Syntactic Query Processing: Dealing with Structure and Time

A. Illarramendi(1) - L. Sbattella(2)

(1) Facultad de Informatica. Basque Country Univ. San Sebastian (Spain)
(2) Dipartimento di Elettronica. Politecnico di Milano. Milano (Italy)

Abstract

The present paper proposes new syntactical simplification opportunities deriving from the presence in a Query Language of constructs explicitly referring to structural and temporal information.

The paper refers to the more general aspects of the Knowledge Representation Language Telos and treats the associated Query Language using an extension of the Predicate Calculus. The Extended Predicate Calculus supports references to the modeling aspects which make Telos particularly expressive and extendible: structured and temporal information, attribute categories, and user-defined meta attributes.

1 Introduction

Knowledge Representation Languages (KRLs) and Semantic Data Models (SDMs) provide some consolidated and widely shared data structuring capabilities and differently consider some desiderata well known in new application environments.

The present paper:

- Refers to the more general aspects of Telos [KMSB89] and treats the associated Query Language using an extension of the Predicate Calculus which supports:
  - Structuring mechanisms
  - The representation and reasoning about temporal knowledge
  - The grouping of attributes through categories and the instantiation of them
  - Extensibility of the KRL through the definition of user-defined meta-attributes in a multi-level structure.

- Proposes new syntactical simplification opportunities derived from the presence in the query language of constructs explicitly referring to the new facilities supported.

The aspects of Telos considered in this paper (see Section 2) are only the most interesting from the Query expression and simplification point of view. Different types of transformation rules can be used to simplify the query or to detect (during the transformation process) inconsistencies present in the query expression. The query expression during the syntactic simplification process can also be enriched with new information used during the following semantic transformation step.

The set of rules defined for syntactical simplification in traditional DBMS [JK84] and working with general
quantified expressions are considered to be applied as a first step of the query simplification. The proposed new rules take into consideration the temporal knowledge, the presence of elements declared as belonging to the multi-level conceptual architecture, the structuring mechanisms and the set-oriented structures.

Section 3.1 introduces a graphical representation of the syntactical aspects of the query which is used during the simplification process and, in Section 3.2, to present the simplification rules. The simplification algorithm is presented in Section 3.3. The new rules and the way in which they can be applied, considered with the Extended Predicate Calculus Query Language introduced, constitute the original part of this work.

The algorithms of simplification given by [RH80] [VK86] are taken into account in Section 3.4 and 3.5 to show syntactical simplification opportunities dealing with comparison operators and temporal predicates.

2 The Knowledge Representation Language Telos and the associated Query Language

2.1 The Knowledge Representation Language Telos

In this section the structuring mechanisms, the extensibility and the representation of time in Telos are going to be shortly presented.

2.1.1 Structuring Mechanisms and Extensibility

Telos provides an O-O representation framework which has been influenced by semantic networks and frame-based representations and supports as structuring mechanisms: instantiation, aggregation, specialization, and disjointness.

The INSTANTIATION mechanism of the KRL defines an infinite dimension along which the elements can be classified as tokens (i.e. elements belonging to the "Level 0" and having no instances), simple classes (i.e. elements belonging to the "Level 1" and having only tokens as instances), metaclasses of the first meta-class level (i.e. elements belonging to the "Level 2" and having only simple classes as instances) and so on. The instantiation mechanism allows the classification and grouping of elements having the same structure (represented through attributes and attribute categories as follows).

An element classified as (simple or meta)class or token is viewed as an AGGREGATE of its components, represented in terms of attributes. Attributes are considered multivalued. Attributes can be grouped through attribute categories and they can be instantiated, specialized, and have attribute of their own. This, together with the multi-level conceptual architecture, provides the basis for the KRL extendibility.

Orthogonally to the instantiation dimension, classes can be SPECIALIZED through ISA hierarchies. A class defined as the specialization of another one, inherits all the attributes of the super class.

Two (meta)classes are DISJOINT when their extensions have no instance in common.

The completion and simplification rules proposed in Section 3.2 will show the semantics of the structuring construct respect to time.

In Figure 2.1a some classes and tokens definitions are given with a representation of the defined elements in the multi-level conceptual architecture.

