TECHNICAL REPORT NO. 342

Translating Query Graphs Into Tarski Algebra Expressions

By

Vijay M. Sarathy, Lawrence V. Saxton
and
Dirk Van Gucht

December 1991

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY
Bloomington, Indiana 47405-4101

Translating Query Graphs into Tarski Algebra Expressions

Vijay M. Sarathy
Indiana University
Computer Science Department
Bloomington, Indiana 47405-4101, USA
e-mail: vijay@cs.indiana.edu

Lawrence V. Saxton
University of Regina
Department of Computer Science
Regina, Saskatchewan S4S 0A2, Canada
e-mail: saxton@mercury.cs.uregina.ca

Dirk Van Gucht
Indiana University
Computer Science Department
Bloomington, Indiana 47405-4101, USA
e-mail: vgucht@cs.indiana.edu

Abstract

Query graphs have been frequently used as intermediate representations of queries during query translation and optimization in the relational database domain. Furthermore, the target implementation platform has usually been a conventional non-decomposed storage model.

In this paper we show how query-graphs (expressed in the Graph Oriented Object Data model) can be mapped into the *decomposed storage model* which has binary relations as its basic data structures. Since the Tarski algebra is specifically designed to manipulate binary relations, we chose it as a low level language for the operational component of the DSM.

We present an algorithm for mapping query-graphs into equivalent Tarski algebra expressions. We also address several interesting query optimization issues, both at the graph level and at the algebra level.

1 Introduction

Since the mid 1980's many new data models and database systems have been proposed and implemented to support data applications involving *complex objects*. This has necessitated the development of new query languages for such databases. Unlike the situation for the relational model, there is no general consensus about the style and/or the degree of expressiveness of such languages.

On the other hand, it is generally agreed that any database system, be it post-relational, object-oriented, or knowledge-based, must offer a high-level *declarative* query language which is effectively and efficiently supported by the system's query translator, query optimizer and query access plan generator. To facilitate these processes, database implementors and database theoreticians alike, have frequently resorted to intermediate query representations. One of the most important classes of such representations is the class of *query graphs*.

Example 1.1 Consider the following simple SQL query which computes, relative to a Parent-Child(Id,CId) relation, the pairs of (different) grandparents that have a common grandchild (the id of the grandchild is also reported):

```
SELECT GP1.Id, GP2.Id, P1.CId
FROM   Parent-Child GP1 GP2 P1 P2
WHERE  GP1.Id <> GP2.Id   AND
       GP1.CId =  P1.Id   AND
       GP2.CId =  P2.Id   AND
        P1.CId = P2.CId
```

In Figure 1 we show how this query is represented in three different query-graph "languages." Notice the similarity between these representations. Although the example is relational, query-graphs similar to those is Figure 1 have also been used to represent post-relational and object oriented query languages.

Given the importance of query graphs in the query processing cycle, the next questions become: 1) how to translate a query-graph into an equivalent procedural program?, and 2) if there are different such programs, which program is likely to be the most efficient?

In this paper we address both questions relative to a simple "procedural" language, the so called *Tarski algebra* [8], which is an algebra operating on the data structures of the *decomposed storage model* (DSM) [11].

In the DSM all data is stored as *binary relations*. As is well known, binary relations have no limitations to model relational databases. For example an ordinary (flat) relation $R(A_1, \ldots, A_n)$ can be represented by n binary relations $R_1(S, A_1), \ldots, R_n(S, A_n)$ where S is a new attribute often called a *surrogate* or *object-identifying* attribute. What is perhaps less known is that binary relations also have no limitations to model complex objects. A relation with complex objects can be represented by a collection of binary relations [4, 5]. Thus the DSM is completely general for modeling data objects.

There are various reasons why we chose the DSM as our target translation platform.

- DSM's versatility in uniformly modeling a wide range of data objects, from the very simple to the very complex.

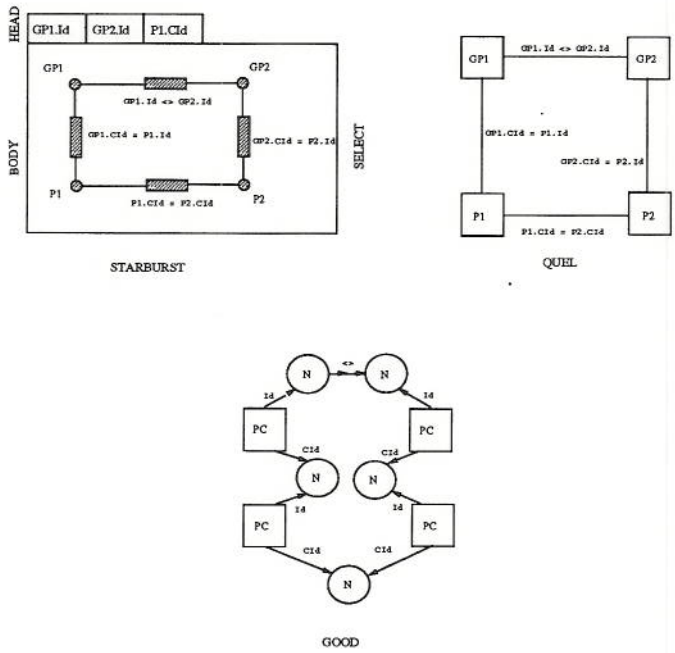


Figure 1: A query in the query graph model of Starburst (top-left), in the INGRES QUEL-connection graph model (top-right), and as a pattern in the Graph-Oriented-Object-Database (GOOD) model (bottom).

- DSM's core data structures are *graphs*.[1] It is thus reasonable to expect that query-graphs will be more naturally translated into such data structures.
- Binary relations are already at the heart of any efficient implementation of a database. For instance, *indexes* are nothing but a collection of $(value, address)$ pairs, thus they are binary relations. Chains (and rings) are nothing but binary relations of the form $\{(p_1, p_2), (p_2, p_3), (p_3, p_4) \ldots\}$. Thus, a translation algorithm that can conceptually incorporate such information is at an advantage.
- As pointed out by Copeland et.al. [11, 4], the DSM offers advantages over tightly-clustered storage models when considering implementations of database systems on *parallel architectures*.

There are also various reasons why we chose the Tarski algebra:

- The Tarski algebra is an algebra closed with respect to the class of binary relations. This is not the case for Codd's relational algebra. Indeed, the join operator allows the construction of relations of arbitrary arity.
- The Tarski algebra has its roots in the foundations of set theory. (Tarski introduced it in the 1940's principally as an algebraic tool to study set theory.) Hence, just as is true for the Codd-relational algebra, the Tarski algebra has a sound mathematical foundation.
- Since binary relations suffice to model arbitrary complex objects, the Tarski algebra offers a procedural (low) level language to express *both* relational and object-oriented queries.

[1] The word "graph" is just a synonym for "binary relation".

The notion of query-graph we consider in this paper is the concept of *pattern* in the GOOD model. The similarity between the query graphs shows that this is a reasonable choice. Thus the results in this paper carry over to other query-graph formulation languages. However, there are special advantages to selecting the GOOD model. The model is completely graph-oriented, i.e, its database schemes and instances, and its queries are all graphs. This renders it a good conceptual model for object-oriented databases and query languages as well as for the DSM. One important property of the GOOD query language is that it is both Codd-complete for relational and nested relational databases [7]. This is unlike other graph-oriented query languages, (Starburst's query-graph model being a notable exception) which focus mostly only on conjunctive, monotonic queries. Thus, the results we obtain in this paper address the entire range of queries considered reasonable (i.e., polynomial-time), not just a special subclass.

We obtain the following results:

- A *pattern (i.e., query-graph) translation algorithm* converting patterns into equivalent Tarski algebra expressions.
- A complete translation of the GOOD query language into Tarski algebra expressions. This indirectly establishes that the Tarski algebra is Codd-complete.
- Query optimization principles useful in the context of translating query-graphs into an algebra closed on the structures of the *decomposed storage model*.

In Section 2, we provide a brief introduction to the GOOD model. Section 3 provides an introduction to the Tarski algebra. Section 4 discusses the mapping of database schemes and instances into the DSM. Section 5 contains the main results of the paper, the translation of graph-oriented queries into equivalent Tarski algebra expressions. Section 6 contains query optimization techniques at the query-graph and Tarski algebra level, respectively. We conclude our paper in Section 7 with a discussion about directions for future research.

2 The Graph Oriented Object Data model (GOOD)

The GOOD model is an object oriented database model in which both data representation and data manipulation are graph-based, i.e., database schemes, database instances, and data manipulation operations are specified as graphs [6, 7].

A *GOOD database schema* is a labeled directed graph, with nodes that represent object classes and edges that specify relationships. Consider the graph in Figure 2. It represents the schema of a persons (P) and vehicles (V) database (vehicles can be cars (C) or airplanes (A)). The rectangular nodes represent *complex object classes*, whereas the circular nodes represent *basic object classes*. So in this example there are four complex object classes P, V, C and A, and two basic object classes, i.e. the class of numbers N and the class of character strings S. The edges in the schema denote relationships. There are two possible edge types, *multivalued* ($\rightarrow\!\!\rightarrow$) edge types and *functional* ($\rightarrow$) edge types. For example, the edge (P, owns, C) is multivalued since a person can own several cars, whereas the edge $(V, \text{s\#}, N)$ is functional since a vehicle has a *unique* serial number.

The graph shown in Figure 3 is an example of a *database instance* over this persons and vehicles schema.[2] Also attached to each base object is its value. Notice how each node, with its incidents edges, agrees with the schema.[3]

[2]The edge $OWNS$ between the node types P and C, is introduced to illustrate the use of non-functional edges, though it is redundant in this example in light of the $OWNER$ edge between the node types C and P.

[3]The instance need not contain all the nodes and edges specified in the schema, it can be a subset satisfying all the constraints specified in the schema, and maybe disconnected.

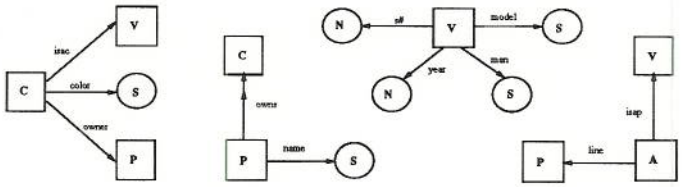


Figure 2: The GOOD database schema of the persons and vehicles database.

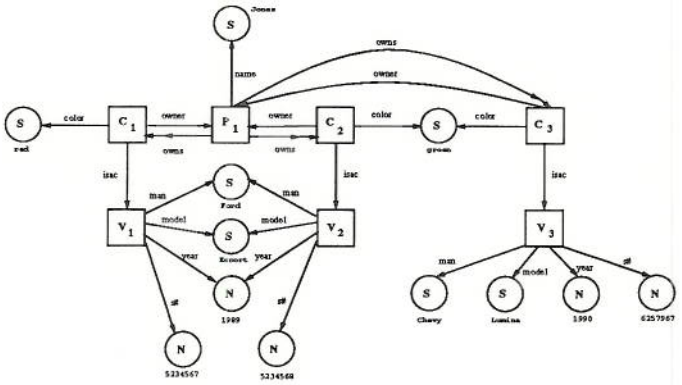


Figure 3: A GOOD database instance over the persons and vehicles schema.

