Query Optimization for KBMSs: Temporal, Syntactic and Semantic Transformations

Thodoros Topaloglou [*]
Dept. of Computer Science
University of Toronto
Toronto, M5S 1A4, Canada

Arantza Illarramandi [†]
Facultad de Informatica
Universidad del Pais Vasco
20080 San Sebastian, Spain

Licia Sbattella [‡]
Dipartimento di Elettronica
Politechnico di Milano
20129 Milano, Italy

Abstract

This paper describes a framework for query optimization for KBMSs based on the knowledge representation language Telos. The developed framework involves temporal and syntactic simplifications and semantic modification of the queries. Temporal simplification attempts to select parts of a knowledge base that are relevant to a query from a temporal viewpoint. Syntactic simplification exploits structural properties of the data model and attempts to transform the query into an equivalent and more efficient one. Semantic transformation uses knowledge specific to the application domain i.e, integrity constraints and rules, in order to transform a user specified query into another form which gets the same answer and it is processed efficiently. The above three steps are integrated into a global algorithm for query optimization, that utilizes all features of the considered KBMS.

1 Introduction

As the number and size of deployed knowledge bases grows, so will the need for management tools, techniques and methodologies. Knowledge Base Management Systems (hereafter KBMSs) have been proposed as generic systems offering facilities for building, accessing and managing large knowledge bases [BM86], [ST89]. KBMSs are expected to support a state-of-the-art knowledge representation scheme, inference mechanisms for deductive as well as non-deductive reasoning, knowledge management tools for knowledge acquisition, validation and documentation. In addition, KBMS are supposed to be efficient to the point of supporting **large** knowledge bases concurrently accessed by multiple users.

This paper focuses on a particular desirable feature for KBMSs: query optimization. This feature requires that a KBMS **plans** how each query with respect to the database is to be processed. This planning is supposed to take into account properties of the knowledge base (e.g. number of instances per class, the IsA hierarchies, etc.) as well as its physical implementation, among others. Query optimization problems have been extensively studied in databases [JK84], [Fre89]. There is virtually no corresponding work for KBMSs.

In general, query processing is considered a highly complex task requiring two main phases. Initially, the submitted query is specified in a non-procedural language describing the conditions that its answer must satisfy. Then, the query is transformed through syntactic and semantic transformations to an equivalent query with minimized cost, according to a given cost function (*query modification phase*). Subsequently, the query is transformed into a procedural program, taking into account physical data structures, quantitative information and the like, which is then executed (*access planning phase*).

In KBMSs the nature and basic structure of query optimization remain the same. However, the differences between knowledge representation schemes such as BACK [PSKQ89], or Telos [MBJK91] and data models introduce new intricacies to the problem and new requirements for its solution. Firstly, the contents of the knowledge base may include (deductive) rules as well as complex object descriptions. Secondly, some of the assumptions under which query processing is performed in DBMSs, such as the closed world assumption, need not carry over for KBMSs when the considered knowledge bases are incomplete. Thirdly, the knowledge management tools that need to be supported by a KBMS (temporal query processing, consistency checking, explanation facilities, etc) add a set of requirements on query optimization beyond those in place for DBMSs.

To make the discussion more concrete, the paper only addresses the query modification phase and offers a framework for accomplishing it for a KBMS based on the knowledge representation language Telos [MBJK91]. The features of the language include structuring facilities for knowledge bases (including generalization, classification, aggregation), deductive rules and declarative integrity constraints, facilities for representing and reasoning with temporal knowledge, classification facilities extended to attributes as well as the ability to define attribute meta-classes.

Among the novelties of the framework described below, we note the formulation of a temporal simplification scheme, the reformulation of syntactic simplification for a full-fledged knowledge representation language and the adoption of semantic transformations for the same. All subphases of the proposed framework are integrated into

[*]Supported by the Dept. of Computer Science, University of Toronto, and the Ontario ITRC

[†]Supported by the Spanish Goverment Grant PF 88 15939031, and the Ontario ITRC

[‡]Supported by the Grant 203.07.16, Consiglio Nazionale delle Ricerche, Italy

a modular query optimizer.

The rest of this paper is organized as follows. Section 2 presents the global picture of the proposed approach. Section 3 describes some features of the underlying knowledge representation language and the syntax of the query language. Sections 4,5,6 describe the temporal, syntactic and semantic transformation algorithms. Section 7 presents some preliminary experimental results of our algorithm. Section 8 summarizes the presentation and discusses further research.

2 The Approach

The approach for query optimization that is taken here is based on the principle of *specializers*. This concept is known both in the artificial intelligence community in the context of reasoning (reasoning with specialists [MS88], theory resolution [Sti85], hybrid reasoning [BGL85]) and in the database community with the specialized query processors in the next generation DBMSs such as PROBE [DBG+85] or application specific interfaces in the extensible DBMS architectures such as DASDBMS [PPS+87]. The basic idea is to break down the query optimization task into subtasks which can be carried out either sequentially or in parallel by dedicated procedures. This becomes feasible when the knowledge representation language supported by the KBMS integrates several different sub-theories (e.g., one for generalization, one for time) for which reasoning can be achieved through specialized inference engines.

The queries that we intend to process are (a) temporal queries (both historical and rollback [Sno87]), (b) queries about the structure and the data of the knowledge base and (c) queries about the world that the knowledge base models. The above classes of queries make use of the temporal, structural and assertional components of Telos. Obviously, each class of queries needs a different treatment. Not bothering about notation yet, let us see how the defined simplification steps would work in the following query:

"Find the addresses of all the tutors who worked at the University of Toronto during September 1989, according to what the system believed on October 15th of the same year."

In relation to the temporal aspects of the query one can check if the class **Tutor** has non-empty extension during the specified temporal periods. Also, one can select deductive rules and integrity constraints which involve tutors, universities, address information etc. and are effective on the time interval the query is asked. These rules will be used later for semantic optimization. The component of the query optimizer which deals with the temporal aspects of the query is called *temporal simplification component*.

Assuming that the class **Tutor** is defined as specialization of the class **GStudent** and any instance of the former is also an instance of the latter, then the above query can be reformulated as:

"Find the addresses of all the tutors which are also gradstudents and.."