The names of classes and tokens begin with a capital letter, the names of attributes and attribute categories are not capitalized and keywords present in the declaration are capitalized. The keywords CLASS and TOKEN distinguish the definition of a class from the definition of a token. The IN clause precedes the list of all the classes of which the defined element is an instance; the ISA clause precedes the list of all the classes of which the defined element is a specialization. S-Class and Ml-Class are built-in classes respectively having as extensions the set of all simple-classes and of all meta-classes defined at Level 2. The list of all the defined built-in classes is given in Table 2.1.

Referring to Figure 2.1a note that:

- paper-descr is for the class Paper-Class an attribute and for the classes Paper and Conf-Paper (both instances of Paper-Class) an attribute category.
- Two different instantiations of the attribute category paper-descr have been defined respect to the classes Paper and Conf-Paper. The first produces the attributes author and title, the second produces the attributes referee and replay-address.
- Being the two classes Paper and Conf-Paper relied through an ISA relationship the two attributes author and title are inherited by the class Conf-Paper.
- The instantiation of the attribute category author in the Token Sdm produces two attributes: one with label first and the other with no label. The first one can be directly referred in assertions, the second one can only be retrieved through the attribute category (see in the following section the definition of the two terms x.n and x|h)

In Figure 2.1b the semantic network corresponding to the represented situation is shown. It can be useful to observe the relationships of instantiation between classes, tokens, attribute classes and attributes.
2.1.2 The representation of Temporal Knowledge

Telos adopts a modified case of Allen's time interval framework [All84] in order to represent historical knowledge about the domain. Allen's framework for reasoning about time, is based on the notion of interval. Thirteen primitive relationships (e.g., before, after, during, at etc.) are used to characterize any possible temporal relationship between two time intervals. Any temporal relationship between two intervals \( t_1 \) and \( t_2 \) can be written as a disjunction of at most 13 positive atomic formulas with predicate symbols corresponding to Allen's primitives, and arguments \( t_1 \) and \( t_2 \). The modified Allen's framework in Telos includes an infinite set of constants corresponding to conventional dates and times (e.g., 1986/Q/20 denoting October 20, 1986), semi-infinite intervals having conventional dates or times as one endpoint (e.g., 1986/Q/20. . *), and the infinite interval \( \text{Alltime} \) (see Table 2.1).

Telos uses all the thirteen temporal relations defined by Allen (see Figure 2.2), but assumes that the inserted temporal statements are of the form \( t \) \(<\text{op}>\) Constant with \( <\text{op}> \) to be any temporal relation and \( t \) a variable which ranges over the built-in class TimeInterval or the infinite number of time constants. In the query language, disjunctions between temporal predicates will not be allowed.

On the other hand, 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. For this reason Telos represents the system's beliefs about this history*. The system will record that it believes an information from the time the transaction committed and on. This information about the beliefs of the system can later be updated to signal the end of the system's belief in certain information.

2.2 The Query Language

The assertional language associated to the KRL (and used to define integrity constraints, deductive rules and the assertional part of queries) is the extended typed first order language described in Appendix A.

In this paper only queries are considered, their syntactical expression is as follows:

\[
\textless\text{query}\textgreater ::= \textless\text{target list}\textgreater \textless\text{assertional part}\textgreater
\]

All the constants appearing in a query expression must be declared as elements "known" to the KB. All the variables appearing in the assertional part of the query range over closed domains. These domains are either extensions of defined knowledge base classes or sets formed by the evaluation of set expressions. The notations for the variable restrictions are:

\[
x/C [t_1,t_2] \\
or \text{instanceOf}(x,C,t_1,t_2)
\]

Both expressions define the variable \( x \) as instance of the class \( C \) during \( t_1 \) and believed during \( t_2 \). The first expression is preferred in the formulation of the target list of a query and in quantified expressions.

\[
x/S [t_1,t_2] \\
or \text{memberOf}(x,S,t_1,t_2)
\]

Both expressions define the variable \( x \) as member of the set \( S \) during \( t_1 \) and believed during \( t_2 \). The first expression is preferred in quantified expressions.

The target list is composed by variables restricted by a class as previously specified.

3 The Syntactical Simplification Process

Syntactic query optimization is an approach to query optimization [KRB85] that uses special properties of the query languages to transform a query into another one which has the same answer and can be processed more efficiently. In our case, properties of the structuring mechanisms and temporal information supported by the query language are used in order to transform the query.

Two syntactically different query expressions are said to be equivalent if their evaluation gives the same result for the same state of the knowledge base.

