Algebraic Foundation and Optimization for Object Based Query Languages

Vijay M. Sarathy*  Lawrence V. Saxton†  Dirk Van Gucht* 

Abstract

We introduce the Tarski algebra as an algebraic foundation for object based query languages. While maintaining physical data independence, the Tarski algebra is shown to be both simple and powerful enough to express all reasonable queries. We show how queries expressed in a graph-oriented query language (based on the functional data model) can be translated into the Tarski algebra. The graphical representation of queries in combination with the Tarski algebra is shown to be a convenient mechanism for effective query optimization.

1 Introduction

Over the last decade, a variety of new database models have been introduced to deal with data applications involving objects with a complex external and/or internal structure. These database models can be classified into three main categories: the complex object models, the function-based object models, and hybrids of these.

Complex object models [1] extend the relational model by allowing, besides flat relations, complex object types obtained as a sequence of type constructions, such as tuple, set, list, array, and pointer formation. On the other hand, function-based object models [1, 12] view a database as a graph of objects organized in classes, where the links between objects express single-valued and multi-valued functions (relationships) between objects.

In the area of query languages for complex object databases, researchers have successfully extended the well-known relational query languages. This has resulted in calculus, algebraic, rule-based, and SQL-like query languages for complex object databases [1, 15].

In contrast, the study of query languages for function-based object databases is less developed. In this area, the most progress has been made in the specification of SQL and rule-based languages [2, 12]. Although proposals for function-based object algebras have appeared in the literature, such algebras strongly resemble the algebras defined for complex object databases [3, 5, 11, 15]. Defining algebras in this fashion deviates from the basic philosophy of the pure function-based approach as the level of abstraction at which these algebras operate is several layers higher than the graph-oriented nature of the function-based approach.

Given this situation, we propose a simple algebra, the Tarski algebra, that is appropriate to support object based query languages. Unlike previously considered algebras, the Tarski algebra operates on graphs (interpreted as binary relations) rather than on objects of complex types. In that respect, the Tarski algebra is at the level of abstraction of function-based object models and is thus more natural and effective than other algebras for such database models.

To demonstrate that the Tarski algebra is more than just an academic exercise, we give algorithms to translate queries specified in a graph-oriented query language into corresponding Tarski algebra expressions. We chose a graph-oriented query specification language because graphs are the natural representation of function-based object databases [7] and are already abundantly used in the specification and translation of high level query languages. To effectively translate query graphs into Tarski algebra expressions, we interpret labeled graphs conceptually as a set of binary relations. It is important to emphasize that this binary interpretation is purely conceptual. In particular, the actual physical representation need not conform to this interpretation. We later show how the Tarski algebra is independent of the underlying physical storage structure and therefore adheres to the paradigm of physical data independence.

In Section 2, we provide a brief introduction to the graph-oriented representation of a function-based ob-

*Computer Science Department, Indiana University, Bloomington, IN 47405. e-mail: {vijay,vugcht}@cs.indiana.edu
†Computer Science Department, University of Regina, Saskatchewan S4S 0A2, Canada. e-mail: saxton@cs.uregina.ca

1 A subset of this algebra was first considered in [8] to study the concept of object-id creation within query languages.
object data model and query language. Section 3 describes the Tarski algebra. Section 4 discusses the encoding of a function-based object database as a set of conceptual binary relations. Section 5 contains the main technical results of the paper, i.e., the translation of graph-oriented queries into equivalent Tarski algebra expressions. Section 6 discusses query optimization techniques that are relevant to this approach. Section 7 discusses how the Tarski algebra and our translation algorithm are independent of the underlying physical storage structure.

2 Graph Representation of Function Based Object Data Models

The graph-oriented representation of a function-based object database views the database as a labeled directed graph. The labeled nodes correspond to the objects in the database, and the labeled edges correspond to the functions or relationships between the objects [7].

Consider the graph in Figure 1 (top). It represents the object base schema of a persons database. The rectangular nodes represent structured object classes, whereas the circular nodes represent basic object classes. In this example there are two structured object classes, P (persons) and SP (set of persons), and one basic object class S (character strings). The edges in the schema denote functions, i.e. properties or relationships between object classes. There are two possible function types, single-valued (→) function types, and multivalued (→→) function types. In our example, the functions (properties) (P, ch, SP), (P, n, S), (P, g, S), and (P, age, S), are single-valued since a person has only a unique set of persons as children, a unique name, a unique gender, and a unique age. The function (property) (SP, c, P) is multivalued since a set of persons can consist of several persons.