In the GOOD model, data manipulation operations such as queries, restructurings, and updates are expressed as graphs transformations. Each graph transformation has two parameters, a *pattern parameter*, denoting the objects involved in the graph transformation, and an *action parameter*. An action parameter is one of: a *node addition*, an *edge addition*, a *node deletion*, an *edge deletion*, an *abstraction* or a *method call*. [4]

More formally, a pattern, relative to a given database instance, defines a set of *embeddings* which are the subgraphs of the database instance that match the pattern. In Figure 4, we show a pattern representing pairs of cars with the same manufacturer. For the instance in Figure 3, this pattern defines five embeddings corresponding to the five car-pairs (C_1, C_2), (C_2, C_1), (C_1, C_1), (C_2, C_2), and (C_3, C_3) that have the same manufacturer.

Rather than consider all action parameters here, we will only illustrate one such action parameter, i.e., a node addition, and discuss the other cases in the remainder of the paper. Reconsider the pattern in Figure 4. Suppose that we want to specify *new* objects corresponding to car pairs that have the same manufacturer. This can be specified in the GOOD data manipulation language

[4] In this paper we only consider the first four action parameters.

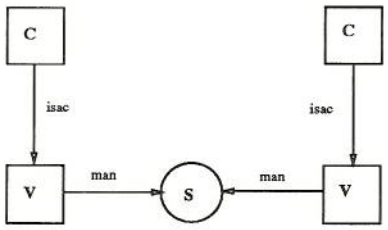


Figure 4: A GOOD pattern over the persons and vehicles database specifying pairs of cars which have the same manufacturer.

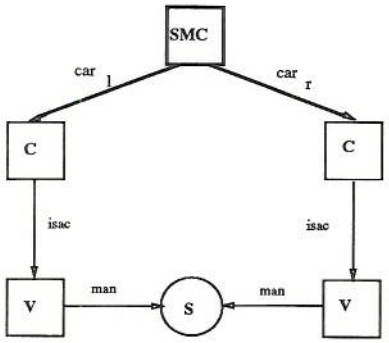


Figure 5: An example of a node addition.

through the node addition operation shown in Figure 5. The graph in that figure contains two distinguishable parts: the first (non-bold) part is the pattern parameter and the second (bold) part is the action parameter. The action parameter specifies the types of node and edges to be added in the course of the node addition. The semantics of this operation is to add, for each embedding of the pattern, a new node (with associated edges) to the database instance. Thus in the case of our running example five new nodes and ten new edges are added.[5] Once conceptually or truly added, the node type SMC can serve the role of a new complex object type in subsequent GOOD data manipulation operations.

3 The Tarski Algebra

Since GOOD database schemes and instances are graphs, a natural way to represent them is as a collection of binary relations [14]. Granted this representation, and with data manipulation operations in mind, it is natural to search for a useful, and hopefully small, set of operations on such binary relations.

In the 1940's Alfred Tarski proposed an algebra to manipulate such binary relations [18].

The kernel of his algebra consists of four well-known operators on binary relations: union ($r \cup s$), relational composition ($r \cdot s$)[6], inverse (r^{-1}), and (finite) complementation ($\overline{r}$)[7]. As it turns out, this algebra is *not* powerful enough to express all reasonable queries. In [8] the basic Tarski algebra was extended to overcome this weakness. The key idea of that paper was to extend the basic Tarski algebra with certain *tagging operators*. These operators allow the creation of tags[8] for ordered-pair and finite-set objects. These tags are succinct representations for these objects and can in turn be used as components of ordered pairs. This allows the representation of arbitrary complex objects within the framework of binary relations.

Specifically, the full Tarski algebra has, besides Tarski's original four basic operators, two ordered-pair tagging operators, i.e, the *left-* and *right-tagging* operators, one *finite-set tagging* operator, and the *new* operator. We illustrate these operators by example.

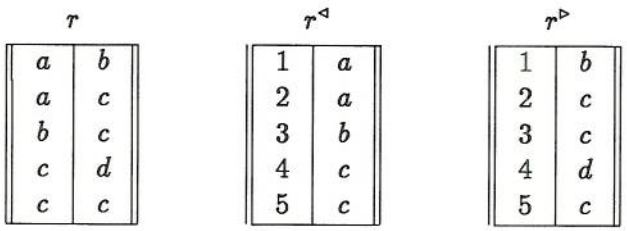
Example 3.1 In Figure 6, we show the result of left and right-tagging a relation r, denoted $r^{\triangleleft}$, and $r^{\triangleright}$, respectively. Notice how each ordered pair in r has received a separate tag. The left-value

[5] Of course, the notion of adding these objects has to be taken liberally in the sense that the addition may simply be in the form of an output to a query, or a view materialization, or as an intermediate result.

[6] The set of ordered pairs (u, w) such that there exists a value v, such that the ordered pair $(u, v) \in r$ and $(v, w) \in s$.

[7] For a more formal definition of the four basic operators, see Appendix A.

[8] Also called object identifiers in other papers.

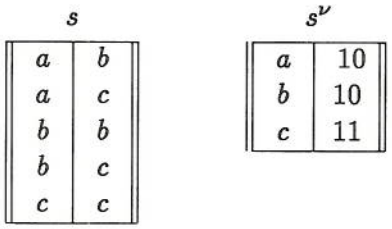


r	
a	b
a	c
b	c
c	d
c	c

$r^{\triangleleft}$	
1	a
2	a
3	b
4	c
5	c

$r^{\triangleright}$	
1	b
2	c
3	c
4	d
5	c

Figure 6: Example of left and right tagging the relation r.



s	
a	b
a	c
b	b
b	c
c	c

s^{ν}	
a	10
b	10
c	11

Figure 7: Example of finite-set tagging a relation s. The tag 10 represents the finite set $\{b, c\}$, and tag 11 represents the finite singleton $\{c\}$.

(right-value) of that tag is found in $r^{\triangleleft}$ ($r^{\triangleright}$). Thus, ordered-pair tagging is a technique to create tags (object-identifiers) for ordered-pairs. If we subsequently use these tags in other ordered-pairs, we gain the capability to represent tuples of arbitrary arity. For instance, if 1 is the tag for the pair (a, b), then the ordered pair $(1, c)$ is a representation of the triple (a, b, c).

We would like to stress that the actual values of the tags are immaterial, as long as they are *unique* representatives of their corresponding complex objects.

Example 3.2 In Figure 7, we show the result of finite set-tagging a binary relation s, denoted s^{ν}. This operation associates to the left component u of an ordered pair (u, v) in s the tag of the finite set $\{w \mid (u, w) \in s\}$. Notice how in the figure the set-tag associated to a and b are the same! Hence, set-tagging allows us to represent finite sets through "atomic" tag values. Notice that we can obtain the set membership relation by composing $(s^{\nu})^{-1}$ (the inverse of s^{ν}) with s. In this composition the set-tag 10 in Figure 7 gets associated with its members b and c, and the set-tag 11 with its element c.

In addition to the various tagging operators, the Tarski algebra has one extra operator, i.e., the *new* operator. The new operator, denoted r^{new}, when applied to an arbitrary binary relation r, generates a singleton binary relation $\{(v_{new}, v_{new})\}$, where v_{new} is a value *not* occurring in r. Notice that as a special case this operator behaves as an object creation operator. Indeed, if $r = \emptyset$ then, by definition, r^{new} introduces a value not occurring in $\emptyset$, i.e. a new tag (or object-identifier).[9]

The four basic Tarski operators and the three tagging operators form the *Tarski algebra*. It was shown in [8] that this algebra is at least as expressive as the Codd-relational algebra and the nested relational algebra. In this paper we will show that it is also powerful enough to support the

[9] It should be noted that the finite-set tagging and new operator were not included in the Tarski algebra as defined in [8].

graph transformations of GOOD. Since that language was designed to specify both relational and object oriented queries, this implies that the Tarski algebra and the DSM offer a platform for both relational and object oriented query processing.

There are also some derived Tarski operators that will prove useful in the subsequent sections. They are r^{id}, r^{τ}, $r \cap s$, $r - s$ and r^{un}. These definitions are:

- $r^{id} = (r^{\triangleleft})^{-1} \cdot r^{\triangleleft} \cup (r^{\triangleright})^{-1} \cdot r^{\triangleright}$.
  This is the set of pairs (v, v), where v is an atomic value in r, and is called the *identity* operator.

- $r^{\tau} = ((r^{\triangleleft}) \cdot (r^{\triangleleft})^{-1})^{id}$, or $((r^{\triangleright}) \cdot (r^{\triangleright})^{-1})^{id}$. This is the set of pairs (t, t), where t is a tag of an ordered pair of r, and is called the *ordered-pair tags* operator

- $r \cap s$.
  This is the *regular intersection* of two binary relations. This is quite complex to express in the Tarski Algebra, and is outlined in [8].

- $r - s = r \cap (\overline{s \cup r^{\mathrm{id}}} \cup \overline{s \cup r^{\mathrm{di}}})$.
  This is the regular *set difference* of two binary relations.

- $r^{un} = r \cup \overline{r}$.
  This is the set of pairs (v, w), where v and w are atomic values occurring in r, and is called the *relative universe* operator.

4 Mapping of the GOOD Data Representation into the DSM

In this section, we show how a database schema and instance in the GOOD model can be mapped into the decomposed storage model.[10]

4.1 Mapping a GOOD Schema

Reconsider the (labeled-directed) graph in Figure 2 which represents the schema of a persons and vehicles database. This GOOD schema can be mapped into a binary relation schema as follows:

1. For each *complex object node* O in the schema, specify a relation schema named O (or a more mnemonic name) with attributes o_l and o_r.

2. For each labeled edge (M, e, N) in the schema, where M, N are complex object nodes, specify a relation schema named (M, e, N) (or simply e if there is no ambiguity), with attributes M and N.

3. For each labeled edge, (N, e, P) in the schema, where N is a complex object node, and P is a basic object node, specify a relation schema named (N, e, P) (or simply e if there is no ambiguity) with attributes N and e.

The binary relation schema, or the *Tarski schema*, for the persons and vehicles database is shown in Figure 8.

[10] We refer to the DSM and the Tarski model interchangeably.