The major goals of the syntactical query optimizer described here are:

- to simplify the query expression applying transformation rules able to detect predicates always true, always false, or inconsistent respect to the KB. (In this case, the user is warned by the syntactical optimizer about the reasons of the evaluation.)
- to express the query in a convenient way respect to the following step of the optimization process.

The most original parts of the simplification process are described in this section. The first part deals with the presence of structuring constructs in the query language (see Section 3.2 and 3.3), the second part deals with the simplification of predicates involving temporal relationships, temporal variables and/or temporal constants (see Section 3.4), and the third part considers the opportunities of simplification relied to predicates involving...
TABLE 2.1

<table>
<thead>
<tr>
<th>Temporal Relations</th>
<th>Inverses</th>
<th>Graphical Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>t: starts &lt;i&gt;in&lt;/i&gt;</td>
<td>t: ends &gt;i&gt;in&lt;i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: contains &lt;i&gt;in&lt;/i&gt;</td>
<td>t: starts &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: overlap &lt;i&gt;in&lt;/i&gt;</td>
<td>t: overlap &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: before &lt;i&gt;in&lt;/i&gt;</td>
<td>t: before &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: before &lt;i&gt;in&lt;/i&gt;</td>
<td>t: before &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: overlap &lt;i&gt;in&lt;/i&gt;</td>
<td>t: overlap &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: starts &lt;i&gt;in&lt;/i&gt;</td>
<td>t: starts &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: ends &gt;i&gt;in&lt;i&gt;</td>
<td>t: ends &gt;i&gt;in&lt;i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: overlap &lt;i&gt;in&lt;/i&gt;</td>
<td>t: overlap &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: before &lt;i&gt;in&lt;/i&gt;</td>
<td>t: before &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: overlap &lt;i&gt;in&lt;/i&gt;</td>
<td>t: overlap &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: starts &lt;i&gt;in&lt;/i&gt;</td>
<td>t: starts &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: ends &gt;i&gt;in&lt;i&gt;</td>
<td>t: ends &gt;i&gt;in&lt;i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: overlap &lt;i&gt;in&lt;/i&gt;</td>
<td>t: overlap &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: before &lt;i&gt;in&lt;/i&gt;</td>
<td>t: before &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: overlap &lt;i&gt;in&lt;/i&gt;</td>
<td>t: overlap &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
<tr>
<td>t: starts &lt;i&gt;in&lt;/i&gt;</td>
<td>t: starts &lt;i&gt;in&lt;/i&gt;</td>
<td></td>
</tr>
</tbody>
</table>
At the beginning of the syntactical simplification process the query is considered, for previous steps, as being:

- safe according to an extension of the classical definition of safe formula [Ull89] respect to the temporal information
- correctly typed (i.e. all the elements involved in \(<\text{comp-op}>, <\text{time-rel}>, \text{memberOf}, \text{and subsetOf} \) atomic formulas and all the elements related through an arithmetic operator or through a set operator must be of the correct type as specified in the Appendix)
- expressed in Prenex Disjunctive Normal Form. (The transformation rules shown by [JK84] dealing with the quantifiers movement and the normalization of expressions are previously applied.)
- Simplified respect to the idempotency rules involving logical expressions [JK84] [DT80] and respect to the rules involving set expressions and arithmetic expressions appearing in terms.

3.1 Representing the Syntactic Aspects of the Query as a Graph

During the syntactic simplification process, the syntactic aspects of the query are represented as a labelled graph:

- **nodes** represent the constants and variables terms,
- **edges** represent the atomic formulas. The edge between the nodes \(X\) and \(Y\) is referred as \(e(X,Y)\)
- **node labels** contain the name of the represented element and the level at which the element is required to be during a time interval. The node label of \(X\) is referred as \(\lambda(X)\)
- **edges labels** contain time information related to the atomic formulas. The edge label of \(e\) is referred as \(\lambda(e)\)

Figures 3.1 and 3.2 illustrate the relationships between the syntactic constructs of the queries and the corresponding elements in its graph representation.

A set expression which appears as an element of the query expression, is represented on the graph as a single node and its elements are represented as nodes related to the set node by an edge representing the relationship memberOf.

A query without disjunctions or negations in the assertional part is represented through one labelled graph. The graph contains both its target list and its assertional part.

If the assertional part of the query contains disjunctions, i.e.

\[ P_1 \lor P_2 \lor \ldots \lor P_n \]

where \(P_1, P_2, \ldots, P_n\) are conjunctions of predicates, then a separate graph is built for each \(P_i\). This graph contains the \(P_i\)'s predicative part its corresponding target list and the quantified part of the initial query related to \(P_i\).