The graph shown in Figure 1 (bottom) is an example of an object base instance over this persons schema. Also attached to each basic object is its value. Notice how each object, with its properties, agrees with the schema. Given this graph representation of a database, it is natural to specify queries in the form of query graphs. A query graph, relative to a given object base instance, defines a set of embeddings which are the subgraphs of the object base instance that match the query graph. For more details, see [7].

Consider the following query: Find all parent-child pairs who have the same name. The nodes in solid lines represent the actual query nodes, i.e. nodes that are involved in the actual computation of the query. The nodes in dotted lines represent the result nodes, i.e. the nodes that represent the final attributes to be reported in the answer to the query. It is possible that a node could be both a query node and a result node. Such nodes are shown with both a dotted and a solid line.

For actual computation of the answer to the query, we consider only the subset of the query graph that consists of the nodes in solid lines, i.e. the query nodes.

Figure 2: A query graph over the persons object base specifying named parent-child pairs who have the same name. The nodes in dotted lines represent result or output nodes. The two P nodes are the selected nodes.

Figure 1: An object base schema of a persons database and an instance over the persons schema.

\[^2\]The label indicates the class of the object.
This is because the result nodes do not constrain the objects involved in the query, but are just properties of the objects involved in the query. Since we are interested in the attributes of the two P nodes as output, we designate them as selected nodes (pointed to by bold arrows) in the subset query graph as shown in Figure 2.

3 The Tarski Algebra

Since the object base schemas and instances we are working with are labeled graphs, a natural (and easy) way to conceptualize them is as a collection of binary relations [9]. To manipulate these conceptual relations and adhere to the function-based approach, it is necessary to develop an algebra that is closed with respect to the class of binary relations and that is expressive enough to handle all reasonable queries. We again emphasize here, that considering graphs as a collection of binary relations is purely conceptual, and that the actual physical representation can be done in a variety of methods as discussed later.

In the 1940's Alfred Tarski proposed an algebra to manipulate binary relations [13]. The kernel of his algebra consists of four well-known operators on binary relations: union \((r \cup s)\), relation composition \((r \cdot s)\), inverse \((r^{-1})\), and (finite) complementation \((\bar{r})\). As it turns out, this algebra is not powerful enough to express all reasonable queries. The techniques to overcome this weakness are to extend the basic Tarski algebra with a constant selection operator and certain object-id creation operators.

Specifically, the full Tarski algebra has, besides Tarski's four basic operators, the constant selection operator, two ordered-pair oid creation operators, (the left- and right-oid creation operators), and one finite-set oid creation operator. The selection operator \(\sigma\), selects a pair if the right attribute of the pair is equal to some specified constant. For example, \(\sigma_{\text{Cindy}}(r)\) selects from relation \(r\) those pairs whose right attribute is equal to the constant "Cindy". We illustrate the ordered-pair oid creation operators by example.8

In Figure 3, we show the result of left and right-oid creation on a relation \(r\), denoted \(r^L\) and \(r^R\), respectively. Notice how each ordered pair in \(r\) has received a separate oid. The left-value (right-value) of that pair is found in \(r^L\) \((r^R)\). Thus, ordered-pair oid creation is a technique to create object-identifiers for ordered-pairs. If we subsequently use these oids in other ordered-pairs, we gain the capability to represent tuples of arbitrary arity. For instance, if 1 is the oid for the pair \((a, b)\), then the ordered pair \((1, c)\) is a representation of the triple \((a, b, c)\).9 This allows the succinct representation of arbitrarily structured objects within the framework of (conceptual) binary relations.

From these examples, it can be seen that the Tarski algebra is very different from other algebras for object based languages in that the structured objects are not explicitly represented, but are specified by oids. Although both approaches are equivalent, the Tarski algebra is more uniform and is not overloaded with multiple operations to deal with the complexity of differently typed structured objects, unlike most other algebras for function-based object databases [1, 15].

There are also some derived Tarski operators that will prove useful in the subsequent sections. They are \(r^L, r^R, r^L \cap r^R\), and \(r \cap s\), where

- \(r^L = (r^L)^{-1} \cdot r^L \cup (r^L)^{-1} \cdot r^R\). This is the set of pairs \((u, v)\), where \(v\) is an atomic value in \(r\), and is called the identity projection operator.