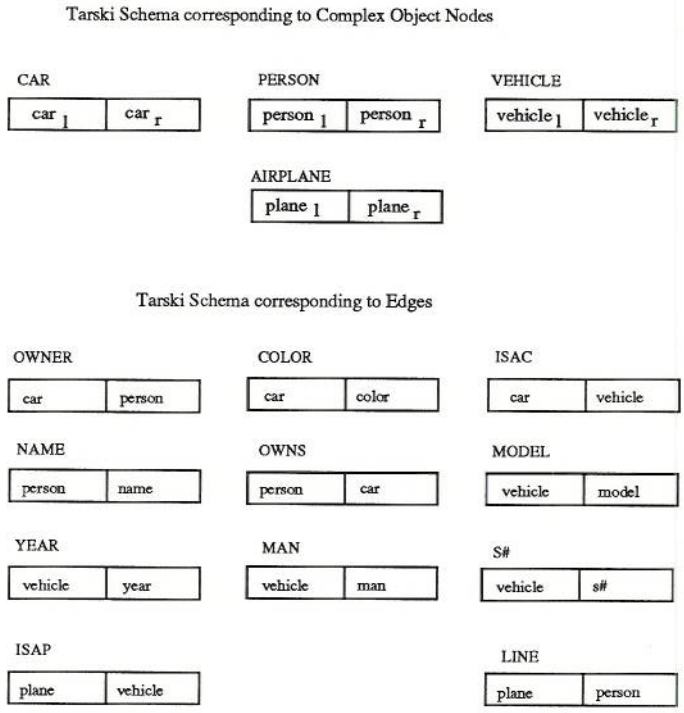


Figure 8: The Tarski schema of the persons and vehicles database.

4.2 Mapping a GOOD Instance

Reconsider the (labeled-directed) graph in Figure 3 which represents an instance of the persons and vehicles database. This instance can be mapped into a binary relation database instance, or a Tarski instance, as follows:

1. For each complex object node o of type O in the instance, add a pair (o, o) to the binary relation O.

2. For each labeled edge (m, e, n) of type (M, e, N) in the instance, where M, N are complex object nodes, add a pair (m, n) to the binary relation (M, e, N).

3. For each labeled edge, (n, e, p) of type (N, e, P) in the instance, where N is a complex object node, and P is a basic object node, add a pair $(n, \lambda(p))$ to the binary relation (N, e, P), where $\lambda(p)$ is the value of the basic object node p.

The binary relation instance or the *Tarski instance* for the persons and vehicles database is shown in Figure 9.[11]

5 Translating GOOD Data Manipulation Operations

We now turn to translating GOOD data manipulation operations into the Tarski model. As previously discussed, each GOOD operation has two parameters, called a *pattern* and an *action* [6, 7].

[11] It must be noted here that the mapping strategy is the same for functional and multi-valued edges. The functionality of an edge, may however be useful for query optimization at the graph level as will be shown by an example later.

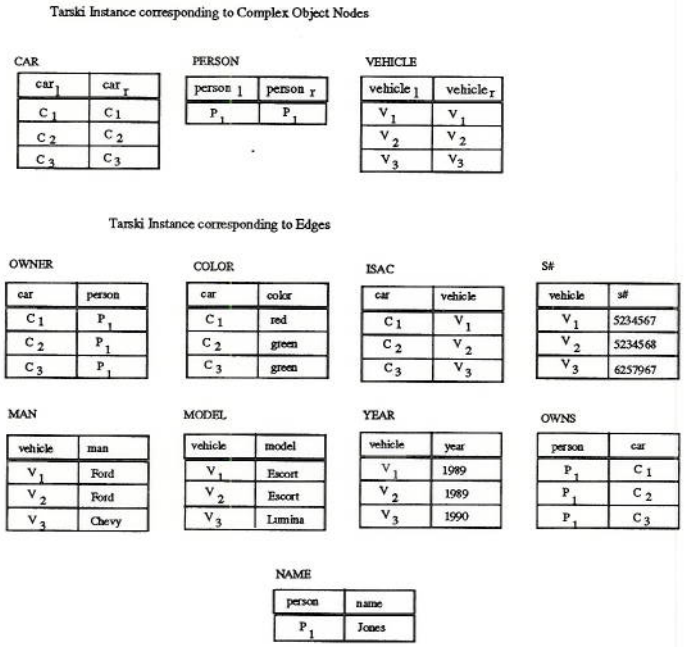


Figure 9: The Tarski instance of the persons and vehicles database.

The pattern specifies the conditions to be satisfied by the objects participating in the operation, and the action indicates whether the operation is a node addition, edge addition, etc. The translation of such manipulation operations into the Tarski model thus involves two steps:

- a translation of the pattern into a corresponding Tarski algebra expression, followed by
- a translation of the specific action into a corresponding Tarski algebra expression.

5.1 Translating Patterns

We first consider translating connected patterns without constants. Next we show how to make a minor change to the algorithm to deal with pattern involving constants. Finally, we show how to translate a pattern containing disconnected components.[12]

The translation of a pattern (or query graph) into a Tarski expression uses two essential graph reduction techniques.

1. The *multiple edge reduction* technique, in which multiple edges between a pair of nodes in the pattern are replaced by a single edge, and
2. The *node combination* technique in which two nodes connected by an edge in the pattern are combined into one composite node.

The *multiple edge reduction* technique is straightforward. Suppose that there are multiple edges connecting nodes M and N, of any type, in the given pattern P. Such edges can be collected into two sets,

1. $\mathcal{E}_1 = \{(M, e_1, N) \mid e_1$ an edge in P leaving node $M\}$, and

[12]For readers familiar with the Wong-Youseffi decomposition algorithm for conjunctive queries [21], we would like to stress that there are similarities but also differences between their algorithm and our pattern translation algorithm.

2. $\mathcal{E}_2 = \{(N, e_2, M) \mid e_2$ an edge in P entering node $M\}$.

The technique then computes a Tarski algebra expression

$$\Psi_{M,N} \equiv \cap_{e_1 \in \mathcal{E}_1}(M, e_1, N) \bigcap \cap_{e_2 \in \mathcal{E}_2}(N, e_2, M)^{-1}$$

The new pattern is then derived by removing from the original pattern all the edges between M and N and replacing these by the single edge $(M, \Psi_{M,N}, N)$. Clearly, a multiple edge reduction reduces the pattern by at least one edge.

The *node combination* technique is more complex. It is applied whenever there are still edges in the pattern. Suppose (M, e, 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 (M, e, N) corresponding to this edge. Perform a left-tagging and right-tagging operation on (M, e, N). This results in the relations $(M, e, N)^{\triangleleft}$ and $(M, e, N)^{\triangleright}$, respectively.

Using one of these newly introduced relations, construct the relation containing the tags introduced by these operations. This can be done by the Tarski expression $(M, e, N)^{\tau}$. Intuitively, each element in $(M, e, N)^{\tau}$ corresponds *uniquely* to an edge in (M, e, N). Now add a node to the pattern with label $(M, e, N)^{\tau}$.

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

- For each edge (M, f, P) (M, P are nodes of any type) in the pattern, other than (M, e, N) (i.e., for each edge in the pattern leaving M), construct a new relation $((M, e, N)^{\triangleleft} \cdot (M, f, P))$ and add an edge from the new node (M, e, N) to node P in the pattern.

- For each edge (P, f, M) in the pattern, (i.e., for each edge in the pattern arriving in M), construct a new relation $((M, e, N)^{\triangleleft} \cdot (P, f, M)^{-1})$ and add an edge from the new node (M, e, N) to node P in the pattern.

- For each edge (N, f, P) (N, P are nodes of any type) in the pattern, (i.e. for each edge in the pattern leaving N), construct a new relation $((M, e, N)^{\triangleright} \cdot (N, f, P))$ and add an edge from the new node (M, e, N) to node P in the pattern.

- For each edge (P, f, N) in the pattern, other than (M, e, N) (i.e., for each edge in the pattern arriving in N), construct a new relation $((M, e, N)^{\triangleright} \cdot (P, f, N)^{-1})$ and add an edge from the new node (M, e, N) to node P in the pattern.

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

The full *pattern translation algorithm* to convert a pattern to an equivalent Tarski expression is now straightforward. Simply start with the pattern and perform successive multiple edge reduction and node combination operations. Eventually this will result in a pattern with a single node labeled by the appropriate Tarski expression. It should be noted that the pattern translation algorithm is non-deterministic because the choice of the nodes in the node combination phase is arbitrary. As will be shown later, this offers opportunities for query optimization. A formal proof that the node combination technique is correct is provided in Appendix B. It is fairly easy to see that the algorithm terminates because both the *edge reduction* and the *node combination* techniques result in a smaller pattern, as indicated earlier. Thus the algorithm is a graph reduction algorithm at every step, and is hence guaranteed to converge down to a single node and terminate.

Let us now apply this pattern translation algorithm to the pattern shown in Figure 4. In Figures 10 to 13, we show the successive steps in the transformation from this pattern to the

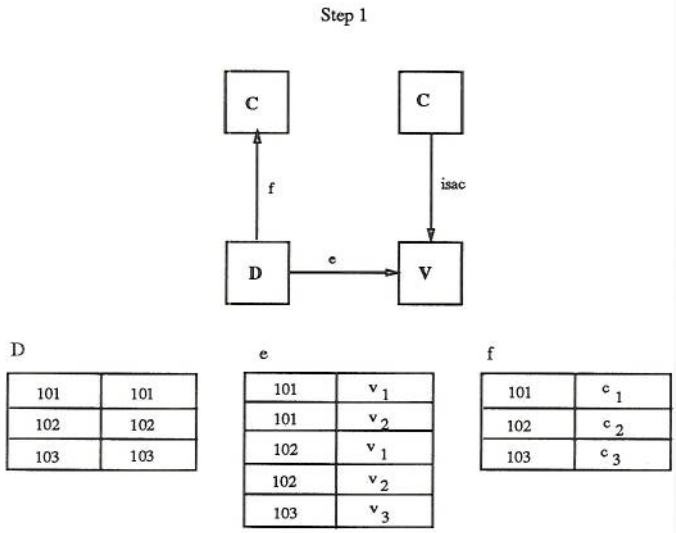


Figure 10: Step 1 in the transformation from the pattern in Figure 4 into the corresponding Tarski expression, i.e. a node combination involving the edge (V, man, S). D, which combines the nodes V and S connected by the edge (V, man, S), stands for MAN^{τ}, e stands for $MAN^{\triangleright} \cdot MAN^{-1}$, and f stands for $MAN^{\triangleleft} \cdot ISAC^{-1}$.

corresponding Tarski expression. We also show the intermediate binary relations at each step for ease of understanding.