All these graphs are considered during the syntactic simplification.

For each predicate preceded by negation a graph is produced considering its "affirmative" part. The negation is considered after the simplification process.

3.2 Syntactic Simplification using Structuring Constructs

In this section, we consider the structural properties of the query language and we define the transformation rules that are used during the syntactic simplification process. The rules are strictly relied to the semantics of the language constructs and to the constraints defined on them. The rules can be graphically represented as done for queries.

In the sequel, the rules are presented giving their assertional representation and the graph pattern which represents them. The bold part in the graph pattern represents the right hand side of the rule (i.e. the information that is produced when the rule is applied).

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

3.2.1 Completion Rules

The rules that are presented here, are going to be used to complete the query. Essentially, they represent properties of principles, as specialization, instantiation, set membership, etc. The new information that is added is used to detect inconsistencies or to simplify the query.

In the presentation of the rules we use the following notation: Symbols \(Cl, C2, C3\) represent different classes; \(Sl, S2, S3\) represent different sets; \(t_1, t_2, \ldots, t_{12}\) represent time intervals; \(x_1, x_2\) denote members of sets and \(X, Y\) denote instances of classes. Lastly, \(t_k = t_1 + t_2\) is defined as the common time interval between two time intervals \(t_1, t_2\) in case they overlap, and it is undefined if they are disjoint. \(^1\)

\(^1\)The definition of the interval intersection is:

\[
\left( s(t_1), e(t_1) \right) \times \left( s(t_2), e(t_2) \right) = \\
\left( \text{undefined} \right) \text{if } s(t_1) > s(t_2) \text{ and } e(t_2) > e(t_1) \\
\left( s(t_2), e(t_2) \right) \text{ if } s(t_2) < s(t_1) \text{ and } e(t_2) < e(t_1) \\
\left( s(t_1), e(t_1) \right) \text{ if } s(t_1) < s(t_2) \text{ and } e(t_1) < e(t_2) \\
\]

where \(T = T_1 + T_2\) is defined as the:

\[
T = t_1 + t_2 = \\
\left\{ \\
\left( s(t_2), e(t_1) \right) \text{ if } s(t_1) < s(t_2) \text{ and } e(t_1) < e(t_2) \\
\left( s(t_1), e(t_2) \right) \text{ if } s(t_2) < s(t_1) \text{ and } e(t_2) < e(t_1) \\
\left( s(t_1), e(t_1) \right) \text{ if } s(t_2) < s(t_1) \text{ and } e(t_1) < e(t_2) \\
\left( \text{undefined} \right) \text{ otherwise }
\right. \\
\]
R1. Transitivity rule for isA

\[ \text{isA}(C_1, C_2, t_1, t_2) \land \text{isA}(C_2, C_3, t_3, t_4) \Rightarrow \text{isA}(C_1, C_3, t_5, t_6) \]

where, \( t_5 = t_1 \cdot t_3 \) and \( t_6 = t_2 \cdot t_4 \).

If rule R1 is applied after rule R12 (as suggested in the following), then the levels of \( C_1, C_2 \) and \( C_3 \) are guaranteed to be equal.

Note that \( C_1, C_2, \) and \( C_3 \) are different classes (identified by different names and represented on the graph as different nodes). Therefore the rule avoids cases where:

\[ \text{isA}(C_1, C_2, t_1, t_2) \land \text{isA}(C_2, C_3, t_3, t_4) \]

R2. Transitivity rule for subsetOf

\[ \text{subsetOf}(S_1, S_2, t_1, t_2) \land \text{subsetOf}(S_2, S_3, t_3, t_4) \Rightarrow \text{subsetOf}(S_1, S_3, t_5, t_6) \]

where, \( t_5 = t_1 + t_3 \) and \( t_6 = t_2 + t_4 \).

Note that \( S_1, S_2, \) and \( S_3 \) are different sets (represented on the graph as different nodes). Therefore the rule avoids cases where:

\[ \text{subsetOf}(S_1, S_2, t_1, t_2) \land \text{subsetOf}(S_2, S_1, t_3, t_4) \]

R3. Combination of instanceof and isA

\[ \text{instanceOf}(X, C_1, t_1, t_2) \land \text{isA}(C_1, C_2, t_3, t_4) \Rightarrow \text{instanceOf}(X, C_2, t_5, t_6) \]

where \( t_5 = t_1 + t_3 \) and \( t_6 = t_2 + t_4 \).

