds would require a deeper understanding of the nature of the $\sqcap$ and $\sqcup$ 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