This modification to the query increases the possibilities for further transformations in the sequel. Such type of modifications are based on structural properties of the data model and the syntactic aspects of the query language. The component of the query optimizer which handles such modifications is called *syntactic simplification component*.

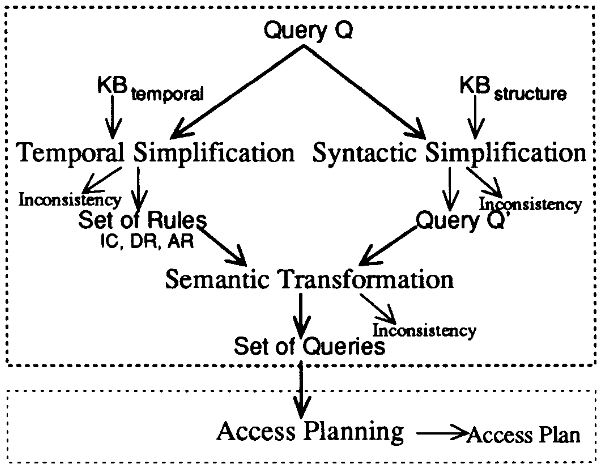


Figure 1: Query Optimizer Structure

Another source for query transformation is domain knowledge encoded in the form of rules and integrity constraints. Assume now that the knowledge base contains a rule saying that *"all graduate students have as address the address of their university"*, then we can further simplify the query. The combination of the query with this rule results in the following formulation:

"The address of all the tutors who worked at the University of Toronto during September 1989, according to what the system believed on October 15th of the same year, is same as the address of the University of Toronto."

The latter query saves accesses for finding the address of each tutor by assigning to all of them the same address which is retrieved in one step from the knowledge base. This component of the query optimizer is called *semantic transformation component*.

We identified three components of the query optimizer for KBMSs. All of them are parts of the query modification phase in the optimization process. So far, all the optimization effort is independent of data structures and implementation techniques. The next component that completes our query optimization algorithm is the *access planning* component. This component gets as input all possible alternative queries obtained by the query modification phase, analyzes them and estimates the execution cost of each one and eventually generates an executable form of the minimum cost query. This component is not included in this presentation. Figure 1 puts all parts together, outlining in this way the global algorithm for query optimization for KBMSs.

3 Highlights of Telos

In this section we present an example Telos knowledge base and emphasize on those features of the language which are related in the discussion to follow. A detailed description of the Telos knowledge representation language appears in [MBJK91].

The following **TELL** statements define the simple classes **Person, Professor, GStudent, Tutor, School** and **Address**. The classes **Professor** and **GStudent** are specializations of class **Person**, and **Tutor** specalization of **GStudent**. Persons have **name, age** and **address** as attributes. Professors may have the same attributes with an additional restriction on the value of the **age** attribute. The types of attribute values are primitive (e.g. **String**) or complex (e.g. **Address**) or specified by integer ranges (e.g. **0..100**).

The deductive rule in class **GStudent** asserts implicit knowledge to the knowledge base by saying that the address of the graduate students is their school address. Telos also allows integrity constraints; these are logical formulas which should be entailed by the knowledge base. Both deductive rules and integrity constraints are expressed in a typed first order language.

The keyword **single** in attribute **school** of class **GStudent**, states that graduate students have to be registered at most at one school. Telos allows one to constrain the definition of attributes (**school** is singlevalued) by grouping attributes in attribute meta-classes and adding to them special semantics. In the rest of this paper attribute meta-classes are referred as attribute restrictions.

```
TELL CLASS Person            TELL CLASS Professor
  IN S_Class                   IN S_Class
  WITH                         ISA Person
    attribute                  WITH
      name:String                attribute
      age:0..100                   age:25..60
      address:Address          END Professor
END Person

TELL CLASS School            TELL CLASS Address
  IN S_Class                   IN S_Class
  WITH                         WITH
    attribute                    attribute
      address:Address              street:String
      dean:Professor               number:Integer
END School                         pcode:String
                               END Address

TELL CLASS GStudent          TELL CLASS Tutor
  IN S_Class                   IN S_CLASS
  ISA Person                   ISA GStudent
  WITH                       END Tutor
    attribute
      advisor:Professor
    attribute, single
      school:School
    deductiveRule
      DR1:  $(∀x/GStudent,∀s/School,∀a/Address,
      ∀t1,t2,t3,t4,t5,t6/Time)(in(x,GStudent,t1,t2)
       ∧ school(x,s,t3,t4) ∧ address(s,a,t5,t6) ∧
      (t1 * t3 overlaps t5) ∧ (t2 * t4 overlaps t6) ⟹
      address(x,a,t5,t6))$
END GStudent
```

In many applications, it is desirable to represent the progression of states of a domain over a period of time in a way that enables the system to answer historical queries not only with respect to its current state but also with respect to a previous state [1]. Telos adopts a modified case of Allen's time framework [All83] in order to represent *historical knowledge* about the domain. Telos also records the progression of *system's beliefs* about the *history* of the domain. When the **TELL** statements are processed, apart from the information contained in these statements, the system will also record that it believes that information from the time this transaction committed and on.

Queries in Telos are expressed through the **ASK** operation [TK89]. The syntax of **ASK** operation is:

$$\begin{array}{l}\texttt{ASK } x_1/S_1, \ldots, x_n/S_n \; : \; \texttt{W} \\ \quad \texttt{ON } t_1 \\ \quad \texttt{AS OF } t_2\end{array}$$

where $x_1, \ldots, x_n$ are target variables for which we want values to be returned; $S_1, \ldots, S_n$ are either set expressions or classes and denote the range of the variables; t_1, t_2 are conventional dates and **W** is a formula of Telos assertional language.

The temporal subexpression in the query formula, has the following meaning: *answer the query W at the historical interval which is over t_1 using only the part of the KB which is believed over the interval t_2.*

4 Temporal Simplification

Temporal simplification attempts to identify those parts of a knowledge base that are relevant to a query from a temporal viewpoint [JK89]. Temporal simplification consists of the following three steps:

1. Check for inconsistent or redundant temporal constraints in the query expression;

2. Check whether there are available data for all the target variables of the query, for the *historical* and the *belief* period that the query refers to;

3. Select the part of the knowledge base which is involved in the answering of the query from a temporal viewpoint. This step has meaning only for the deductive rules, the integrity constraints, and for the restrictions on attributes expressed through built-in meta-attributes.