The binary relation corresponding to the final node H has five pairs. As illustrated in Figure 16, each of these pairs is conceptually a quintuple that corresponds to the five nodes in the original pattern of Figure 4. Also, there are five pairs in the relation H, which correspond to the five embeddings of the pattern in the instance of Figure 3.[13] The final Tarski expression corresponding to the pattern of Figure 4 is

$$
\begin{aligned}
H &= i^{\tau} \\
&= (i^{\triangleleft} \cdot (i^{\triangleleft})^{-1})^{id} \\
&= ((h^{\triangleleft} \cdot g)^{\triangleleft} \cdot ((h^{\triangleleft} \cdot g)^{\triangleleft})^{-1})^{id} \\
&= (((e^{\triangleleft} \cdot ISAC^{-1})^{\triangleleft} \cdot (e^{\triangleleft} \cdot f))^{\triangleleft} \cdot (((e^{\triangleleft} \cdot ISAC^{-1})^{\triangleleft} \cdot (e^{\triangleleft} \cdot f))^{\triangleleft})^{-1})^{id} \\
&= ((((MAN^{\triangleright} \cdot MAN^{-1})^{\triangleleft} \cdot ISAC^{-1})^{\triangleleft} \cdot ((MAN^{\triangleright} \cdot MAN^{-1})^{\triangleleft} \cdot (MAN^{\triangleleft} \cdot ISAC^{-1})))^{\triangleleft} \cdot \\
&\quad ((MAN^{\triangleright} \cdot MAN^{-1})^{\triangleleft} \cdot ISAC^{-1})^{\triangleleft} \cdot (((MAN^{\triangleright} \cdot MAN^{-1})^{\triangleleft} \cdot (MAN^{\triangleleft} \cdot ISAC^{-1}))^{\triangleleft})^{-1})^{id}
\end{aligned}
$$

5.1.1 Translating Patterns with Constants

Translating GOOD patterns with constants is a little more complicated than patterns without constants. To do so, we need to introduce a new operator to the Tarski Algebra described in [8], which we call the the *select operator* $\sigma_{rightattr='const'}(r)$.[14] This is just like the select operator in the relational algebra and selects only those pairs from r that satisfy the selection condition. We

[13] Of course it is entirely coincidental that the number of embeddings and the number of nodes in the pattern is five. In general, of course, there is no relationship between these quantities.

[14] We select the right attribute because constants occur only as values of basic object nodes and since all basic object nodes have only edges entering them, the constant values occur only as the right attribute of the corresponding binary relation.

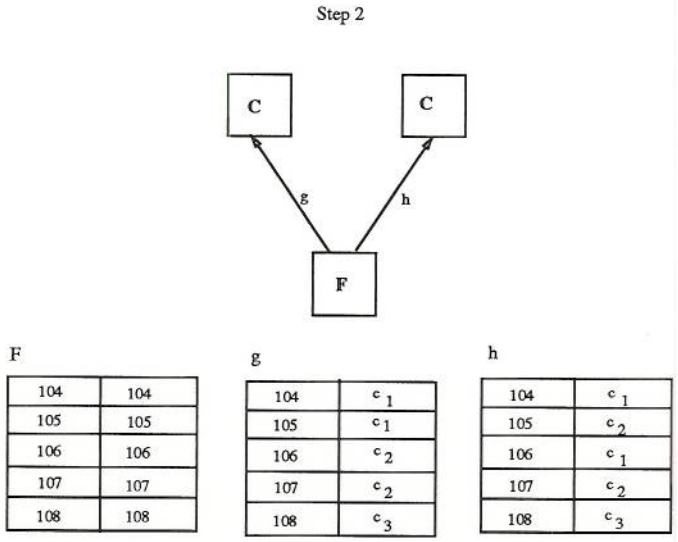


Figure 11: Step 2 in the transformation from the pattern in Figure 4 into the corresponding Tarski expression, i.e., a node combination involving the edge (D, e, V). F, which combines the nodes D and V connected by the edge (D, e, V), stands for e^{τ}, g stands for $e^{\triangleleft} \cdot f$, h stands for $e^{\triangleright} \cdot ISAC^{-1}$.

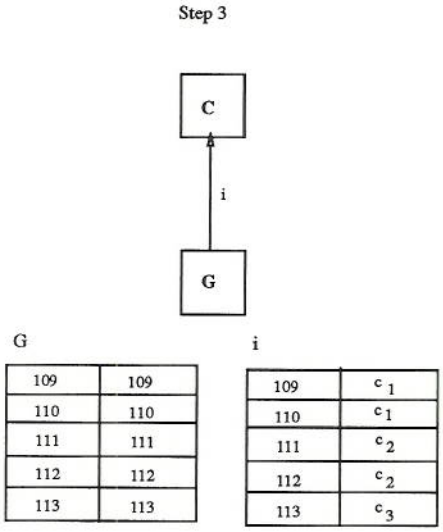


Figure 12: Step 3 in the transformation from the pattern in Figure 4 into the corresponding Tarski expression, i.e., a node combination involving the edge (F, h, C). G, which combines the nodes F and C connected by the edge (F, h, C), stands for h^{τ}, i stands for $h^{\triangleleft} \cdot g$.

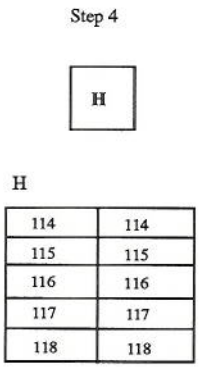


Figure 13: Step 4 in the transformation from the pattern in Figure 4 into the corresponding Tarski expression, i.e., a node combination involving the edge (G, i, C). H, which combines the nodes G and C connected by the edge (G, i, C), stands for i^{τ}.

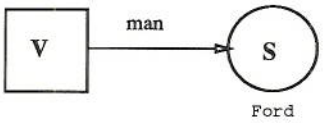


Figure 14: Translating GOOD patterns with constants.

now make a slight modification in the pattern translation algorithm. Suppose that there is an edge e between nodes N and P, where N is a complex object node and P is a basic object node with a constant value C. Instead of using the relation e corresponding to the edge e to combine nodes N and P as was done earlier, we now use the relation $\sigma_{e=C}e$ to combine the nodes N and P. Let us call this relation e_C. This applies to every edge that arrives at a basic object node with a value. Everything else in the algorithm remains the same.

Consider the pattern in Figure 14. Here complex object node V is connected to basic object node S with constant value *'Ford'*, by the edge *man*. The algorithm would use the relation $\sigma_{man=Ford}(MAN)$ (called MAN_f for future reference) instead of MAN to combine nodes V and S.

5.1.2 Translating Patterns with Disconnected Components

Thus far we have tackled the translation of connected patterns. If however, the pattern is disconnected, we perform the pattern translation algorithm on each connected component of the pattern and then combine the different "final nodes" by taking the *cartesian product* of the corresponding Tarski expressions. The cartesian product is unavoidable in this case. Also, it is the only place where it is used in the pattern translation algorithm. This is obviously a desirable property of our translation algorithm.

Assume we have a pattern with $n \geq 2$ connected components $C_1, C_2, \ldots C_n$. Let $T(C_1), T(C_2), \ldots, T(C_n)$ be the Tarski expressions derived using the pattern translation algorithm for the components $C_1, C_2, \cdots, C_n$, respectively. The cartesian product of these Tarski expressions is done as follows. Consider $T(C_1)$ and $T(C_2)$ and the Tarski expression $cart_{C_1,C_2}$ defined as

$$T(C_1) \cdot (T(C_1) \cup T(C_2))^{un} \cdot T(C_2)$$

(Recall that r^{un} is a short-hand for the expression $r \cup \overline{r}$.)

The main term in the expression $cart_{C_1,C_2}$ is the factor

$$(T(C_1) \cup T(C_2))^{un}$$

This factor contains all pairs (v, w) where v and w are values in $T(C_1) \cup T(C_2)$. However, what we really need are all the pairs (v, w), where v is in $T(C_1)$ and w is in $T(C_2)$. To select these pairs from $(T(C_1) \cup T(C_2))^{un}$, we compose this expression on the left with $T(C_1)$ and on the right with $T(C_2)$. The result of this expression returns the desired pairs (and thus the cartesian product).

Once this expression is evaluated we need to reduce the pattern with n components to a pattern with $n-1$ components. To achieve this we drop the nodes corresponding to C_1 and C_2 and add a node with name $cart_{C_1,C_2}$. In addition, we associate with this node, the binary relation

$$(cart_{C_1,C_2})^{\tau}$$

(This relation will contain a tag for each pair in the cartesian product of $T(C_1)$ and $T(C_2)$.)

We now repeat the same process on the pattern with $n-1$ nodes until we reach a single node. This node will correspond to the cartesian product of $T(C_1), T(C_2), \ldots, T(C_n)$.

5.2 Translating Node Addition

In the previous subsection we showed the translation of a GOOD pattern into a Tarski expression. In this section we will concentrate on the node addition operator and show how this action can be translated into a corresponding Tarski expression.

5.2.1 Node Additions with Selected Nodes

Reconsider the pattern in Figure 4. Suppose that we want to specify new objects corresponding to car pairs such that the two cars have the same manufacturer. This can be specified in the GOOD data manipulation language by the node addition operation shown in Figure 5. This figure contains two distinguishable parts: the first (non-bold) part is the pattern shown in Figure 4, and the second (bold) part specifies the types of nodes and edges to be added. The semantics of this operation is that for each embedding of the pattern in an instance, a new node is added with edges as specified in the bold. When applied to the instance shown in Figure 3, there will be five such node additions since there are five embeddings of the pattern in the instance.

As can be seen from Figure 5, the node addition parameter, indicates two *special* nodes in the pattern, i.e., the two CAR nodes. As will become clear from the discussion about the simulation of the other action parameters, each GOOD operation identifies special nodes in a pattern. We will say that an action parameter *selects* special nodes in the pattern, and we will call such nodes *selected nodes*.

Assume that we have applied the pattern translation algorithm to the pattern in Figure 4. In Figure 15, we show the "trace" of successive node combinations that led to the binary relation H. The bold nodes and edges indicated the successive node combinations. If we expand (conceptually) what the pairs in the relation H mean, we see that they are actually quintuples as shown in Figure 16. Now, as shown in the trace diagram, for each node that appeared originally in the pattern there is a *unique* path from the node H to that node along the *bold* nodes and edges, i.e. edges that are the result of left and right tagging operations.

For our example, the paths to each of the five nodes in the pattern are:

- (H, C_l), where C_l is the left-most C node,
- (H, G, C_r), where C_r is the right-most C node,

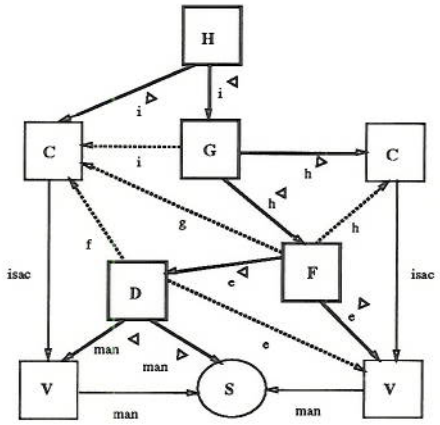


Figure 15: Trace of the node combinations performed on the pattern in Figure 4.

- (H, G, F, D, V_l), where V_l is the left-most V node,
- (H, G, F, V_r), where V_r is the right-most V node, and
- (H, G, F, D, S) for the S node.

Once these paths are known, it is straightforward to determine the five "projections" of the quintuple relation onto its respective attributes.[15] It is important that these "projection paths" involve only the bold faced nodes and edges, i.e. the nodes created during the pattern translation algorithm, and the edges that involve the left and right tagging operators. We do not traverse a path that involves the nodes and edges involved in the original pattern, (except of course, the final node to be projected on) or the edges that do not involve a left or right tagging operator because doing so would cause us to lose the information contained in the tag values created in the translation, thereby causing erroneous results. Each projection corresponds to the composition of the edge labels appearing on the just mentioned paths. Thus, to obtain the "projection" on:

- C_l, we have the expression $i^{\triangleright}$,
- C_r, we have the expression $i^{\triangleleft} \cdot h^{\triangleright}$,
- V_l, we have the expression $i^{\triangleleft} \cdot h^{\triangleleft} \cdot e^{\triangleleft} \cdot MAN^{\triangleleft}$,
- V_r, we have the expression $i^{\triangleleft} \cdot h^{\triangleleft} \cdot e^{\triangleright}$,
- S, we have the expression $i^{\triangleleft} \cdot h^{\triangleleft} \cdot e^{\triangleleft} \cdot MAN^{\triangleright}$

Let us call the result of these expressions P_{C_l}, P_{C_r}, P_{V_l}, P_{V_r}, and P_S, respectively.