- \(r^L = (r^L)^{-1} \cdot r^L\). This is the set of pairs \((u, v)\), where \(v\) is an atomic value in the left attribute of \(r\), and is called the left identity projection operator.

- \(r^L = (r^L)^{-1} \cdot r^L\). This is the set of pairs \((u, v)\), where \(v\) is an atomic value in the right attribute of queries involving aggregate functions and grouping. It is illustrated in [10].

We must mention here, that oid creation will be used only when absolutely required for the computation. In many cases, the creation of new oids can be avoided.
of r, and is called the right identity projection operator.

- \( r^* = ((r^*) \cdot (r^*{-1})^*) \), or \( ((r^*) \cdot (r^*{-1})^*)^* \). This is the set of pairs \((t, t)\), where \( t \) is an oid of an ordered pair of \( r \), and is called the ordered-pair oid operator.

- \( r \cap s \). This is the regular intersection of two binary relations and can be simulated in the Tarski algebra as shown in [8].

4 Encoding an Object Base as Conceptual Binary Relations

Reconsider the (labeled-directed) graph in Figure 1 which represents the schema and an instance of the persons object base. This object base can be encoded as a set of binary relations, as follows:

1. For each structured object \( o \) (belonging to class \( O \) in the schema) in the instance, specify a relation named \( O \) with attributes \( o_1 \) and \( o_1 \), and add a pair \((o, o)\) to the binary relation \( O \).

2. For each property \((m, a, n)\) (of type \((M, a, N)\) in the schema) in the instance, where \( M, N \) are structured object classes, specify a relation named \( a \) with attributes \( M \) and \( N \), and add a pair \((m, n)\) to the binary relation \( a \).

3. For each property \((n, b, p)\) (of type \((N, b, P)\) in the schema) in the instance, where \( N \) is a structured object class and \( P \) is a basic object class, specify a relation named \( b \) with attributes \( N \) and \( b \), and add a pair \((n, p(b))\) to the binary relation \( b \), where \( p(b) \) is the value of the basic object \( p \).

The conceptual binary relation instance or the Tarski instance for the persons object base is shown in Figure 4.

5 The Query Graph Translation Algorithm

We now turn to translating query graphs into equivalent Tarski algebra expressions. The translation algorithm has two stages.

---

10It would have been more natural to encode \( O \) as a unary relation, but this would cause non-uniformity in our data definition and make the algebraic operations more complex.
We illustrate the use of the graph constraining technique in the section on query optimization, where we show examples of queries with constants. Techniques 1, 2, and 4 suffice for our translation algorithm, but the chain reduction technique is very useful in query optimization. For the example query translation that follows, we will not use the chain reduction technique, and then show later in Section 6 how chain reduction is useful in query optimization. We now proceed to explain the techniques of multiple edge reduction and node combination.

5.1.1 The Multiple Edge Reduction Technique

Suppose that there are multiple edges connecting objects \( M \) and \( N \) (of any type) in the query graph. Such edges can be collected into two sets, set \( F \) of edges leaving \( M \) and arriving at \( N \), and set \( T \) of edges leaving \( N \) and arriving at \( M \). The technique then computes a Tarski algebra expression \( \Psi_{M,N} \equiv \bigcap_{e \in F} \bigcap_{t \in T} t^{-1} \). The new query graph is then derived from the original subset query graph all the edges between \( M \) and \( N \) and replacing them by the single edge \( (M, \Psi_{M,N}, N) \). Clearly, a multiple edge reduction reduces the query graph by at least one edge.

5.1.2 The Node Combination Technique

This technique is more complex. It is applied whenever there are edges remaining in the query graph, and none of the other reduction techniques can be applied. Suppose \((M,a,N)\) (\(M, N\) are nodes of any type) is such an edge (we may assume that it is the only edge between \( M \) and \( N \), otherwise, a multiple edge reduction is first applied). Consider the relation \( a \) corresponding to this edge. Perform a left-oid-creation and right-oid-creation operation on \( a \). This results in the relations \( a^* \) and \( a^* \), respectively.