4.1 Consistency of Temporal Constraints

There may be cases where many temporal constraints referring to the same event. The query below is such an example:

```
ASK x/D:
   (∃t1,t2/Time)(age(Person,x,t1,t2)                  ∧
during(t1,1980))
   ON 1987
   AS OF 1987/12
```

Before processing, the query is transformed into an internal form called *full form*. In this form the temporal part from the tail of the query is factored in. The full form of our example query considering only the history time, is:

```
return x:
   in(x,D,t3) ∧ over(t3,1987) ∧ memberOf(x,age.Person,t1)
   ∧during(t1,1980) ∧ over(t1,1987)
```

This means that the same event, i.e. the membership of **x** in **age.Person** set is constrained by two different

[1]Recently, this feature has received considerable attention in the area of temporal and deductive relational databases [Sno87], [Spi88].

temporal constraints, which in fact contradict each other (e.g., `[during, 1980]` and `[over,1987]`). Another case is the one where two (or more) constraints do not contradict but the one implies the other, e.g. `C1=[over,1987]` and `C2=[over, 1980..1990]`. In this case the time period which is determined by `C1` is part of the time period which is determined by `C2`, obviously `C1` is redundant. The temporal simplification component detects such cases and eliminates the redundancies.

Computationally, the checking for contradiction or implication between two temporal constraints of this form is a cheap operation. This is done using the *addition* operation that is defined in [All83] for the interval representation and redefined in [VK86] for the equivalent point representation. This operation involves a lookup on the *addition table* (see [TK89]) and returns "0" if the two constraints do not interfere and the longer temporal period (the most general constraint) otherwise.

4.2 Checking for Empty Answers

Sometimes, the classes referenced in a temporal query may not have instances in the knowledge base for the time periods that the query is asked. Then, it is obvious that their answer is the empty answer.

The checking for temporal validity of the queries, requires to maintain some simple form of meta-information which is acquired during the `TELL` operation. This meta-information must be able to support schema-queries of the form:

1. Does class `C` have instances in the KB at the historical (resp. belief) period `HP` (resp. `BP`)?
2. Find the deductive rules (resp. integrity constraints) that are relevant from a temporal viewpoint, in the KB for the historical period `HP` and the belief period `BP`?

To support this type of schema-queries, we use a data structure called *temporal dictionary* and store for each class the longest temporal period for which the class has instances, i.e.:

Class	HP	BP
class1	hp1	bp1
class2	hp2	bp2
...	...	...

The problem of checking if there are available data in the knowledge base for the interesting temporal period is now reduced in finding if there is an overlap between the time period specified in the query and a time period from the above table.

We define the point representation of a temporal period, to be a conjunction of Iq terms, where I is an integer and q may be either ">" or "<", from now on this is called *duration*. Iq terms in a duration are sorted on the value of I (e.g. D={9<,31>,37<}). Two successive terms, I_1q_1, I_2q_2 compose a *closed term* $[I_1, I_2]$ iff $I_1 < I_2$ and q_1 is ">" and q_2 is "<", (e.g. [31,37]). An Iq term is called *left open* (resp. *right open*) iff q is "<" (resp. q is ">") and I less (resp. greater) than any other I_x in D. Eventually, D is normalized in a triple of a left open, a closed and a right open term (some of them may be null). It is easy to see that two durations D_1 and D_2 *overlap* if at least one of their terms overlaps. A constant time algorithm which tests the terms' overlapping is very straightforward.

The next example illustrates the checking for an empty answer of a query.

Query: *Find all the tutors of the department of computer science on August of 1988 as the knowledge base knew at September of 1989.*

The analysis of the query from a temporal point of view will result in the following information:

variable	class	HP	BP
x	Tutor	$I(1.8.88) > \wedge$ $I(31.8.88) <$	$I(1.9.89) > \wedge$ $I(30.9.89) <$

Each line of the formed table (here there is only one) forms a schema-query and then it is answered using the previous table. The schema-queries that are formed, are:

`Q1a(Tutor,(`$I(1.8.88)>, I(31.8.88)<$`)).` (S1)
`Q1b(Tutor,(`$I(1.9.89)>, I(30.9.89)<$`)).` (S2)

the predicates `Q1a` and `Q1b` denote the type of the query according the types of the schema-queries we specified at the beginning of this paragraph. `S1, S2` are `true/false` queries and answered as described above, e.g. `Q1a` is `true`, if $I(1.8.88)>, I(31.8.88)<$ overlaps with `hp1`. A query is valid if all of the created schema-queries are answered as `true`.

4.3 Temporal Selection

Deductive rules (DRs) and integrity constraints (ICs) in Telos are specified through the attribute facility. Consequently, a temporal constraint of the form (HP, BP) is associated with any of them. The next example illustrates the temporal constraints which constrain the deductive rule `DR1`.

```
DR1 exists during 1988/2..Now       (T1)
DR1 believed costarts 1988/1/10     (T2)
```

The two orthogonal temporal intervals define a rectangular area which determines the validity of the rule in the two-dimensional space defined by the history and belief time. This allows us to index rules and constraints with respect to history and belief time using a spatial access method such as the R-tree [Gut84]. The query's history and belief time define a search window. The spatial search which is performed by the R-tree returns all the DRs and ICs whose rectangle intersects with the query window. Once these rules are returned, we further select those that refer to same classes that the query refers. We call this operation temporal selection of rules and constraints.

With the temporal selection we restrict the set of rules and constraints which are involved in the query optimization, to only those that are relevant to the query[2]. Therefore temporal selection provides to the semantic query optimizer fewer ICs and DRs to work with. If the temporal selection does not find ICs and DRs that are effective for time periods that a query is asked to the KB, then there is not any reason to apply the next steps of the query optimization.