We finally can specify the effect of the node addition in Figure 5. In this action, C_l and C_r are the selected nodes in the pattern. The effect of the node addition is:

1. Addition of a new relation CAR_L corresponding to the car_l edge. This relation is given by the expression P_{C_l}.
2. Addition of a new relation CAR_R corresponding to the car_l edge. This relation is given by the expression P_{C_r}.

[15] The attributes of the quintuple correspond to each of the nodes involved in the original pattern.

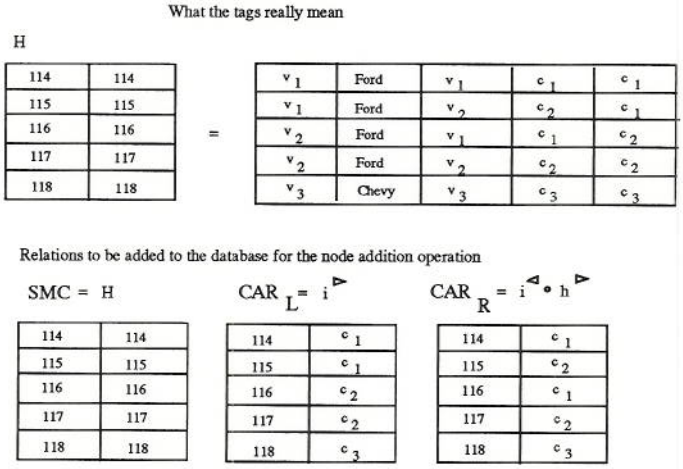


Figure 16: Relations to be added to the database for the node addition in Figure 5.

3. Addition of a new relation SMC corresponding to the SMC node. This relation is the relation H.

The result is shown in Figure 16.

5.2.2 Node Additions without Selected Nodes

In the previous section, the node addition operation had a *non-empty* set of selected nodes in the pattern. The GOOD model also defines a node addition operation in which there are no selected nodes. The semantics of such a node addition relative to a pattern P can best be thought of as an if-then-else statement. Let $T(P)$ be the Tarski expression corresponding to the pattern P. If $T(P)$ evaluates to a non-empty set then the effect of the node addition is to add a *single* new node to the database, otherwise, the node addition has no side-effect.

As shown in [8], it is possible to write a Tarski expression corresponding to an if-then-else statement:

$$\text{if } E_1 \neq \emptyset \text{ then } E_2 \text{ else } E_3$$

where E_1, E_2 and E_3 denote Tarski expressions.[16]

The Tarski expression which corresponds to a node addition without selected nodes then simply becomes:

$$\text{if } T(P) \neq \emptyset \text{ then } T(P)^{new} \text{ else } T(P)$$

Recall that the Tarski new operator always adds exactly one new tag value.

5.2.3 Node Addition in the Absence of a Pattern

The GOOD model also supports node addition without a pattern parameter. The semantics of such an operation is to just add one node. So the corresponding Tarski operation is simply $\emptyset^{new}$.

[16]The Tarski expression corresponding to this if-then-else statement is:

$$E_2 \cdot (E_1 \cdot (E_1 \cup E_2)^{\text{un}})^{\text{id}} \cup E_3 - E_3 \cdot (E_1 \cdot (E_1 \cup E_3)^{\text{un}})^{\text{id}}$$

Relations to be added to the database for the node addition operation that includes identical pairs

SMC

114	114
115	115
116	116
117	117
118	118

CAR_L

114	c_1
115	c_1
116	c_2
117	c_2
118	c_3

CAR_R

114	c_1
115	c_2
116	c_1
117	c_2
118	c_3

Relations to be added to the database for the node addition operation that removes identical pairs

115	115
116	116

$CAR_L - CAR_R$

115	c_1
116	c_2

$CAR_R - CAR_L$

115	c_2
116	c_1

Figure 17: Final result without identical pairs.

Recall that the *new* operator always introduces a new tag value not occurring as a value in its operand. In this case the operand is empty, so the effect is simply a new tag.

5.2.4 Removing Identical Pairs from the Result

The problem of identical pairs being part of the result can be resolved in two different ways. In the first method, the problem is tackled at the very end. Perform the node addition operation exactly as before. Now when it comes to adding the new relations, instead of adding the relations CAR_L and CAR_R to the database, add the relations $(CAR_L - CAR_R)$ and $(CAR_L - CAR_R)$, and instead of adding the relation SMC add the relation $((CAR_L - CAR_R)^\tau)$.[17] This is illustrated in Figure 17.

We will illustrate the second method of removing identical pairs in the context of query optimization.

5.2.5 Removing Duplicate Pairs from the Result

This problem can be tackled by the use of the *abstraction* operator in the GOOD model. The idea of the abstraction operators is that it groups into finite sets objects that are equivalent in a certain way. For example, if t_1 and t_2 are different objects having as value (i.e. local state, in the object-oriented sense) the same complex object, say for example the ordered pair (a, b), then it might be said that t_1 and t_2 are equivalent in the sense of local state. Abstraction is an operation which can collect such equivalent objects into a new object with members as the elements of an equivalence class. In our example this object would correspond to the set $\{t_1, t_2\}$. Now this object can be thought of as the *abstraction* of the objects t_1 and t_2. Abstraction has many practical applications, *duplicate elimination* being one of those.

We will not illustrate the translation of abstraction in the GOOD model into the Tarski Algebra in this paper, but we will just mention that the finite-set tagging operator is the key operation in the Tarski algebra for this simulation.

[17] Recall that r^τ is a relation of identical pairs of tag values of the ordered pairs of r.

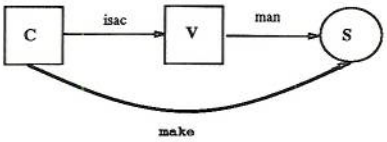


Figure 18: Pattern and edge to be added in the edge addition.

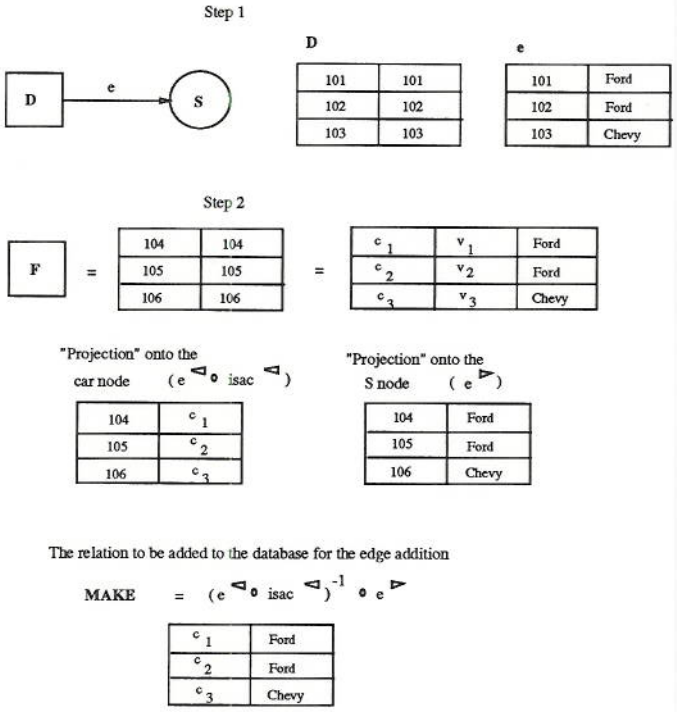


Figure 19: Relation to be added to the database for the edge addition in Figure 18, D stands for $ISAC^{\tau}$, e stands for $ISAC^{\triangleright} \cdot MAN$, F stands for e^{τ}.

5.3 Translating Edge Addition

New edges are introduced in the GOOD model via edge additions. Consider the edge addition shown in Figure 18. Again, the non-bold part of the figure is the pattern and the bold part is the action specifying the edge(s) to be added in the operation. The effect of this edge addition is to add for each car a new edge, *make*, directly to its manufacturer.

This edge addition can be realized in the Tarski model as follows:

First compute the Tarski expression corresponding to the given pattern using the pattern translation algorithm (see Figure 19). Once we have the final Tarski expression (single node) corresponding to the pattern (just like in the node addition case), we do a "projection" (in the sense of Section 5.2 on the selected attributes (i.e., on the two nodes between which the new edge is to be added). In the current example, the selected attributes are C and S.

In general, let the edge to be added be from node M to N. Let the projection of the final Tarski expression on to node M be g, and on to node N be h. Then, compute $g^{-1} \cdot h$, and add this relation to the database. This will exactly correspond to the new edge between the nodes M and N. The effect of this is shown in Figure 19.

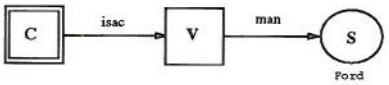


Figure 20: Pattern and node to be deleted in the node deletion.

5.4 Translating Node Deletion

A node deletion operation removes certain nodes. Consider the node deletion shown in Figure 20. Again there is a pattern and an action, however this time the action is indicated by a box in double outline.[18] This node is by definition the selected node of this action. The effect of this node deletion is the removal of all cars manufactured by "Ford".

Node deletion is complicated because besides the removal of nodes from the database, extra work is required to delete all the edges to/from the nodes to be deleted.

Again, we first compute the Tarski expression corresponding to the given pattern. The steps in the translation process are shown in Figure 21. Once we have the final Tarski expression (single node) corresponding to the pattern, (like the node addition case), we do a projection on the selected node (corresponding to the node to be deleted). Then we do the following operations to update the relations in the database to reflect the node deletion.

Let the projection on to the selected node to be deleted be g, and let the relation in the database corresponding to the type of the node to be deleted be R.

1. Compute $g^{-1} \cdot g$. This will denote all the pairs to be deleted from R.
2. Update the node relation R to $R - g^{-1} \cdot g$.
3. For all the other edge relations S in the database which involve R objects,
   - If S has the R objects represented in its left attribute, update them as $S - g^{-1} \cdot (g \cdot S)$. This will correspond to deleting all edges leaving the node to be deleted.
   - If S has the R objects represented in its right attribute, update them as $S - (S \cdot g^{-1}) \cdot g$. This will correspond to deleting all edges entering the node to be deleted.

The effect of this is shown in Figure 21.

5.5 Translating Edge Deletion

An edge deletion operation removes certain edges. Consider the edge deletion shown in Figure 22.[19] The edge in double outline is the action parameter. The effect of this edge deletion is the removal of edges of type *man* leaving vehicles manufactured by "Ford." The selected nodes are the source and sink nodes of the edge in double outline. In the current example these nodes are V and S.