If we apply the rule after rules R12 and R13, then the levels of \( C_1, C_2, \) and \( X \) are guaranteed to be the appropriate.

Rule R3 expresses the effect of isA relation into instanceof relation.

R4. Combination of isA through Attributes

\[ \text{isA}(C_1, C_2, t_1, t_2) \land P(C_1 | n, t_3, t_4) \land Q(C_2 | n, t_5, t_6) \Rightarrow \text{isA}(C_1 | n, C_2 | n, t_7, t_8) \]

where \( P(C_1 | n, t_3, t_4) \) and \( Q(C_2 | n, t_5, t_6) \) are wffs involving \( C_1 | n, C_2 | n, t_3, t_4, t_5, \) and \( t_6 \).

\( t_7 = t_1 + t_3 + t_5 \) and \( t_8 = t_2 + t_4 + t_6 \).

This rule is also required to be applied after rule R12, so that the instantiation levels of \( C_1 \) and \( C_2 \) will be the appropriate.

Rule R4 is due to a specialization constraint saying that specialization between classes implies specialization between attribute classes.

\( s(t), e(t) \) are the beginning and the ending of an interval \( t \), respectively.

R5. Combination of memberOf and subsetOf

\[ \text{memberOf}(x_1, S_1, t_1, t_2) \land \text{subsetOf}(S_1, S_2, t_3, t_4) \Rightarrow \text{memberOf}(x_1, S_2, t_5, t_6) \]

where \( t_5 = t_1 + t_3 \) and \( t_6 = t_2 + t_4 \).

R6. Combination of isA and disjoint

\[ \text{disjoint}(C_1, C_2, t_1, t_2) \land \text{isA}(C_3, C_1, t_3, t_4) \Rightarrow \text{disjoint}(C_3, C_2, t_5, t_6) \]

where \( t_5 = t_1 + t_3 \) and \( t_6 = t_2 + t_4 \).

The levels of \( C_1, C_2 \) and \( C_3 \) are guaranteed to be equal when rules R12 and R14 are applied before R5.

3.2.2 Simplification Rules

The objective of the simplification rules is:

- to obtain a simplified representation of the query, eliminating redundant information;
- to detect inconsistent queries.

R7. Reflexivity with respect to isA

Replace \( \text{isA}(C_1, C_1, t_1, t_2) \) with \( \text{True}, \forall t_1, \forall t_2 \).

R8. Reflexivity with respect to subsetOf

Replace \( \text{subsetOf}(S_1, S_1, t_1, t_2) \) with \( \text{True}, \forall t_1, \forall t_2 \).

R9. Double isA between classes

Replace \( \text{isA}(C_1, C_2, t_1, t_2) \land \text{isA}(C_2, C_1, t_3, t_4) \) with \( \text{False} \), if there exists a time period common to either \( t_1 \) or \( t_2 \).

Rule R9 is applied after R12, so that the instantiation levels of \( C_1 \) and \( C_2 \) are guaranteed to be the appropriate.

R10. Double subsetOf between sets

Replace \( \text{subsetOf}(S_1, S_2, t_1, t_2) \land \text{subsetOf}(S_2, S_1, t_3, t_4) \) with \( \text{False} \), if there exists a time period common to either \( t_1 \) or \( t_2 \).

Rule R10 is applied after R12, so that the instantiation levels of \( C_1 \) and \( C_2 \) are guaranteed to be the appropriate.

R11. Double disjoint

Replace \( \text{disjoint}(C_1, C_2, t_1, t_2) \land \text{disjoint}(C_2, C_1, t_3, t_4) \) with \( \text{disjoint}(C_1, C_2, t_5, t_6) \) or \( \text{disjoint}(C_2, C_1, t_5, t_6) \),

where \( t_5 = t_1 \lor t_3 \) and \( t_6 = t_2 \lor t_4 \).
R12. IsA with respect to levels

Only classes residing on the same level of the instantiation dimension can be related through specialization and they do so when they concurrently exist. Tokens cannot be related through specialization.