4.4 Temporal Simplification Algorithm

The three steps of the temporal simplification are summarized in the following algorithm:

algorithm1: temporal query simplification
input: the user submitted query, `Q`
output: "empty answer" if temporal contradiction is discovered, otherwise an equivalent query Q_t (redundant temporal constraints have been removed) and a set of

[2]Relevant DRs and ICs are these ones that refer to the same KB object as the query refers and for the same temporal periods

```
selected rules and constraints, RB.
begin
  step1: /* temporal simplification */
    for each variable x in Q do
      T(x) is the set of temporal constraint
      restricting x
      if addition(T(x))=∅,
        return("empty answer")
      otherwise, T(x)=addition(T(x))
    endfor
  step2: /* checking for empty answer */
    for each class C in Q do
      formulate schema queries:
      Q1a(C), Q1b(C)
      /*answer queries using the temp. dictionary
      and the definition of overlap from section 4.2*/
       if (answer(Q1a(C))=false or
        answer(Q1b(C))=false,
        return("empty answer")
    endfor
  step 3: /*temporal selection */
    let W=[HP,BP], where HP,BP the
      temporal arguments of Q
    RB=search-R-tree(overlap, W)
    return(RB)
end ◇
```

The procedure *addition(L)* takes a list L of temporal constraints and "adds" them one by one using the Allen's addition table (see section 4.1). The procedure *answer(Q)* takes a schema query and answers it against the temporal dictionary (see section 4.2). The procedure *search-R-tree* returns those rules and constraints whose temporal validity rectangle overlaps with the query window.

5 Syntactic Simplification

Syntactic query optimization exploits special properties of query languages to transform a query into another one which has the same answer and can be processed more efficiently [KRB85]. These properties are expressed usually by rules. In our case, the properties of the query language that we base the syntactic simplification are related to the structural features of Telos.

Our approach to syntactic query optimization attempts: First, to simplify the query expression by applying transformation rules which allow detection of those predicates which are always true, or always false, or inconsistent with respect to the knowledge base. Second, to transform the query in a form which is more convenient to be used in the next steps of query optimization. Consequently, the proposed syntactic simplification algorithm works in two steps: The first step deals with the simplification based on structural properties of the query language. The second step considers the opportunities of simplification on predicates involving numerical terms and comparison operators.

At the beginning of the syntactic simplification process the query have to be: (a) safe [3], (b) correctly typed, (c) expressed in Prenex Disjunctive Normal Form and (d) simplified with respect to the idempotency rules involving logical expressions [JK84], [DST80] and to the rules involving set expressions and arithmetic expressions appearing in terms.

During the syntactic simplification process, the syntactic aspects of the query are represented as a labelled graph. *Nodes* represent the constant and variable terms. *Edges* represent the atomic formulas. If the query expression contains disjunctions, i.e. $P_1 \vee P_2 \vee \ldots \vee P_n$, where $P_1, P_2, \ldots P_n$ are conjunctions of predicates, then a separate graph is built for each P_i. For each negated predicate a graph is produced considering its "affirmative" part. [4] The negation is considered after the simplification process.

5.1 Syntactic Simplification using Structuring Constructs

In this section, we describe transformation rules that will be used in the syntactic simplification which are based on the structural properties of the query language. These rules also take a graph representation similar to the one of the queries.

The set of transformations rules presented in the rest, is divided in two groups: *completion rules* and *simplification rules*.

5.1.1 Completion Rules

These rules are going to be used for the completion of the query. Essentially, they represent properties of concepts, as specialization, instantiation, set membership, etc. The new information that is added is used either in inconsistency detection or query simplification.

No	Name	Expression
R1	IsA transitivity	isA(C1,C2,t1,t2) ∧ isA(C2,C3,t3,t4) ⇒ isA(C1,C3,t1*t3,t2*t4)
R2	Subset transitivity	subsetOf(S1,S2,t1,t2) ∧ subsetOf(S2,S3,t3,t4) ⇒ subsetOf(S1,S3,t1*t3,t2*t4)
R3	Instance of subclasses	in(X,C1,t1,t2) ∧ isA(C1,C2,t3,t4) ⇒ in(X,C2,t1*t3,t2*t4)
R4	Structural inheritance	isA(C1,C2,t1,t2) ∧ P(C1^n,t3,t4) ∧ Q(C2^n,t5,t6) ⇒ isA(C1^n,C2^n,t1*t3*t5,t2*t4*t6)
R5	Member of a subset	memberOf(x1,S1,t1,t2) ∧ subsetOf(S1,S2,t3,t4) ⇒ memberOf(x1,S2,t1*t3,t2*t4)
R6	Specialization of disjoint classes	disjoint(C1,C2,t1,t2) ∧ isA(C3,C1,t3,t4) ⇒ disjoint(C3,C2,t1*t3,t2*t4)

Table 1: Completion Rules

In the presentation of the rules we use the following notation: Symbols C1, C2, C3 represent different classes; S1, S2, S3 represent different sets; t1, t2, ..., t12 represent time intervals; x1, x2 denote members of sets and X, Y denote instances of classes. Lastly,

[3] A temporal query is unsafe with respect to time if it contains a temporal constraint of the form [θ t] and t is unbound.

[4] See [IS90] for details about the graphical notation of the queries.

$t_k = t_i * t_j$ is defined as the common time interval between two time intervals t_i, t_j in case they overlap, and it is undefined if they are disjoint. The full set of completion rules is listed on table 1.

5.1.2 Simplification Rules

The objective of the simplification rules is:

(a) to obtain a simplified representation of the query, eliminating redundant information;

(b) to detect inconsistent queries.

The full set of simplification rules is listed on table 2.

No	replace	with	condition
R7	`isA(C1,C1,t1,t2)`	`True`	
R8	`subsetOf(S1,S1,t1,t2)`	`True`	
R9	`isA(C1,C2,t1,t2)` ∧ `isA(C2,C1,t3,t4)`	`False`	`t2*t4`, `t1*t3` are defined
R10	`subsetOf(S1,S2,t1,t2)`∧ `subsetOf(S2,S1,t3,t4)`	`False`	`t2*t4`, `t1*t3` are defined
R11	`disjoint(C1,C2,t1,t2)`∧ `disjoint(C1,C2,t3,t4)`	`disjoint(C1,C2,t5,t6)`	`t5 = t1 + t3` and `t2 * t6` is defined
R12,R13,R14		see fixing-level rules	
R15	`in(X,C1,t1,t2)` ∧ `in(X,C2,t3,t4)` ∧ `disjoint(C1,C2,t5,t6)`	`False`	`t1*t3*t5`, `t2*t4*t6` are defined
R16	`isA(C1,C2,t1,t2)`∧ `disjoint(C1,C2,t3,t4)`	`False`	`t1*t3`, `t2*t4` are defined

Table 2: Simplification Rules

Fixing-level rules. Since Telos organizes the propositions in a multilayer hierarchy (e.g., tokens are at level 0, simple classes are at level 1, metal-classes at level 2, etc.), a level constraint is associated to any of **`isA`**, **`in`** and **`disjoint`** predicates. Rules R12-14 detect inconsistencies with respect to these constraints. A description of rules R12-14 is presented in [IS90].