First compute the Tarski expression corresponding to the given pattern. The steps in the translation process are shown in Figure 23. Once we have the final Tarski expression (single node) corresponding to the pattern, we do a projection on the selected nodes. Then we do the following operations to update the relations in the database to reflect the edge deletion.

[18] As per the conventions in the GOOD model [6, 7], the whole figure is the pattern and the *double outlined* node is the node to be deleted.

[19] As per the conventions in the GOOD model [6, 7], the whole figure is the pattern and the *double outlined* edge is the edge to be deleted.

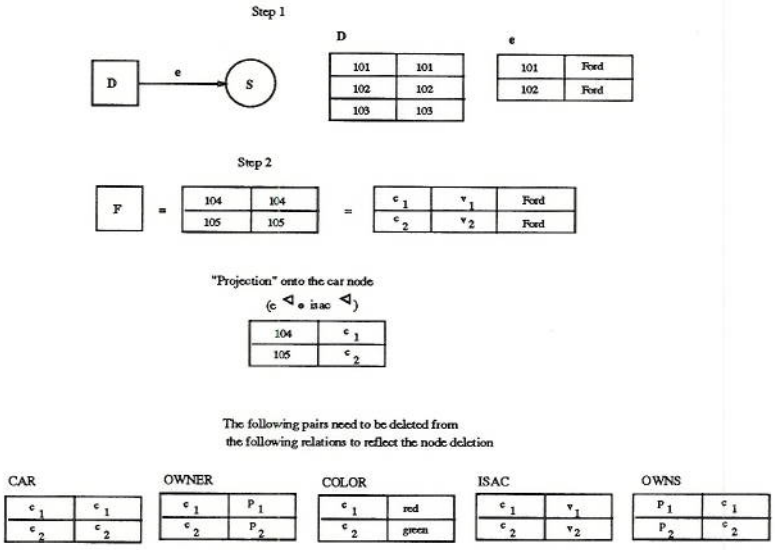


Figure 21: Relations to be updated in the database for the node deletion in Figure 20, D stands for $ISAC^{\tau}$, e stands for $ISAC^{\triangleright} \cdot MAN_f$, F stands for e^{τ}.

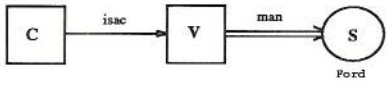


Figure 22: Pattern and edge to be deleted in the edge deletion.

Let the edge to be deleted be from node M to N, let the projection on to node M be g, and let the projection on to node N be h. Let the relation corresponding to the edge to be deleted be R.

1. Compute $g^{-1} \cdot h$. This will correspond to the edge to be deleted.
2. Update the edge relation R to $R - g^{-1} \cdot h$.

The effect of this is shown in Figure 23.

5.6 The Complete Translation Algorithm

In Figure 24 we outline the complete algorithm for translating the query-graph data manipulation operations in GOOD into their corresponding Tarski expressions.

As a by product of our translation, this establishes that the core of the GOOD query-graph data manipulation language can be simulated in the Tarski Algebra. In addition, it is relatively straightforward to map each basic Tarski operation into a series of GOOD data manipulation operations as shown in Figure 25. Therefore, we can state the following theorem.

Theorem 1 *The Tarski algebra and the core of the GOOD query-graph data manipulation language can express the same set of queries.*

Since query-graphs in other query-graph languages can be mapped in a straightforward manner into corresponding GOOD query-graph operations, this theorem renders the Tarski algebra and our translation algorithm applicable to such query-graph languages.

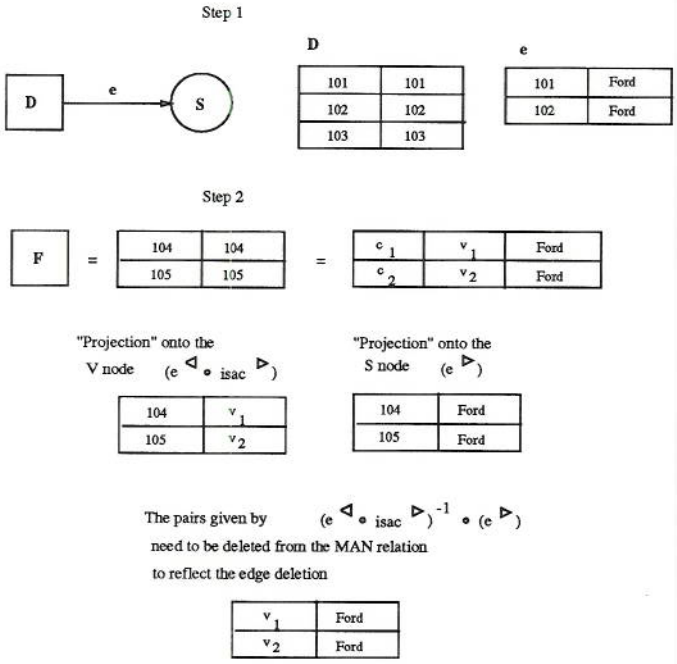


Figure 23: Relations to be updated in the database for the edge deletion in Figure 22, D stands for $ISAC^{\tau}$, e stands for $ISAC^{\triangleright} \cdot MAN_f$, F stands for e^{τ}.

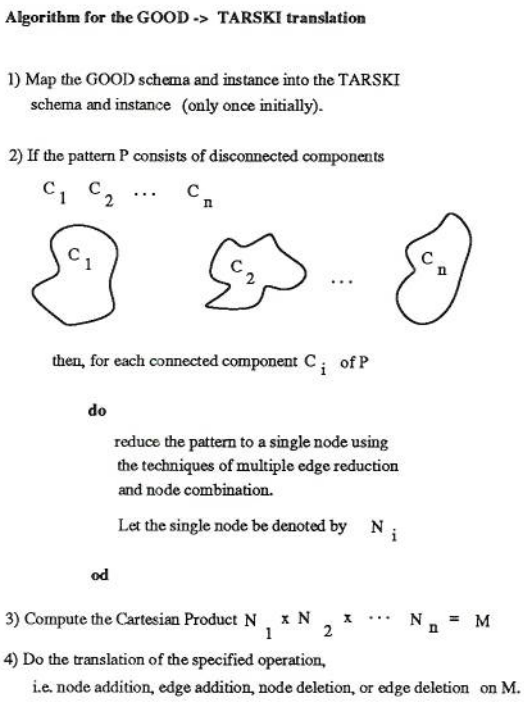


Figure 24: Complete algorithm for translating the GOOD data manipulation operations into the Tarski algebra.

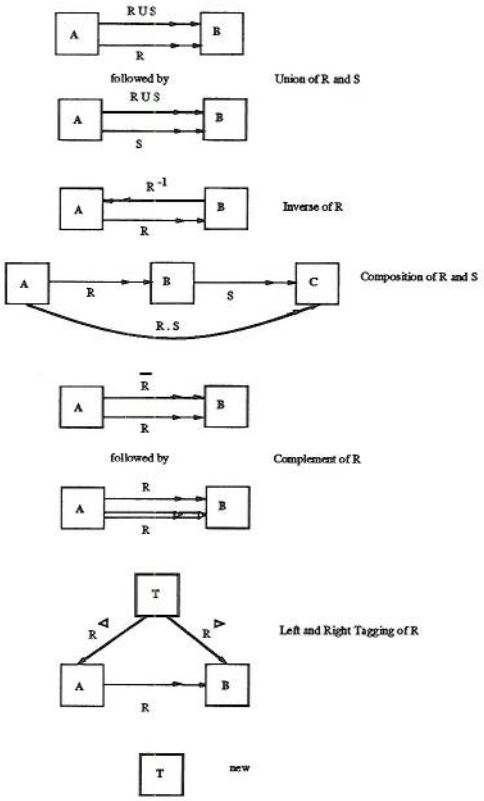


Figure 25: GOOD patterns corresponding to the Tarski operations.

6 Query Optimization Techniques

In this section, we will describe some ideas and results about query optimization relative to the previously described translation process.

The problem of optimizing the translation process just described can be expected to be at least as difficult as optimizing arbitrary relational calculus queries, since such queries can be readily converted into the GOOD data manipulation operations.

As such, we present several *heuristics* which can be used in the translation process of GOOD data manipulation operations into optimized Tarski expressions. We certainly don't claim that all these ideas are new, in fact many heuristics are reformulations of similar heuristics used in standard Codd-relational query optimization.[20]

However, we have found that these heuristics take on an elegance in our framework, which is not always found in standard relational query-optimization. We think that this is due to the simplicity and uniformity of both the decomposed storage model and the Tarski algebra.

6.1 Syntactic Optimizations at the Graph Level

6.1.1 Change of Sequence of Node Combination

The pattern translation algorithm is non-deterministic in the sense that the node combination technique can start combining any pair of nodes in the pattern. 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.

Let us now illustrate how this non-determinism can be exploited and what we mean by an "appropriate" pair of nodes by means of an example. In step 1 of the node addition operation in Figure 10, we combined nodes V and S. These two nodes had a combined total of 3 edges entering/leaving them. It would have been better to start off by combining nodes C and V first because these nodes have a combined total of 2 edges leaving/entering them and then proceeding in the sequence shown in Figure 26 (we leave out the intermediate details). The overall Tarski expression for the pattern in this case is simpler than the one obtained earlier.

The overall Tarski expression computed with this node combination sequence is

$$\begin{aligned}
H &= g^{\tau} \\
&= (g^{\triangleleft} \cdot (g^{\triangleleft})^{-1})^{id} \\
&= ((f^{\triangleright} \cdot ISAC^{-1})^{\triangleleft} \cdot ((f^{\triangleright} \cdot ISAC^{-1})^{\triangleleft})^{-1})^{id} \\
&= (((e^{\triangleright} \cdot MAN^{-1})^{\triangleright} \cdot ISAC^{-1})^{\triangleleft} \cdot (((e^{\triangleright} \cdot MAN^{-1})^{\triangleright} \cdot ISAC^{-1})^{\triangleleft})^{-1})^{id} \\
&= ((((ISAC^{\triangleright} \cdot MAN)^{\triangleright} \cdot MAN^{-1})^{\triangleright} \cdot ISAC^{-1})^{\triangleleft} \cdot \\
&\quad ((((ISAC^{\triangleright} \cdot MAN)^{\triangleright} \cdot MAN^{-1})^{\triangleright} \cdot ISAC^{-1})^{\triangleleft})^{-1})^{id}
\end{aligned}$$

This is certainly simpler than the Tarski expression with the naive node combination.

The *heuristic rule* here therefore 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. In other words, it is better to combine nodes that have a smaller fan-in and fan-out sum first. This is similar to *pruning a tree starting at the leaves.*

[20] For a thorough introduction to relational query optimization, we refer to [20], Chapter 11 and Chapter 14. The latter chapter on conjunctive query optimization is highly relevant.