Using one of these newly introduced relations, construct the relation containing the new oids introduced by these operations. This can be done by the Tarski expression \( a^* \). Intuitively, each element in \( a^* \) corresponds uniquely to a pair in the relation \( a \). Now add a node (object) \( D \) to the query graph with label \( a^* \).

Now, there are four cases depending on whether there are edges entering/leaving node \( M/N \).

- For each edge \((M,b,P)\) in the query graph other than \( a \) (i.e. for each edge leaving \( M \)), add an edge with label \( a^* \cdot b \) from the new node \( D \) to node \( P \) in the query graph.

**Multiple Edge Reduction**

\[
\begin{array}{c}
\text{M} \\
\downarrow^a \\
\text{N} \\
\end{array}
\quad \rightarrow \quad
\begin{array}{c}
\text{M} \\
\downarrow^{a \cdot b \cdot c^{-1}} \\
\text{N} \\
\end{array}
\]

**Node Combination**

\[
\begin{array}{c}
\text{M} \\
\downarrow^b \\
\text{N} \\
\end{array}
\quad \rightarrow \quad
\begin{array}{c}
\text{D} \\
\downarrow^a \\
\text{N} \\
\end{array}
\]

\[
\begin{aligned}
a^* &= a^* \\
b &= a^* \cdot d \\
c^{-1} &= a^* \\
d &= a^* \\
e^{-1} &= a^* \\
\end{aligned}
\]

Figure 5: The main techniques used in the query graph translation algorithm.

- For each edge \((P, c, M)\) in the query graph (i.e. for each edge arriving at \( M \)), add an edge with label \( a^* \cdot c^{-1} \) from the new node \( D \) to node \( P \) in the query graph.
- For each edge \((N, d, P)\) in the query graph (i.e. for each edge leaving \( N \)), add an edge with label \( a^* \cdot d \) from the new node \( D \) to node \( P \) in the query graph.
- For each edge \((P, e, N)\) in the query graph other than \( a \) (i.e. for each edge arriving at \( N \)), add an edge with label \( a^* \cdot e^{-1} \) from the new node \( D \) to node \( P \) in the query graph.

Next delete nodes \( M \) and \( N \) (and also all their incident edges) from the query graph. It should be clear that this new query graph has one fewer node and one fewer edge, i.e., node combination is a graph reduction operation.

The techniques of edge reduction and node combination are summarized in Figure 5.

The full query graph translation algorithm\(^{11}\) to convert a query graph to an equivalent Tarski expression is now straightforward. Simply start with the query graph and perform the four graph reduction techniques successively (whichever is applicable). Eventually this will result in a query graph with a single node labeled by the appropriate Tarski expression. It should be noted that the query graph translation algorithm is non-deterministic because the choice of the nodes in the node combination phase is arbitrary. This offers opportunities for query optimization as discussed later.

Let us now apply this algorithm to the subset query graph shown in Figure 2. As explained earlier, we use the subset of the query graph that consists of the query

\(^{11}\)A formal proof that it is correct is included in [10].
Figure 6: Step 1: A node combination of the nodes $P$ and $SP$ results in a new node $A$ as shown in Figure 7, where $A$ stands for $ch^*$, $a$ stands for $ch^* \cdot c$, and $b$ stands for $ch^* \cdot n$.

Step 2

Figure 7: Step 2: A node combination of the nodes $A$ and $P$ results in a new node $B$ as shown in Figure 8, where $B$ stands for $a^*$, $d$ stands for $a^* \cdot b$, and $e$ stands for $a^* \cdot n$.

Step 3

Figure 8: Step 3: An edge reduction involving the edges $d$ and $e$ results in a new edge $f$ as shown in Figure 9, where $f$ stands for $d \cap e$.

Figure 9: Step 4: A node combination of the nodes $B$ and $S$ results in a new node $D$, where $D$ stands for $f^*$. Termination: Final result of the query translation algorithm.

nodes, and then later obtain the output attributes via appropriate projections onto the result nodes. In Figures 6 through 9 we show the successive steps in the translation of this query graph into the corresponding Tarski expression. We also show the intermediate binary relations at each step for ease of understanding.