5.1.3 Syntactic Simplification Algorithm

The syntactic simplification algorithm takes as input the user submitted query `Q` and either aborts the query if a contradiction is found or returns as output an equivalent query `Q'` (`Q'` returns same answer as `Q`). Initially, the algorithm builds a query graph which corresponds to the query. Then, for each connected component of the graph applies three simplification steps.

algorithm2: syntactic query simplification algorithm
input: the user submitted query, `Q`,
output: aborts `Q` if a contradiction is found, otherwise returns an equivalent query (returns the same answer as `Q`), `Q'`.

```
begin
   form the graph that corresponds to the
   query,
   for each connected component of the
      graph do
      1a. for all isA, instanceOf and
          disjoint predicates of query, apply
          respectively the simplification rules:
          R12, R13 and R14 to fix the levels.
      1b. apply successively the simplific.
          rules R7, R8 to eliminate redundancies
      2. complete the graph, applying rules:
          R1, R2, R3, R4, R5 and R6
      3. apply the simplification rules:
          R9, R10, R11, R15 and R16
   end for
   formulate a new query Q' using the new
   graph
end ◇
```

The simplification algorithm always terminates in a finite number of steps. The reason is that all of the three steps which are applied for any connected component of the graph terminate in finite time. At step 1, the application of rules R12, R13 and R14 on query `Q` results in **`False`** or `Q'`, where `Q'` is the same with `Q` with the instantiation levels checked and fixed; Rules R7 and R8 exclude conjuncts which are always **`True`**. Step 2, works iteratively until no new predicates (edges) are added to the subquery (graph component). The number of iterations is bounded by the number of edges times the number of nodes of the subgraph. Lastly, at step 3 the only effect of the application of the rules is to make certain predicates of the subquery false and perform the appropriate absorptions; this takes also finite number of steps. Therefore, it always terminates.

5.2 Syntactic Simplification using Comparison Operators

During the simplification process the parts of the graphs involving the <comparison operator> edges and their heads and tails can be considered with respect to the semantics of the comparison operators applied to terms of numeric types. Inconsistencies can be detected using the algorithm proposed by [RH80]. In fact, the single subgraphs represent a conjunctive predicate involving the comparison operators $=$, $>$, $<$, $\leq$, $\geq$.

6 Semantic Transformation

In semantic query optimization we use domain knowledge in the form of deductive rules, integrity constraints and attribute restrictions as they were selected during the temporal simplification phase and intend to (a) transform the query into another query (or queries) which has the same answer and can be processed more efficiently, or (b) detect an inconsistent query.

There are two approaches to the semantic query optimization problem for databases. The first one is motivated by database theory (e.g., functional dependencies and inclusion dependencies) [Jar86]. The second approach applies resolution methods in order to transform the query [HZ80], [Kin81], [CGM88], [DIB88], (using integrity constraints and rules). Our method combines both of them and enhances them taking into account the new features offered by the KBMSs.

Briefly, our method proceeds as follows: It takes as input the set of relevant deductive rules, integrity constraints and attribute restrictions (hereafter *rule base* (RB)) which is returned from the temporal simplification algorithm and the query form which is returned from the syntactic simplification algorithm. Then it transforms the query using (a) resolution techniques and (b) database theory based transformations exploiting the attribute restrictions.

The rule base which is produced by the temporal simplification is small in size and depends on the query being optimized. Earlier semantic optimizers in databases [HZ80], [Kin81] work with larger rule bases which in fact are the same for all queries. The small size of rule base affects the performance of the next steps of the algorithm because the search space for the query transformations is seriously reduced. In this presentation we discuss only the resolution based transformations. The rule base in this case contains only ICs and DRs and is called *Domain Dependent Rule Base (DDRB)*. The resolution method that we use is a variation of Stickel's theory resolution.

6.1 Theory Resolution based Search Strategy

The goal is to obtain a set of queries which are semantically equivalent to the given query using partial narrow theory resolution.

Theory resolution [Sti85] is a method which integrates specialized reasoning procedures in a resolution theorem prover. Consequently, reasoning is divided in two subtasks: (a) special cases such as reasoning about taxonomic or temporal information which are handled efficiently by specialized reasoning procedures, and (b) a general reasoning procedure which is based on resolution and controls the theorem proving. Theory resolution is in general more efficient than classical resolution because it decreases the length of refutations and the size of the search space.

6.1.1 Specialized Reasoning Procedures

Procedure for Taxonomic Information. As the Telos query processor is designed [TK89], the structural information guides the query processing steps. Instantiation and specialization axioms are treated as forward chaining rules, in order to reduce the deductions steps any time that the **in** or **isa** predicates are invoked. Therefore, if A is an instance of a class C, after the forward chaining of the specialization axiom it becomes an instance of any of the C's superclasses. Obviously, integrity constraints and deductive rules defined in the superclasses of C are now relevant to A.

Having this information already computed the taxonomic reasoning procedure's job is to locate and retrieve rules and constraints related to the query. Note that our goal here is not to find an answer for the query but to select constraints and rules from the knowledge base which reference entities that exist in the query formula.

Procedure for Inequalities. In our case, deductive rule and integrity constraint expressions may involve an inequality subexpression. Moreover, an inequality subexpression may also appear in the user's query. The goal of this reasoning procedure is to prove that the query's inequality subexpression either contradicts with or implies the inequality subexpressions of the rules and the constraints. The algorithm that we use is similar to the algorithm in [RH80].

Procedure for Temporal Information. The problem here is to check the compatibility between the temporal predicates appearing in the rules and the ones appearing in the query. Algorithms for this purpose were discussed in section 4.