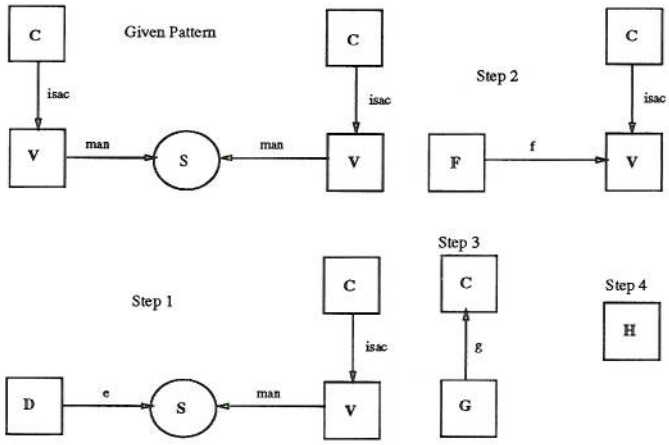


Figure 26: Change of sequence of node combination for optimization. D stands for $ISAC^{\tau}$, e stands for $ISAC^{\triangleright} \cdot MAN$, F stands for e^{τ}, f stands for $e^{\triangleright} \cdot MAN^{-1}$, G stands for f^{τ}, g stands for $f^{\triangleright} \cdot ISAC^{-1}$, H stands for g^{τ}.

6.1.2 Combine Nodes that Involve Constants Early

This technique helps to reduce the size of the intermediate binary relations by pushing in the selections as far in as possible just like optimization in the standard relational algebra.

6.1.3 Collapse a Chain into a Single Edge

This is by far the most effective optimization technique. It is also very simple. Consider the pattern in Figure 27. If a chain is detected in the graph, it can be replaced by a single edge (which is just the composition of the component edges in the chain) between the two extreme nodes as shown in the figure. The intermediate nodes can be removed in one single step. Of course, this is only possible if the intermediate nodes are not involved in the selected nodes of the actual action to be performed. Also, 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. The algorithm can proceed as before after this optimization step.

Also, if there is a choice between two different chains, (and both involve using the inverse edges to make the chain) it is better to collapse the chain that has edge inverse relations (indexes) already stored in the database [4]. This will further enhance the speed of the operation, by exploiting the index (inverse relation) information.

6.1.4 Avoid Combination of Selected Nodes

Recall that selected nodes refer to the nodes that are involved in the GOOD data manipulation operation, i.e. the actual nodes involved in the node/edge addition/deletion. The following heuristic aims at combining other nodes first and postponing as late as possible the combination of selected nodes. Of course, node combination of one of the selected nodes and other nodes is allowed, otherwise the algorithm will not converge. As is evident from the projection argument in Section 5 it might be possible to directly read off a solution when we are left with the selected nodes and an edge connecting them. This rule might lead to a conflict with the first two rules, but the optimizer should choose the best possible alternative as is normal.

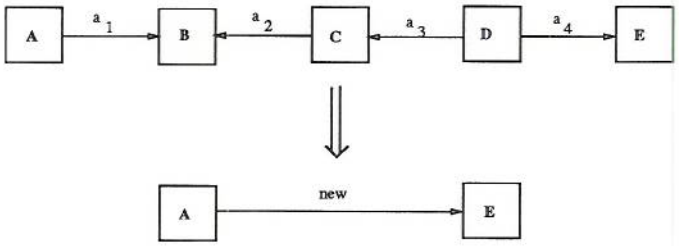


Figure 27: Collapsing a chain of edges into a single edge for optimization, *new* stands for $a_1 \cdot (a_2)^{-1} \cdot (a_3)^{-1} \cdot a_4$. (It is implicitly assumed in this example that the nodes B, C and D are not selected nodes of a GOOD operation).

6.1.5 Read off Result as Early as Possible

This rule is most appropriate in the context of an edge addition or deletion. It is possible to read off the solution when the pattern is reduced to the two nodes involved and an edge between them. Then there is no need for an extra node combination followed by the final projection. For example, if we went back to the example of edge addition of Figure 18, we see that instead of combining nodes C and V in the first step, we could have used the *collapsing heuristic* to remove node V and connect nodes C and S with an edge that is $ISAC \cdot MAN$. Then we observe that the edge to be added is between the nodes C and S, and we stop the translation process. The edge to be added $MAKE$ is just $ISAC \cdot MAN$ and there is no need for projection and recomposing the two edges after the projection. Thus, in the case of edge addition/deletion the pattern translation algorithm can be improved by avoiding the combination of the selected nodes.

6.1.6 Exploit Common Sub-paths when Computing "Projections"

As was already indicated, computing the "projections" on the selected nodes is done by appropriate compositions of the intermediate left and right tag relations obtained in the pattern translation algorithm. There is a unique way to obtain the projections on to each node. However, it is possible that two different nodes can have a common sub-path as can be seen in Figure 15. It is efficient to avoid recomputation of the common sub-path and use the sub-path already computed for one node in computing the path for the other node. This is similar to common subexpression optimization in the relational model. This could be exploited by the query optimizer to minimize the computation.

6.1.7 Syntactic Checking of the Edge/Node Deletion Patterns

In case of node/edge deletion patterns, if the pattern contains a node/edge that does not correspond to any relation in the instance, then it obviously means that the pattern will not create any embeddings in the instance and the instance will not change. In such a case, there is no need to do anything.

6.1.8 Stop as Soon as an Intermediate Result is Empty

As soon as an intermediate result in the pattern translation algorithm is empty, the algorithm can stop because the final result will definitely be empty and there is no need to continue. This can also be used as a heuristic by the query optimizer to check if any sequence of node combinations leads to an empty intermediate result even before starting the algorithm.

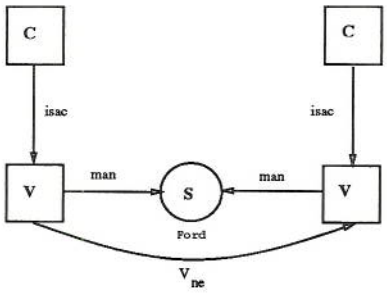


Figure 28: The GOOD pattern with the conceptual V_{ne} relation.

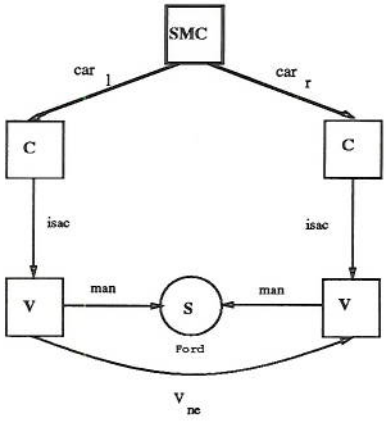


Figure 29: Node addition corresponding to the pattern in Figure 28.

6.2 Semantic Optimizations at the Graph Level

6.2.1 Using the Functionality of Edges

The functionality of edges can be very useful in transforming the pattern into a semantically equivalent pattern that is more amenable to syntactic optimizations. First, consider the pattern of Figure 28. In contrast to Figure 4, here we have introduced an extra edge V_{ne} between the V nodes, which *conceptually* relates different vehicle objects. Assuming that the DBMS has a total order defined on the vehicle identifiers, this *conceptual* relation, therefore consists of the following non-identical pairs of vehicles: $(v_1, v_2), (v_1, v_3), and (v_2, v_3)$. In addition we also consider the DBMS tracking a similar conceptual relation C_{ne} between different cars. Therefore, the pattern in Figure 28 corresponds to the pairs of non-identical cars that have the same manufacturer.

The node addition to be performed in association with the pattern in Figure 28 is shown in Figure 29. Since the pattern corresponds to non-identical cars that have the same manufacturer, the pattern translation algorithm would result[21] in a relation with non-identical pairs of cars with the same manufacturer.[22]

Now, since the C nodes have a functional edge $ISAC$ to the V nodes, this pattern is semantically equivalent to the pattern in Figure 30, where the V_{ne} edge between the V nodes has been replaced by the C_{ne} between the C nodes. This pattern can now be reduced syntactically by collapsing the chain between the C nodes and proceeding with the node addition from there on. The chain can be

[21] We leave out the details here, as they are very similar to the first node-addition example, except that now we also consider the edge V_{ne} in the translation.

[22] This is the second method of removing identical pairs that was alluded to in section 5.2.1.

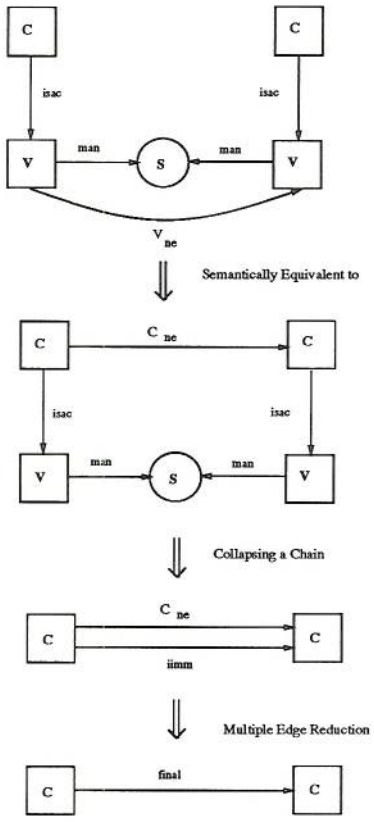


Figure 30: Semantic optimization to make the pattern better suited for syntactic optimization, *iimm* stands for $ISAC \cdot MAN \cdot (MAN)^{-1} \cdot (ISAC)^{-1}$.

reduced by having only one edge *iimm* between the two C nodes instead of the chain and the C_{ne} edge. These two can then be combined by the multiple edge reduction technique as $iimm \cap C_{ne}$. The final result can easily be verified to be exactly the same as was obtained before, only much more easily. Therefore, the semantics of the functionality of edges was exploited to transform the pattern into a pattern better suited for syntactic query optimization.

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[23], like in the relational algebra. We will not talk about these in this paper, but it should be clear that these optimization rules can be exploited at an implementation level to manipulate binary relations after the Tarski expression for the pattern is computed. Also, index information and clustering can be exploited since the binary model is a *decomposed storage model*, which can be beneficial for optimization at a low level (We refer to [4, 5] for a more detailed discussion on this issue.)

As argued by Copeland et.al. (again, see [4, 5]) this approach could also lends itself ideally for implementation on parallel architectures.

[23] As an example, $r \cdot (s \cup t) = (r \cdot s) \cup (r \cdot t)$.

6.3.2 Evaluation of Common Subexpressions

The final Tarski expression corresponding to the pattern in Figure 4 with the naive node combination was

$$H = ((((MAN^{\triangleright} \cdot MAN^{-1})^{\triangleleft} \cdot ISAC^{-1})^{\triangleleft} \cdot ((MAN^{\triangleright} \cdot MAN^{-1})^{\triangleleft} \cdot (MAN^{\triangleleft} \cdot ISAC^{-1})))^{\triangleleft} \cdot ((MAN^{\triangleright} \cdot MAN^{-1})^{\triangleleft} \cdot ISAC^{-1})^{\triangleleft} \cdot (((MAN^{\triangleright} \cdot MAN^{-1})^{\triangleleft} \cdot (MAN^{\triangleleft} \cdot ISAC^{-1}))^{\triangleleft})^{-1})^{id}$$

It has 3 common subexpressions each of which are repeated twice. In the evaluation of this expression, 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.