In any atomic formula of the form isA(C1, C2, t1, t2), the instantiation level of C1 must be the same with the level of C2. The instantiation levels are shown on the graph on the labels of the nodes, i.e.

\[ \nu(C1) = C1(t_1, t_3, t_4) \quad \text{and} \quad \nu(C2) = C2(t_2, t_5, t_6) \]

1. If the values of \( l_1 \) and \( l_2 \) are given, then
   - If \( l_1 = l_2 = 0 \) then replace the isA expression with False
   - \( \forall t, t^* \in \{ t_3, t_4, t_5, t_6 \} \)
   - (Tokens are not related with isA relationship)
   - If \( l_1 = l_2 \neq 0 \) during a time interval \( t_7 = t_3+t_5 \)
   - And this is believed by the knowledge base during the time interval \( t_8 = t_4+t_6 \) then
   - If one of the \( t_1+t_3+t_5 \) and \( t_2+t_4+t_5 \)
   - Is not defined then replace the isA expression with False

2. If at one of \( l_1 \) and \( l_2 \) is not known, then
   - Assign the value of the one to the other.

3. If both \( l_1 \) and \( l_2 \) are not known then find \( l_1 \) or \( l_2 \) from applying R12 and R13 to other query predicates and then apply 2.

R13. instanceof with respect to levels

Only elements residing on different and consecutive levels of the instantiation dimension can be related through instantiation and they do so when they concurrently exist.

In any atomic formula of the form instanceof(X, Cl, t1, t2), the instantiation level of X must be one less than the instantiation level of Cl.

1. If the values of \( l_1 \) and \( l_2 \) are given, then
   - If \( l_1 = l_2 - 1 \) then replace the instanceof expression with False
   - \( \forall t, t^* \in \{ t_1, t_3, t_4, t_5, t_6 \} \)
   - (Instantiation relates only objects of successive instantiation levels)
   - If \( l_1 = l_2 - 1 \) during a time interval \( t_7 = t_3+t_5 \)
   - And this is believed by the knowledge base during the time interval \( t_8 = t_4+t_6 \) then
   - If one of the \( t_1+t_3+t_5 \) and \( t_2+t_4+t_5 \)
   - Is not defined then replace the instanceof expression with False

2. If \( l_1 \) or \( l_2 \) is not known, then
   - Assign an appropriate value to the other such that \( l_1 = l_2 - 1 \).

3. If both \( l_1 \) and \( l_2 \) are not known then find \( l_1 \) or \( l_2 \) applying R12 and R13 to other query predicates and then apply 2.

R14. disjoint with respect to levels

Only classes residing on the same level of the instantiation dimension can be related through disjointness and they do so when they concurrently exist. Tokens cannot be related through disjointness.

In any atomic formula of the form disjoint(Cl, C2, t1, t2), the instantiation levels of Cl and C2 must be equal.

1. If the values of \( l_1 \) and \( l_2 \) are given, then
   - If \( l_1 = l_2 = 0 \) then replace the disjoint expression with False
   - \( \forall t, t^* \in \{ t_3, t_4, t_5, t_6 \} \)
   - If \( l_1 = l_2 \neq 0 \) during a time interval \( t_7 = t_3+t_5 \)
   - And this is believed by the knowledge base during the time interval \( t_8 = t_4+t_6 \) then
   - If one of the \( t_1+t_3+t_5 \) and \( t_2+t_4+t_5 \)
   - Is not defined then replace the disjoint expression with False

2. If \( l_1 \) or \( l_2 \) is not known, then
   - Assign the value of the one to the other.

3. If both \( l_1 \) and \( l_2 \) are not known then find \( l_1 \) or \( l_2 \) from applying R12 and R13 to other query predicates and then apply 2.

R15. Combination of instanceof and disjoint

Replace

\[ \text{instanceOf}(X, Cl, t_1, t_2) \land \text{disjoint}(Cl, C2, t_3, t_4) \]

with False when there exists a time period common to \( t_1 \) and \( t_3 \) or \( t_5 \) or common to \( t_2 \) and \( t_4 \) and \( t_6 \).

R16. Combination of isA and disjoint

Replace

\[ \text{isA}(Cl, C2, t_1, t_2) \land \text{disjoint}(Cl, C2, t_3, t_4) \]

with False when there exists a time period common to both \( t_1 \) and \( t_3 \) or common to \( t_2 \) and \( t_4 \).

If the rules R12 and R14 are applied first (as suggested in the following) the levels of Cl and C2 are guaranteed to be equal.
3.3 Simplification Algorithm

The algorithm for syntactic query simplification is described in the following in an informal way.