6.1.2 The General Reasoning Strategy

For the second part of the theory resolution method, the general reasoning strategy, we use the input level saturation strategy. *Input* means that it always resolves only with the input rules. The input set is composed of the query and a set of rules corresponding to assertions defined in the rule base related to the query. *Level Saturation* means that at each step it derives all the information of level I before deriving any from level I+1. This strategy is not complete, [5] but performs better when compared to other variants of the saturation resolution. For more details see [IB90].

The semantic transformation algorithm takes as input the DDRB (i.e. a set of deductive rules and integrity constraints selected during temporal simplification) and a query Q, and returns either "empty answer" or a set of equivalent queries. The algorithm progresses as follows:

```
algorithm3 semantic simplification using theory resolution strategy.
input:
    1. DDRB: the set of rules (IC, DR)
    selected during temporal simplification
    2. Query Q
output: "empty answer" or a set of
        equivalent queries
begin
    j=1
    $B_1$=refine(DDRB,Q)
    $K$= DDRB $\bigcup$ Q
    while (Bj $\neq$ { }) and (Bj does not
        contain the empty clause) do
        j=j+1
        $B_j$=refine(K,$B_{j-1}$)
    endwhile
    n=j
    if $B_n$ contains the empty clause then
        return empty answer
    else
        return $\bigcup_{i=1}^{j} B_i$ /*set of equivalent queries*/
    endif
end ◇
```

The *refine* procedure takes as input two sets of clauses S1 and S2 and returns a third set of clauses S if and only if for each C1 in S1 and C2 in S2, if C is the resolvent of C1 and C2 then C is in S. S is called *refinement* of S1 and S2 and it actually summarizes many resolution steps. The tricky part of the algorithm is its unification procedure. It does not unify literals but clauses which contain taxonomic, temporal and inequality parts. Then, it sends parts of the resolvent to the specialized reasoning procedures.

[5] Completeness is not a requirement in an optimization problem. The case here is that the more information we can derive, the bigger possibilities we have to obtain a better execution plan, but there is not any requirement to obtain all the possible information.

The next example illustrates the algorithm using the example query of section 2. The notation `F[C1,C2]` denotes the resolvent of `C1` and `C2`, and the boldface **B**s are refinement sets.

Query: *Find the addresses of all tutors worked at the university of Toronto during 1989.*

Q: **return x**:(∃y)(∃t1,t2,t3)
    in(y,Tutor,t1)∧during(t1,1989)∧
    school(y,UofT,t2)∧during(t2,1989)∧
    address(y,x,t3)∧during(t3,1989)). [6]

We assume that there is not any rule or constraint defined in the scope of class `Tutor`. Therefore, the temporal simplification component does not return anything. However, it checks whether class `Tutor` has a non-empty extension for the specified temporal period and let us assume that it succeeds. Class `Tutor` is a specialization of class `GStudent`, hence, the syntactic simplification component transforms the initial query by applying the completion rule R3 from table 1. The query which finally is transferred to the semantic transformation component is:

Q′: **return x**:(∃y)(∃t1,t2,t3,t4)
    (in(y,Tutor,t1)∧during(t1,1989)∧
    in(y,GStudent,t4)∧during(t4,1989)∧
    school(y,UofT,t2)∧during(t2,1989)∧
    address(y,x,t3)∧during(t3,1989)).

Semantic transformation starts with DDRB to be empty and the **Q**′ to be in clausal form:

DDRB: {}
Query:
  Q1:in(y,Tutor,t1),during(t1,1989).
  Q2:in(y,GStudent,t4),during(t4,1989).
  Q3:school(y,UofT,t2),during(t2,1989).
  Q4:address(y,x,t3),during(t3,1989).
K=DDRB ∪ Query.

Resolution steps:

B1.

taxonomic reasoner, Q2 : K_1=K ∪ DR1.
  DR1:
    ¬in(z,GStudent,t5)∨¬school(z,s,t6)
    ∨¬address(s,a,t7)∨address(z,a,t8)
    and t5,t6,t7,t8 **at** 1988..**Now**.

general resolution:
  F1[DR1,Q2]: ¬school(y,s,t6)∨
    ¬address(s,a,t7)∨address(y,a,t8)
    where t6,t7,t8 **at** 1988..**Now**.
    θ_1={y/z}.
    temporal reasoner: during(t4,1989) implies
    at(t5,1988..**Now**), where t4=t5.
  F2[DR1,Q3]: ...
  F3[DR1,Q4]: ...

B2.

general resolution:
  F4[F1,Q3]:
    ¬address(UofT,a,t7)∨address(y,a,t8)
    where t7,t8 **at** 1988..**Now**.
    θ_4={UofT/s}.
    temporal reasoner: during(t2,1989) implies
    at(t6,1988..**Now**), where t2=t6.

...

B3.
  Fk[F4,Q4]: address(UofT,x,t8)
    where **t8 at 1988..Now**.
    θ_k={x/a}.
    temporal reasoner: **during(t3,1989)** implies
    at(t7,1988..**Now**), where t3=t7.

...

A low cost query obtained during the above optimization process is:

Q": **return x**: address(UofT,x,t3)∧during(t3,1989).

Note that, at any elimination step four elements are employed i.e., {**literal**, **temp_constraint1**, ¬**literal**, **temp_constraint2**}, {**literal**, ¬**literal**} are eliminated by *subsumption elimination* [GN87], while {**temp_constraint1**, **temp_constraint2**} are eliminated only if they imply each other. Implicitly, two resolution steps are performed at once, the one is handled by the resolution procedure and the other by the temporal reasoner. The use of classical resolution would not serve our purposes because **during(t2,1989)** and **at(t5, 1988..Now)** can not be selected even if {**during(t2,1989)**, ¬**at(t5, 1988..Now)**} can be eliminated. We solve this problem at the unification level by requiring pairs of **(literal, temp_constraint)** to unify on the **literal**.