6.4 A Small Set of Tarski Algebra Operators

The Tarski algebra has only seven basic operators, namely: composition, inverse, union, complementation, left-tagging, right-tagging, and finite-set tagging.

This is clearly a very elegant and small instruction set from a theoretical point of view. However, from an implementation point of view, the addition of carefully chosen derived operators can be warranted on reasons of efficiency. We have found the following set of basic and derived Tarski operators the most frequent and useful and we put it forward as a useful set of primitives to be supported by a storage model such as the DSM. These operators are:

1. Composition ($r \cdot s$),
2. Left- and Right-Tagging ($r^{\triangleleft}$ and $r^{\triangleright}$),
3. Union ($r \cup s$),
4. Intersection ($r \bigcap s$),
5. Difference ($r - s$),
6. Inverse (r^{-1}),
7. Selection ($\sigma_{selcond}(r)$),
8. Ordered-pair Tags (r^{τ}),
9. Relative Universe (r^{un}),
10. New (r^{new}), and
11. Finite-Set Tagging (r^{ν}).

7 Future Research

We like to conclude this paper by pointing out some future research directions.

One definite research direction is a detailed study of query optimization principles (both at the graph and algebraic level) for a platform such as presented here: the decomposed storage model and the Tarski algebra. We expect that many optimization results about relational query languages should carry over to this setting, but we also expect to discover new query optimization techniques

specifically suited for the DSM platform. Furthermore, since the (DSM and Tarski) framework is both suited for relational as well as object-oriented query processing, this study should increase our understanding of object-oriented query optimization, an area in which progress has been more difficult and slower than in that for relational query optimization.

We also believe that our techniques are not just limited to databases represented in the DSM. We don't see any serious difficulty with allowing the management of objects with multiple *data* fields. It is entirely reasonable to consider such fields as semantically related, and thus also physically related. This paper really addresses how the linkage fields (i.e., tags and object-identifiers) can be most usefully created, composed, compared etc. to solve queries effectively in an information environment that can be conceptualized as a graph. (The actual atomic data fields only play an important role in the selection of initial objects relevant to the query).

Finally, we are interested in exploring how to support and implement a storage model such as (DSM and the Tarski algebra) on parallel platforms. The arguments made in [4, 5] suggest that this is an interesting and important research direction.

8 Acknowledgments

The authors would like to thank Marc Gyssens and Ed Robertson for discussions on the Tarski algebra, and Jan Van den Bussche for pointing out the work on the decomposed storage model.

References

[1] S. Abiteboul and V. Vianu. Procedural languages for database queries and updates. *Journal of Computer and System Sciences* 41, 1990, pp. 181–229.

[2] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In *Proc. 1st Int'l Conf. on Deductive and Object-Oriented Databases*, Kyoto, Japan, 1989, pp. 40–57.

[3] F. Bancilhon. Object-oriented database systems. In *Proc. 7th ACM SIGACT-SIGMOD-SIGART Symp. on Princ. Database Systems*, Austin, Texas, 1988, pp. 152–162.

[4] G. Copeland and S. Khoshafian A decomposition storage model. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, Austin, Texas, 1985, pp. 268–279.

[5] S. Khoshafian, G. Copeland, T. Jagodits, H. Boral and P. Valduriez A query processing strategy for the decomposed storage model. In *IEEE Data Engineering*, 1987. pp. 636-643

[6] M. Gyssens, J. Paredaens, and Dirk Van Gucht. A graph-oriented object model for database end-user interfaces. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, Atlantic City, New Jersey, 1990, pp. 24–33.

[7] M. Gyssens, J. Paredaens, and Dirk Van Gucht. A graph-oriented object database model. In *Proc. 9th ACM SIGACT-SIGMOD-SIGART Symp. on Princ. Database Systems*, Nashville, Tenn., 1990, pp. 417–424.

[8] M. Gyssens, L. Saxton, and Dirk Van Gucht. Tagging as an alternative to object creation. Presented at *The Dagstuhl Seminar on Query Processing in Object Oriented, Complex Object, and Nested Relation Databases*

[9] R. Hull and M. Yoshikawa. ILOG: Declarative creation and manipulation of object identifiers. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proc. 16th Int'l Conf. on Very Large Databases*, Brisbane, Australia, 1990.

[10] W. Kim and F.H. Lochovsky, editors. *Object-oriented concepts, databases, and applications.* ACM Press (Frontier Series), 1989.

[11] S. Khoshafian, G. Copeland, T. Jagodits, H. Boral and P. Valduriez. A query processing strategy for the decomposed storage model. In *Proc. 3rd IEEE Int'l Conf. on Data Engineering*, Los Angeles, Cal., 1987, pp. 636–643.

[12] W. Kim, D.S. Reiner, and D.S. Batory. Query Processing in Database Systems. Springer Verlag Publishers, 1985.

[13] D. Maier. The Theory of Relational Databases. Computer Science Press, 1983.

[14] O. Ore. *Theory of graphs.* American Mathematical Society, 1962.

[15] A. Rosenthal and C.G. Legaria Query Graphs, Implementing Trees, and Freely-Reorderable Outerjoins. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, Atlantic City, New Jersey, 1990, pp. 291–299.

[16] L.M. Haas, J.C. Freytag, G.M. Lohman and H. Pirahesh Extensible Query Processing in Starburst IBM Almaden Research Center, San Jose, California

[17] M. Stonebraker, editor. *Readings in database systems.* Morgan Kaufmann Publ., 1988.

[18] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6, 1941, pp. 73–89.

[19] A. Tarski and S. Givant. *A formalization of set theory without variables.* American Mathematical Society, Providence, Rhode Island, 1986.

[20] J.D. Ullman. Principles of Database and Knowledge-Base Systems, Volume II. Computer Science Press, 1989.

[21] W. Wong and K. Youseffi. Decomposition-a strategy for query processing. *ACM Transactions on Database Systems*, Vol. 1, No. 3, pp. 223–241.

[22] S.B. Zdonik and D. Maier, editors. *Readings in object-oriented database systems.* Morgan Kaufmann Publ., 1989.

A The Tarski Algebra

Assume that we have an enumerable set V of basic values. A *binary relation* over V is a finite subset of $V \times V$. If r is a binary relation, we denote by $r\downarrow$ the smallest subset of V such that $r \subset r\downarrow \times r\downarrow$, i.e., $r\downarrow = \{v \in V \mid \exists w \in V : (v,w) \in r \vee (w,v) \in r\}$. The set $r\downarrow$ will be called the *active domain* of r. Similarly, if $r_1, \ldots, r_m$ are relations, then $(r_1, \ldots, r_m)\downarrow = r_1\downarrow \cup \ldots \cup r_m\downarrow$ is the active domain of the database consisting of the mathematical relations $r_1, \ldots, r_m$.

Definition A.1 *Let r and s be relations. The four operators of the basic Tarski algebra are:*

- *The* inverse *of r, denoted r^{-1}, is the relation $\{(v,w) \mid (w,v) \in r\}$.*
- *The* complement *of r, denoted $\overline{r}$, is the relation $\{(v,w) \mid (v,w) \in r\downarrow \times r\downarrow \wedge (v,w) \notin r\}$.*
- *The* union *of r and s, denoted $r \cup s$, is the relation $\{(v,w) \mid (v,w) \in r \vee (v,w) \in s\}$.*
- *The* composition *of r and s, denoted $r \cdot s$, is the relation $\{(u,w) \mid \exists v \in V : (u,v) \in r \wedge (v,w) \in s\}$.*

B Proof of Correctness of the Node Combination Technique

Consider the pattern shown in Figure 31.

This is equivalent to the following expression:

$$a(A,B) \wedge b(A,D) \wedge c(C,A) \wedge d(B,E) \wedge e(F,B) \wedge f(G,C)$$

If we combine nodes A and B connected by edge a using the node combination technique, the resulting pattern is as shown in Figure 32.

This is equivalent to the following expression:

$$f(G,C) \wedge a^{\triangleleft} \cdot c^{-1}(Q,C) \wedge a^{\triangleleft} \cdot b(Q,D) \wedge a^{\triangleright} \cdot d(Q,E) \wedge a^{\triangleright} \cdot e^{-1}(Q,F)$$

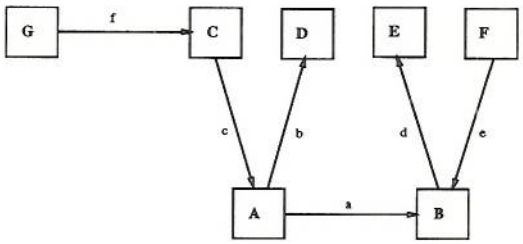


Figure 31: Example pattern.

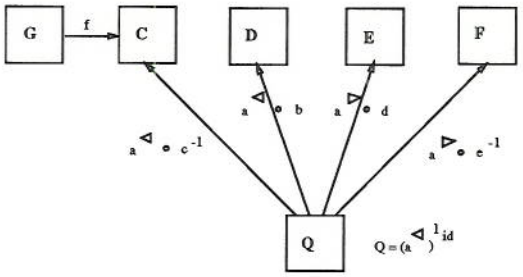


Figure 32: Pattern after node combination of nodes A AND B.

Notice that Q is a variable name occurring in 4 conjuncts. By definition of composition, the above expression is equivalent to

$\exists A_1 \exists A_2 \exists B_1 \exists B_2$ $[f(G,C) \wedge a^{\triangleleft}(Q,A_1) \wedge c^{-1}(A_1,C) \wedge a^{\triangleleft}(Q,A_2) \wedge b(A_2,D) \wedge a^{\triangleright}(Q,B_1) \wedge d(B_1,E) \wedge a^{\triangleright}(Q,B_2) \wedge e^{-1}(B_2,F)]$

This is equivalent to

$\exists A_1 \exists A_2 \exists B_1 \exists B_2$ $[f(G,C) \wedge a^{\triangleleft}(Q,A_1) \wedge c(C,A_1) \wedge a^{\triangleleft}(Q,A_2) \wedge b(A_2,D) \wedge a^{\triangleright}(Q,B_1) \wedge d(B_1,E) \wedge a^{\triangleright}(Q,B_2) \wedge e(F,B_2)]$

Consider the conjunction

$$a^{\triangleleft}(Q,A_1) \wedge a^{\triangleleft}(Q,A_2) \wedge a^{\triangleright}(Q,B_1) \wedge a^{\triangleright}(Q,B_2)$$

that is part of the previous expression. By the definition of $a^{\triangleleft}$ and $a^{\triangleright}$, this conjunction is equivalent to one of the two simple conjuncts, $a(A_1,B_1)$ or $a(A_2,B_2)$. Since the Q node is common, we can deduce that

$$A_1 = A_2$$

and

$$B_1 = B_2$$

Therefore, after renaming we get that, this is equivalent to $a(A,B)$. Therefore, the original conjunct is equivalent to the conjunct after the node combination.