Algorithm: syntactic query simplification algorithm

**Input:** the user submitted query, \( Q \)

**Output:** aborts \( Q \) if a contradiction is found, otherwise returns an equivalent query \( Q' \) (or a set of equivalent queries) that gives \( Q' \) the same answer as \( Q \).

begin
form the graph that corresponds to the query,
for each connected component of the graph do
1. apply successively the simplification rules: \( R_7, R_8, R_{12}, R_{13}, \) and \( R_{14} \)
2. complete the graph, applying rules: \( R_1, R_2, R_3, R_4, R_5, \) and \( R_6 \)
3. apply the simplification rules: \( R_9, R_{10}, R_{11}, R_{15}, \) and \( R_{16} \)
end for
reform the query using the new graph
end

The components of the graph correspond to the conjuncts of the query. These are analyzed separately. The result of the algorithm is obtained by combining the simplified subgraphs, for this purpose we use properties of the logical operators.

At the beginning of the algorithm a dictionary containing information related to instantiation, specialization, disjointness and level relations is available. Essentially this information is coming from the knowledge base. This type of structural information can be very important for the query processing: if maintained in a special way, the access cost can be minimal. The algorithm utilizes this information in order to label the edges of the constant nodes, verify their consistency, find inconsistencies in the query and lastly add more information in the query.

3.3.1 Halting Problem

The simplification algorithm always terminates in a finite number of steps. The following reasons motivate this assertion.

Each query is transformed into a graph with one or more (finite in the number) components. For each component steps 1,2 and 3 are applied successively. Each of the components represents a conjunctive subformula \( Q \). At step 1, the application of rules \( R_{12}, R_{13} \) and \( R_{14} \) on the query \( Q \) results in \( \text{False} \) or \( Q' \), where \( Q' \) is either the same as \( Q \) but with the instantiation levels checked to be correct or different with appropriate information added to \( Q \). Rules \( R_7 \) and \( R_8 \) simply cut conjuncts which are always \( \text{True} \).

Step 2, works iteratively until no new predicates (edges) are added to the subquery (graph component). The number of the 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 rules application is to falsify certain predicates of the subquery and perform the appropriate absorptions and of course it always takes a finite number of steps.

In conclusion, the syntactic simplification algorithm always terminates, and the result is the empty answer to the query if a contradiction is detected, or a new formulation of the query which is utilized by the next phases of the query optimizer.

3.4 Syntactical Simplification using Temporal Information

During this step of the simplification process only the parts of the graphs involving the edges \( \text{temporal} \) and their heads and tails are going to be considered in order to check inconsistencies in the \( \text{assertional} \) part of the query involving temporal predicate, or to simplify the temporal expression.

Vilain and Kautz in [VK86] propose a temporal constraint propagation algorithm dealing with temporal expressions formulated in terms of the point algebra. The algorithm is a modification of the Allen's algorithm [Al1984] and can be applied to a subset of the expressions allowed by the Allen's time interval algebra.

The temporal predicates allowed in the proposed knowledge representation language, belong to the subset defined by Vilain and Kautz (the vectors of temporal relations are always composed by only one element so that disjunction is not allowed between temporal predicates and no ambiguities can be introduced). Therefore, the temporal variables and the temporal constants present in temporal predicates can always be transformed in terms of point algebra. A modified version of the algorithm of Vilain and Kautz (which is sound and complete) can be applied to compute in polynomial time the closure of the assertions and to determine the satisfiability of the temporal assertions.

3.5 Syntactic Simplification using Comparison Operators

During the simplification process the parts of the graphs involving the \( \text{comparison} \) 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]. The single subgraphs in fact represent a conjunctive predicate involving the comparison operators: \( =, >, <, \leq, \geq \).
4 Conclusions and Future Work

New syntactical query simplification opportunities for knowledge representation languages that offer structuring capabilities, the representation of temporal information and an extensibility mechanism in a multi-level conceptual structure have been defined in this paper.

A new set of transformation rules and an algorithm that works with them have been defined. It has also been shown that the simplification process is decidable.

The completion and simplification rules proposed, are strictly relied to the semantics of the structuring constructs of the language respect to time. The syntactical simplification process is based on this semantics.

The set of rules is being implemented within the general ones traditionally defined in the context of databases by [JK84]. The implementation is based on QUINTUS Prolog on a SUN 360 workstation. The first results indicate that the time needed to realize this kind of optimization is really short. To give a complete evaluation of the proposed approach, new results are going to be obtained respect to more complex queries and larger knowledge bases.