The binary relation corresponding to the final object $D$ has two pairs. As illustrated in Figure 11, each of these pairs is conceptually a quadruple that corresponds to the four query node objects in the subset query graph of Figure 2. There are two pairs in the relation $D$, which correspond to the two embeddings of the query graph in the instance of Figure 1 (bottom). The final Tarski expression corresponding to the query graph of Figure 2 is:

$$D = f^*$$

$$= (f^* \cdot (f^*)^{-1})^*$$

$$= ((d \cap e)^* \cdot ((d \cap e)^*)^{-1})^*$$

$$= (((a^* \cdot b) \cap (a^* \cdot n))^* \cdot (((a^* \cdot b) \cap (a^* \cdot n))^*)^{-1})^*$$

$$= (((ch^* \cdot c)^* \cdot (ch^* \cdot n))^* \cap ((ch^* \cdot c)^* \cdot n))^*$$

$$= (((ch^* \cdot c)^* \cdot (ch^* \cdot n))^* \cap ((ch^* \cdot c)^* \cdot n))^*)^{-1}$$

5.2 Computing the Result of the Query

In the previous subsection we showed the translation of the subset query graph into a Tarski expression. In this section we will show how this expression can be used to provide the final result of the query.

Reconsider the query graph in Figure 2. We are interested in the name and gender of the parent-child pairs that have the same name. Since, we are interested in the attributes of the parent and child, we designated the two $P$ nodes as the selected objects of
the subset query graph (indicated by bold arrows in Figure 2).

Assume that we have applied the translation algorithm to the subset query graph in Figure 2. In Figure 10, we show the trace of successive node combinations that led to the binary relation \( D \). The bold nodes and edges indicate the successive node combinations. If we expand (conceptually) what the pairs in the relation \( D \) mean, we see that they are actually quadruples as shown in Figure 11 (top).

Now, as shown in the trace diagram, for each object that appeared originally in the subset query graph there is a unique path from the object \( D \) to that object along the bold nodes and edges, i.e., edges that are the result of left and right oid creation operations.

For our example, the paths (called projection paths) to each of the four objects in the query graph are:

- \((D, B, A, P)\), where \( P \) is the left \( P \) object,
- \((D, B, P)\), where \( P \) is the right \( P \) object,
- \((D, B, A, SP)\) for the \( SP \) object,
- \((D, S)\) for the \( S \) object.

Out of these four paths, the ones of interest with respect to the query are those that lead to the selected nodes \( P_1 \) and \( P_2 \). Once these paths are known, it is necessary to determine the projections of the conceptual quadruple\(^{12}\) relation onto these selected nodes. To do so, the projection paths to \( P_1 \) and \( P_2 \) will be used. The projections are computed as the composition of the edge labels appearing on the projection paths\(^{13}\). Thus, to obtain the projection on

\[ \text{Result of the query} \]

\[ \begin{array}{c|c|c|c|c}
\text{parent} & \text{child} & \text{par-gender} & \text{child-gender} \\
\hline
P_1 & P_2 & M & M \\
\hline
P_1 & P_3 & M & F \\
\hline
P_2 & P_4 & M & F \\
\hline
\end{array} \]

\[ \text{Projection on } P_1 \]

\[ \begin{array}{c|c|c|c|c}
11 & 12 & 13 & 14 \\
\hline
11 & 12 & 13 & 14 \\
\hline
11 & 12 & 13 & 14 \\
\hline
11 & 12 & 13 & 14 \\
\hline
\end{array} \]

\[ \text{Projection on } P_2 \]

\[ \begin{array}{c|c|c|c|c}
11 & 12 & 13 & 14 \\
\hline
11 & 12 & 13 & 14 \\
\hline
11 & 12 & 13 & 14 \\
\hline
11 & 12 & 13 & 14 \\
\hline
\end{array} \]

Figure 11: Result of the example query.

- \( P_1 \), we have the expression \( f^a \cdot a^s \cdot ch^s \),
- \( P_2 \), we have the expression \( f^s \cdot a^s \).

These projections are shown in Figure 11 (middle). Let us call the result of these expressions \( P_{P_1} \) and \( P_{P_2} \) respectively. We compute the parent-child pairs of the query graph with the TarSKI expression \( P_{P_1} \cdot P_{P_2} \).

Let this relation be named as \( \text{result} \). Once we have the parent-child pairs, getting the required output attributes is simple and is done by appropriate compositions as follows:

1. The name and gender of the parents are given by the expressions \( \text{(result)}^{*1} \cdot n \) and \( \text{(result)}^{*1} \cdot g \) respectively.
2. The name and gender of the children are given by the expressions \( \text{(result)}^{*s} \cdot n \) and \( \text{(result)}^{*s} \cdot g \) respectively.