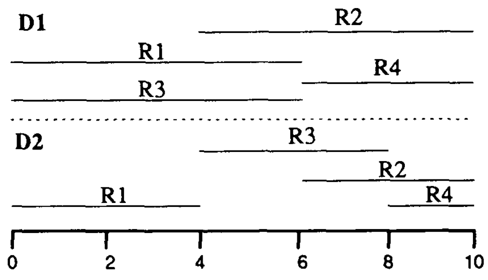
7 Preliminary Performance Results

This section presents some preliminary performance results of the presented method. This method is partially implemented but has not yet been integrated with one of the Telos implementations that we have [TK89], [Jar91]. Therefore, the experiments reported below are based on simulated execution of the derived access plan over a cost model. The cost model assumes a secondary storage knowledge base whereas all existing Telos implementations are main memory resident. The set-up of the experiments include the following components:

knowledge base: The knowledge base is taken form the ConceptBase implementation [Jar91] and consists of three classes (namely **Employee**, **Manager** and **Department** with 200, 20 and 10 instances respectively), four attribute relations (**boss**, **dept**, **salary**, **head** with 400, 400, 600 and 40 tuples respectively) and four rules/constraints The knowledge base contains historical knowledge for a ten years period, e.g. 1..Now, Now=10.

assumptions: We assume that every class and every attribute is implemented as a secondary storage relation. We consider only historical time. Lastly, we assume uniform distribution of data over the knowledge base history.

rule distributions: We test the behaviour of our methods for 2 different distributions of rules over time.



In distribution D1, the four rules cover, in total, a duration of 24 units over the 10 units history and they have

[6]For simplicity we are dealing only with the historical time.

20 units of mutual overlap. In distribution D2, rules have shorter lifetimes and cover 14 units of history with only 6 units of mutual overlap. This means that temporal selection over D1 returns larger rule bases than D2, and D1 has better potentials for query optimization.

cost model: Access of secondary relations is done using two indices, one temporal which locates the tuples that are valid at the given historical time and an inverted list on attribute values for the attribute relations and on instance name for the class relations, respectively. Access of these indices as well as the rule base index costs 2 time units. Access of selected data costs time proportional to the amount of data being accessed. All main memory operations take .001 time units, including resolution steps, rule rewritting, comparison, etc. The total cost of a query is the optimization cost plus the execution cost.

queries: We run four queries on five different historical intervals for any of the two rule distributions. We measured the execution cost of the query for the no-optimization case and the total cost for the optimization cases. The queries (taken again from the ConceptBase implementation) are:

1. *Find all employees who work for Hubert,* **ON** T.

2. *Find all employees who work in the marketing department,***ON** T.

3. *Find all employees who work in the marketing department and make more that 10k,* **ON** T.

4. *Find all the managers with salary from 50 to 60K,* **ON** T.

The following tables summarize the results of our experiments:

	Query #1			Query #2		
T	no-opt	D1	D2	no-opt	D1	D2
Now	28.46	30.43	30.43	30.11	32.11	32.12
1	28.44	18.01	18.01	30.11	13.01	13.01
2..5	97.53	33.02	33.02	103.23	30.04	30.04
4..5	50.34	23.01	52.35	54.33	14.02	56.34
6..7	50.34	52.35	52.35	54.33	56.34	56.34

	Query #3			Query #4		
T	no-opt	D1	D2	no-opt	D1	D2
Now	54.12	56.15	56.15	9.01	2.01	2.01
1	54.15	18.01	18.01	9.01	2.01	11.01
2..5	111.56	38.06	38.06	30.14	2.01	2.01
4..5	60.46	20.14	62.46	18.01	2.01	2.01
6..7	60.46	62.46	62.46	18.06	2.01	2.01

These results show improvements from 44% to 71% in the total query cost when a better execution plan is obtained, and up to 94% improvement if the semantic optimizer finds that the query will obtain empty answer and suspend its execution. If the optimization fails to find a better execution plan, either because no rules were selected or just because no better execution is possible, then there is a 3% to 7% overhead in the total cost and this tends to be smaller if the size of the database increases. Better performance is obtained for T=4..5, average 79%, whether in some other time intervals (i.e., T=6..7) the algorithm paid an overhead. Obviously, this depends on the rules' contents and their distribution over the history. In average, we have got a 45% improvement on all queries with the D1 rule distribution and a 32% average improvement with D2. Therefore, more dense rule distributions have better optimization potentials. Lastly, we test our algorithm under the assumption that data are uniformly distributed over the knowledge base history. This however was a very conservative assumption. If data are condensed at the recent history, then is more likely that the temporal simplification algorithm will recover "empty answer" for the very past historical states of the knowledge base.

8 Summary and Future Work

This paper describes a framework for query optimization for KBMSs. The contributions of the paper, are: (a) the definition of *temporal simplification*, (b) the reformulation of *syntactic simplification* and (c) advances in *semantic transformation*.

Although temporal databases have been studied extensively (e.g., [Sno87]) the issue of temporal optimization has not received as much attention. The few existing work dealing with this problem (e.g., [Ahn86], [SG89]), focuses on storage level issues. In this paper we define algorithms for (a) checking if all the temporal constraints that appear in a query are consistent, (b) checking if there are available data for the interesting time period, and (c) selecting an appropriate subset of the knowledge base which is involved in the answering of the query from a temporal viewpoint.

The rule based syntactic transformation we presented here is a step forward from the syntactic transformation defined in the database context [JK84]. We added new rules which deal with built-in predicates corresponding to the instantiation, specialization and aggregation concepts. We have designed an efficient algorithm that allows, on one hand, completion of the query with new information needed for the semantic transformation step and, on the other hand, simplification of the query, eliminating redundant information and detecting inconsistent (sub)queries.

Our method for semantic transformation combines the two different approaches used in databases, so far, for semantic query optimization (i.e., [Kin81], [Jar86]), although this paper discusses only the artificial intelligence based transformations. The innovation in our algorithm is the theory resolution based strategy for searching for semantically equivalent queries.

Moreover, our algorithm attempts to minimize the cost of the optimization procedure in two ways. First, it selects a rule base which is small in size and depends on the query. Second, it uses theory resolution which reduces substantially the number of inference steps during query transformation. A different approach to control the cost of semantic query optimization is taken in [SSD88].

The main focus in this work is the query modification phase of query optimization. The preliminary performance results are very encouraging. The next step is to extend our experiments for various distributions of the temporal data and rules and investigate the *access planning phase*. As the experiments suggested, many queries share "similar" parts and eventually receive "similar" execution plans. Cost reduction in such cases can be achieved through *multiple query optimization*. Some ideas developed in the multiple query optimization research area [Sel88] can be considered for the KBMS query processing too.

Acknowledgements
Special thanks are due to prof. J. Mylopoulos for providing us his guidance and the research environment to work. We also thank prof. R. Demolombe, B. Nixon, M. Koubarakis, V. Chaudhri, D. Plexousakis for their help.