---

References


---

Figure 24 Node labels of the graph representation of the query

Figure 25 Edges and edge labels of the graph representation of the query
APPENDIX

Terms:

- Variables and constants are terms.
- Sets are also terms. Sets can either be explicitly enumerated or are derived by the evaluation of set expressions listed below.
  - Explicitly enumerated sets:
    \[ \{C_1, C_2, \ldots, C_n\} \]
    where \( C_1, C_2, \ldots, C_n \) are constants.
  - Set expressions:
    1. \( x.n \). It is defined only for the case where \( x \) is an instance of a class and an attribute with label \( n \) has been defined for the class of \( x \) (\( n \) is for \( x \) an attribute category). It returns the values of all the attributes of \( x \), induced by the attribute category \( n \).
    2. \( x\in n \). It returns the set of values of the attribute \( n \). (Values composing the set are all classes or all tokens)

- If \( S_1 \) and \( S_2 \) are set terms then \( S_1 \, set\_operator \, S_2 \) is a set term.
- If \( n_1 \) and \( n_2 \) are terms of numeric type then \( n_1 \, arith\_operator \, n_2 \) is a term of numeric type.

Atomic Formulas:

- \( \text{instanceOf}(x, y, t_1, t_2) \)
  \( x \) is an instance of \( y \) for the time period \( t_1 \) and this is believed by the system for the time period \( t_2 \). \( x \) can be a token or a class, and \( y \) is a class. \( x \) and \( y \) must belong to different levels.
- \( \text{isA}(x, y, t_1, t_2) \)
  \( x \) is a specialization of \( y \) for the time period \( t_1 \) and this is believed by the system for the time period \( t_2 \). \( x \) and \( y \) must be classes that belongs to the same level.
- \( \text{disjoint}(x, y, t_1, t_2) \)
  \( x \) and \( y \) are disjoint classes for the time period \( t_1 \) and this is believed by the system for the time period \( t_2 \). \( x \) and \( y \) must be classes that belong to the same level.
- \( \text{comp\_op}(x, y) \)
  \( x \) and \( y \) are terms of numeric type.

Well Formed Formulas (wff):

- Every atomic formula is a wff.
- If \( a \) and \( b \) are wffs then \( a \lor b, a \land b, a \Rightarrow b, \) and \( a \iff b \) are also wffs.
- If \( a \) is a wff and \( b \) is a temporal atomic formula then \( a \land b \) is also a wff.
- If \( a \) is a wff then \( \neg a \) is a wff.
- If \( a \) is a wff then \( (a) \) is a wff.
- If \( P \) is a wff, \( C \) is a class, \( S \) is a set, \( x \) is a variable and \( t_1 \) and \( t_2 \) are time intervals, then
  1. \( (\forall x/C \{t_1, t_2\})P \)
  2. \( (\forall x/S \{t_1, t_2\})P \)
  3. \( (\exists x/C \{t_1, t_2\})P \)
  4. \( (\exists x/S \{t_1, t_2\})P \)

Temporal Atomic Formulas:

- \( \text{time\_rel}(x, y) \) **
  \( x \) and \( y \) must be time constant, and \( x \) or \( y \) can be a variable restricted by a built-in class including all the time intervals.

Well Formed Formulas (wff):

- Every atomic formula is a wff.
- If \( a \) and \( b \) are wffs then \( a \lor b, a \land b, a \Rightarrow b, \) and \( a \iff b \) are also wffs.
- If \( a \) is a wff and \( b \) is a temporal atomic formula then \( a \land b \) is also a wff.
- If \( a \) is a wff then \( \neg a \) is a wff.
- If \( a \) is a wff then \( (a) \) is a wff.
- If \( P \) is a wff, \( C \) is a class, \( S \) is a set, \( x \) is a variable and \( t_1 \) and \( t_2 \) are time intervals, then
  1. \( (\forall x/C \{t_1, t_2\})P \)
  2. \( (\forall x/S \{t_1, t_2\})P \)
  3. \( (\exists x/C \{t_1, t_2\})P \)
  4. \( (\exists x/S \{t_1, t_2\})P \)

**time\_rel** is one of temporal relation defined in Figure 2.2.

**time\_constant** is one of Now, Alltime or a conventional date (i.e., 01/12/1988).