The result is shown in Figure 11 (bottom). In general, the result may contain duplicates, which may be eliminated as argued in [10].

Our algorithms establish that the core of a function-based object query graph language can be simulated in the TarSKI algebra. Since the functional model conforms to the object oriented paradigm [5], our results provide a strong (yet simple) algebraic foundation for object based query languages and optimization.

6 Query Optimization

The graph-oriented TarSKI algebra approach lends itself to effective query optimization. In this section\(^{14}\),

\[ \text{bold faced nodes and edges, because they contain information encoded in the oid values created in the translation, without which the results would be erroneous.} \]

\[ \text{Due to space constraints, we only present a few important optimisation techniques here. The reader is referred to [10] for more details.} \]
we describe query optimisation heuristics and techniques relative to this approach. Though some of our heuristics and techniques correspond to similar concepts in standard Codd-relational query optimization, they take on an elegance in our framework, which is not always found in relational or complex object query optimisation. We think that this is due to the simplicity and uniformity of both the conceptualization of graphs as binary relations and the Tarski algebra.

6.1 Syntactic Optimizations at the Graph Level

6.1.1 The Graph Constraining Technique

We now explain the graph constraining technique that we referred to in the previous section. This technique helps to reduce the size of the intermediate binary relations by pushing in the selections as far as possible similar to optimization in the standard relational algebra.

Consider the query graph shown in Figure 12 (top), corresponding to the query "Find the parents of persons named Jim". To process this query graph it is best to rewrite it as the query graph shown in Figure 12 (bottom). The new node $P_{Jim}$ is computed by the Tarski expression $(P \cdot \sigma_{Jim}(n))^{n}$, where $P$ is the persons relation and $n$ is the name relation as in Figure 4. The new edge $c'$ is computed by the Tarski expression $c \cdot P_{Jim}$. After this, the algorithm proceeds as in Section 5.

Therefore, the general rule is to constrain those nodes that have associated constant valued nodes attached to them and to also constrain all other property edges entering/leaving them.

6.1.2 The Chain Reduction Technique - Avoiding Old-Creation

In combination with the graph constraining technique, this is a very effective optimization technique. It is also very simple and fits elegantly with our algebraic approach based on the Tarski algebra. Consider the subset query graph in Figure 13. There are two chains, $(P, SP, P)$ and $(P, S, P)$, between the selected $P$ nodes. The second chain can be formed because if edges in the chain are pointing in the wrong direction and prevent the collapsing of the chain, those particular edges can be replaced by their inverses. These two chains can be replaced by the single edges $(ch \cdot c)$ and $(n \cdot n^{-1})$ respectively (the compositions of the edges in the chains). The intermediate nodes $SP$ and $S$ can thus be eliminated in a single step without using the node combination technique, thereby avoiding old-creation\(^{18}\). We can then evaluate the parent-child pairs in one step with the Tarski expression $(ch \cdot c) \cap (n \cdot n^{-1})$.

6.1.3 Change of Sequence of Node Combination

The query graph translation algorithm is non-deterministic in the sense that the node combination technique can start combining any pair of nodes in the query graph. The non-determinism in the algorithm can be exploited to effectively reduce the complexity of the intermediate and final Tarski expressions by picking the appropriate pair of nodes to combine at every stage of the node combination. The heuristic rule here is to combine those nodes that affect a smaller number of edges before those nodes that affect a greater number of edges. What this means is that it is better to combine nodes that have fewer edges leaving/entering the two nodes first. The overall Tarski expression for the query graph would then be simpler.

\(^{18}\)Sometimes old-creation is unavoidable when the query graph is very complex and has no chains.
Figure 14: Semantic query optimization: logical query rewrite.

6.2 Semantic Optimizations at the Graph Level

The semantics of single valuedness of edges can be exploited to derive some useful query optimization techniques. We present two heuristics in this context.