References

[Ahn86] I. Ahn. Towards an Implementation of Database Management Systems with Temporal Support. *Data Engineering '86*, pp. 374–381, 1986.

[All83] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *CACM*, 26(11):832–843, 1983.

[BGL85] R.J. Brachman, V.P. Gilbert, and H.J. Levesque. An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON. In *Proc. of IJCAI-85*, pp. 532–539, 1985.

[BM86] M. Brodie and J. Mylopoulos, editors. *On Knowledge Base Management Systems: Integrating AI and Database Technologies.* Springer Verlag, 1986.

[CDRS86] M. Carey, D. DeWitt, J. Richardson, and E. Shekita. The Architecture of the EXODUS Extensible DBMS. In *Proc. of the Asilomar Workshop on Object-Oriented Database Systems*, 1986.

[CGM88] V.S. Chakravarthy, J. Grant, and J. Minker. Foundations of Semantic Query Optimization for Deductive Databases. In *Foundations of Deductive Databases and Logic Programming*, Morgan-Kaufmann, 1988.

[DBG+85] U. Dayal, A. Buchmann, D. Goldhirsch, S. Heiler, F. A. Manola, J.A. Orenstein, and A.S. Rosenthal. PROBE - A Research Project in Knowledge-Oriented Database Systems: Preliminary Analysis. CCA-85-03, CCA, 1985.

[DIB88] R. Demolombe, A. Illarramendi, and J.M. Blanco. Semantic Optimization in Data Bases Using Artificial Intelligence Techniques. In *Proc. of the IFIP Working Conf. on Data and Knowledge Bases*, 1988.

[DST80] J. Downey, R. Sethi, and R.E. Tarjan. Variations of the Common Subexpression Problem. *JACM*, 27(4), 1980.

[Fre89] J.C. Freytag. The Basic Principles of Query Optimization in Relational Database Management Systems. In *Proc. of IFIP-89*, pp. 801–807, 1989.

[GN87] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, 1987.

[Gut84] A. Guttman. R-Trees: A Dynamic Index Structure For Spatial Searching. In *Proc. of ACM SIGMOD-84*, pp. 47–57, 1984.

[HZ80] M. Hammer and S. Zdonik. Knowledge Based Query Processing. *VLDB-80*, pp. 137–147, 1980.

[IB90] A. Illarramendi and J.M. Blanco. Semantic Query Optimization: From DBMS To KBMS. In *Proc. of Int. Workshop on the Deductive Approach to Information Systems and Databases*, 1990.

[IS90] A. Illarramendi and L. Sbattella. Syntactic Query Processing: Dealing with Structure and Time. In *Proc. of Int. Workshop on the Deductive Approach to Information Systems and Databases*, 1990.

[Jar86] M. Jarke. External Semantic Query Simplification:A Graph Theoretic Approach and its implementation in PROLOG. In *Proc. from the 1st Int. Workshop on Expert Database Systems*, pp. 675–696, 1986.

[Jar91] M. Jarke. ConceptBase V3.0 User Manual. Technical Report MIP-9106, Univ. of Passau, 1991.

[JK84] M. Jarke, J. Koch. Query Optimization in Database Systems. *Computing Surveys*, 16(2):111–143, 1984.

[JK89] M. Jarke and M. Koubarakis. Query Optimization in KBMS: Overview, Research Issues, and Concepts for a Telos Implementation. KRR-TR-89-6, Dept.of CS, Univ. of Toronto, 1989.

[Kin81] J.J. King. *Query Optimization by Semantic Reasoning.* PhD thesis, Dept.of CS, Stanford, 1981.

[KRB85] W. Kim, S. Reiner, S. Batory, editors. *Query Processing in Database Systems.* Springer-Verlag, 1985.

[MBJK91] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: A Language for Representing Knowledge About Information Systems. *ACM TOIS*, 8(4), October 1991.

[MS88] S. Miller, L. Schubert. Using Specialists to Accelerate General Reasoning. *AAAI-88*, 161–165, 1988.

[PSKQ89] C. Peltason, A. Schmiedel, C. Kindermann, and J. Quantz. The BACK System Revisited. KIT-Report 75, Tech. Univ. Berlin, 1989.

[PSS+87] H.-B. Paul, H.-J. Schek, M.H. Scholl, G. Weikum, and U. Deppish. Architecture and Implementation of a Darmstadt Database Kernel System. In *Proc. of ACM SIGMOD-87*, pp. 196–207, 1987.

[RH80] D.J. Rosenkrantz and M.B. Hunt. Processing Conjunctive Predicates and Queries. In *Proc. of VLDB-80*, pp. 64–74, 1980.

[Sch86] P. Schwarz. Extensibility in the Starburst Database System. In *Proc. of the Asilomar Workshop on Object-Oriented Database Systems*, 1986.

[Sel88] T.K. Sellis. Multiple Query Optimization. *ACM TODS*, 3(1), 1988.

[SG89] A. Segev and H. Gunadhi. Event-Join Optimization in Temporal Relational Databases. In *Proc. of VLDB-89*, pp. 205–215, 1989.

[Sno87] R. Snodgrass. The Temporal Query Language TQuel. *ACM TODS*, 12(2):247–298, June 1987.

[Sri88] S.M. Sripada. A logical framework for temporal deductive databases. *VLDB-88*, pp. 171–182, 1988.

[SSD88] S. Shekhar, J. Srivastava, S. Dutta. A formal model of trade_off between optimization and execution costs in semantic query optimization. *VLDB-88*, 1988.

[ST89] J.W. Schmidt and C. Thanos, editors. *Foundations of Knowledge Base Management: Contributions from Logic, Databases, and Artificial Intelligence Applications.* Springer Verlag, 1989.

[Sti85] M.E. Stickel. Automated Deduction by Theory Resolution. In *Proc. of IJCAI-85*, pp. 455–458, 1985.

[TK89a] T. Topaloglou and M. Koubarakis. An Implementation of Telos. KRR-TR-89-8, Dept.of CS, Univ. of Toronto, 1989.

[VK86] M. Vilain and H. Kautz. Constraint Propagation Algorithms for Temporal Reasoning. In *Proc. of AAAI-86*, pp. 377–382, 1986.