Heuristic 1 (Logical Query Graph Rewrite)
The single-valuedness of edges can be very useful in transforming the query graph into a semantically equivalent query graph that is more amenable to syntactic optimizations. Consider an example query as shown in Figure 14 (top). This query asks for pairs of persons who have different social security numbers, but with the same first digit. The edge $SSN_m$ between the $SSN$ nodes (encoded as structured objects) relates different $SSN$ objects in the database. This query graph can be solved with the query graph translation algorithm, but since there is no chain, we cannot apply the chain reduction heuristic to simplify the query graph. However, since the $P$ nodes have a single-valued edge oid to the $SSN$ nodes, this query graph is semantically equivalent to the query graph shown in Figure 14 (bottom), where the $SSN_m$ edge between the $SSN$ nodes has been replaced by a conceptual $P_m$ edge between the $P$ nodes, that relates different $P$ nodes. This query graph can now be reduced syntactically by reducing the chain between the $P$ nodes and proceeding with the algorithm from there on. The chain can be reduced by having only one edge between the two $P$ nodes instead of the chain, given by the Tarski expression $P_m \cap (oid \cdot first\_digit \cdot first\_digit^{-1} \cdot oid^{-1})$. The final result can easily be verified to be exactly the same as would have been obtained by the naive translation algorithm, only much more easily.

Heuristic 2 (Extended Chaining)
Another example of semantic query optimization, extended chaining, is shown in Figure 15. This query selects parents with at least two children. Notice that there is a single-valued edge $ch$ between the topmost $P$ node and the $SP$ node. In this case, it is possible to generalize the chain reduction technique by replacing the two paths $(P, SP, P)$ and $(P, SP, P)$ by the composition of the constituent edges as shown in Figure 15, even though these are not explicit chains. This eliminates the $SP$ node by extended chaining and again avoids node combination. The single-valuedness of the $ch$ edge guarantees that the new query graph is semantically equivalent to the original query graph. After this step, the ordinary chain reduction technique can be used to remove the two lower $P$ nodes. If the $ch$ edge were multi-valued, this technique would not have worked. In such cases, node combination cannot be avoided to eliminate the $SP$ node.

6.3 Optimizations at the Tarski Algebra Level

6.3.1 Local Algebraic Optimization Rules

There are several algebraic optimization rules in the Tarski algebra\footnote{For example, $r \cdot (s \cup t) = (r \cdot s) \cup (r \cdot t)$; $\sigma_s(r \cdot s) = r \cdot \sigma_s(s)$; $(r \cdot s)^{-1} = s^{-1} \cdot r^{-1}$; etc.}, like in the relational or complex object algebras. These optimization rules can be exploited after the Tarski expression for the query graph is computed.

6.3.2 Evaluation of Common Subexpressions

The final Tarski expression corresponding to the query graph in Figure 2 with the naive node combination
Illustrated in Section 5.1.2, has 2 common subexpressions, \((ch^n \cdot c)\) and \((ch^n \cdot n)\), repeated several times. In the evaluation of expression \(D\), each of the subexpressions could be evaluated only once, thereby reducing the evaluation time. Since common subexpressions occur often as a result of this algorithm, this is a very useful optimization technique.

7 Physical Data Independence of the Tarski Algebra

The fact that the Tarski algebra operates on binary relations might suggest that the underlying physical storage organization of an object base needs to consist of binary relations. In this section, we show how the Tarski algebra can be used with several proposed physical organizations for manipulating complex objects, and show how the Tarski algebra is independent of the underlying physical storage structure. The representation of the object base as a collection of binary relations is purely conceptual.

The decomposed storage model (DSM) [4] is a storage organization based on binary relations in which an n-ary relation is broken into several binary relations consisting of surrogate-attribute pairs. Clearly, the DSM lends itself naturally as a physical storage model to support the Tarski algebra. The DSM is best suited for join queries that do not report more than a few attributes in the final result [14].

However, queries that need to report a large number of related attributes of an object, are better supported by the normalized storage model (NSM), which is the standard storage organization for relational databases, in which all the attributes of a structured object are clustered together in one n-ary relation. To speed up query processing, various secondary indices might exist on the n-ary relations. All the operators in the Tarski algebra can be easily implemented in this setting since the n-ary relations store binary relations implicitly. All that is necessary is a mapping between the conceptual view of the database as a collection of binary relations and the actual normalized physical organization.

The DSM and the NSM are two special cases of the more general partial decomposed storage model (P-DSM) [14], in which the partitioning of conceptual versus physical attribute representations is mixed to take maximal advantage of both the DSM and the NSM. Again, a similar argument as made in the case of the NSM, clearly shows that the Tarski algebra can easily be implemented over the P-DSM.

References


