Querying Temporal Databases Using Controlled Natural Language

Rani Nelken  Nissim Francez
Computer Science Department
The Technion
Haifa 32000, Israel

Abstract
Recent years have shown a surge in interest in temporal database systems, which allow users to store time-dependent information. We present a novel controlled natural language interface to temporal databases, based on translating natural language questions into SQL/Temporal, a temporal database query language. The syntactic analysis is done using the Type-Logical Grammar framework, highlighting its utility not only as a theoretical framework but also as a practical tool. The semantic analysis is done using a novel theory of the semantics of temporal questions, focusing on the role of temporal preposition phrases rather than the more traditional focus on tense and aspect. Our translation method is considerably simpler than previous attempts in this direction. We present a prototype software implementation of the method.

1 Introduction
Traditionally, database management systems were designed to store snapshot information, valid at a particular moment of time (state). However, many applications require handling dynamic time-dependent information, pertaining not only to the present, but also to the past and future. Adding temporal support to databases has proved to be a surprisingly thorny issue (TCG+93). A recent drive to consolidate research efforts has led to the design of a consensus temporal data model and associated temporal database query language, SQL/Temporal, an extension of the popular Structured Query Language (SQL) (Sno00). SQL/Temporal represents a significant improvement over standard SQL in allowing programmers to express temporal queries (Sno00). Since temporal database (TDB) implementations are still in their infancy (Boe95), there is little practical experience with SQL/Temporal; let alone experience of non-expert users. However, it is our belief that such users are bound to find the expression of complex temporal queries in SQL/Temporal to be extremely difficult.

In an attempt to counter this problem, we present a translation method from controlled natural language (NL) to SQL/Temporal. Following the standard pipeline architecture of such methods, translation is done via an intermediate meaning representation language illustrated in Figure 1. NL questions are first parsed using a grammar in the Type-Logical Grammar (TLG) framework (Car98; Mor98). Simultaneously with parsing, the NL question is translated into a formula of a formal language called \(L_{Allen}\) (Tom96), based on the interval operators of (All83). The translation is based on an independently motivated novel semantics of sentences modified by \textit{temporal Preposition Phrases (PPs)} (PF97; NF99; PF00). The constructed \(L_{Allen}\) formula is then translated into an SQL/Temporal query, which is subsequently

\[\text{\footnotesize \cite{Sno00}: Through continued design, SQL/Temporal has evolved from predecessor versions named TSQL2 (Sno95) and ATSQL2 (BCST96). It is expected to be incorporated within the new version of SQL named SQL3.}\]
submitted to a prototype TDB implementation for evaluation. Finally, the TDB’s answer is presented to the user.

\[
\text{NL} \rightarrow \text{LAllen} \rightarrow \text{SQL/Temporal} \rightarrow \text{TDB}
\]

Figure 1: The translation pipeline

We have implemented this method as a prototype software tool, called QWERTY, (almost) an acronym for “Querying with English of Relational Temporal Databases”. Parsing and translation to LAllen is done using the TLG Theorem Prover (Car99). The translation from LAllen to SQL/Temporal is done using an adaptation of a temporal logic (TL) to ATSQL2 translator of (BCST96). The resulting query is submitted to a prototype TDB implementation, called TimeDB (Ste97b). The different modules are coupled into an integrated system implemented in Sicstus Prolog on a UNIX platform with a WWW-based graphical front-end. We discuss some of the directions required in order to turn the system from a research prototype to a working tool.

2 Related work

There is voluminous literature on the design of NL interfaces to general (non-temporal) databases (see (PG88; CJ90; ART95) for surveys) and by now their main advantages and disadvantages are well understood. Much less work has been devoted to the design of NL interfaces to TDBs (Cli90; Hin88) or other computer systems involving a temporal dimension (CP93). Of particular relevance is (And96), who presents a linguistically motivated translation method from NL queries into TSQL2 using an HPSG (PS94) grammar and a TL as an intermediate representation language. Our approach shares many characteristics with (And96), but there also important differences, which we point out throughout the paper.

We begin our presentation of the translation method with a brief overview of the TDB, as its structure determines many of the design choices taken in devising the translation method.

3 The TDB

A TDB is a two-sorted first-order structure. The domain consists of a Data Domain, \( D \), and a Temporal Domain of intervals, \( T_I \), defined as follows. Let \( T_P \) be a set (of time points) with a discrete linear order without endpoints, \( \leq \). \( T_I \) is defined as the set of pairs: \( T_I = \{(a, b) \mid a \leq b \wedge a, b \in T \cup \{-\infty, \infty\}\} \). A relational database schema is a set of single-sorted predicate symbols \( (R_1, \ldots, R_k) \). Given a relational database schema \( \rho \), a TDB schema \( \rho' \) is the set of two-sorted predicate symbols \( (R'_1, \ldots, R'_k) \), where the sort of \( R'_i \) is \(\text{arity}(R_i) \times T_I\). A TDB instance of schema \( \rho' \) is a set of relations \( R'_i \subseteq \text{arity}(R_i) \times T_I \), where each \( R'_i \) is finite.

For instance, assume a database schema \( \rho \) consisting of a single binary predicate symbol \( \text{work} \), storing for each employee the department in which she is employed. The TDB schema \( \rho' \) consists of the relation \( \text{work}' \), called a valid-time state table, which adds a temporal argument to the original relation, called the valid-time of the table. The temporal argument can be used to store the history (and perhaps even future plans) of departments in which employees are employed. Following a suggestion of (And96), we also include in \( \rho' \) relations mapping names of calendrical items to temporal intervals. For instance, we store a relation \( \text{year}' \) mapping the year 2000 to the interval \([1.1.2000-31.12.2000]\) (which in turn is mapped to an element of \( T_I \)).

We now turn to describe the translation process.

4 The translation process

The translation process accepts input NL questions in a controlled subset of NL. Restricting input language in this way enables effective processing of a sufficiently rich fragment while avoiding many of the well-know problems of unrestricted NL. We use a formal grammar in the TLG framework. Our grammar is specially designed for use with a particular TDB schema. Future work will allow easier configuration of the grammar with respect to the schema.

Our grammar is based on work on an independently motivated theory of the semantics of temporality. Most of the research in this field (see (Ste97a) for a survey) has focused on the issues of tense and aspect. We handle tense, but purposefully not aspect, which plays an important role in (And96). Aspect, which is used to reflect speakers’ temporal viewpoint with respect to reported situations is an impor-
tant facet of NL temporality. However, its relevance to TDBs is questionable, as it is unlikely that a realistic TDB would actually encode such subjective viewpoints. Moreover, handling aspect requires postulating a more complex data model. For instance, (And96) augments the TDB model with event-like “occurrence identifiers”, and adds an additional argument to  

assumptions about temporal relations indicating whether a given event has culminated or not. While such devices may perhaps be linguistically justified, it is unlikely that the TDB community would adopt such augmentations of the model.

Instead, following (PF97; PF00; NF99) our focus is on sentences modified by temporal PPs. These PPs are analyzed as variants of standard quantified quantifiers (BC81), in which quantification is over time. Using this framework, we handle questions that refer explicitly to the
temporal dimension (e.g. When/during which year ... ) as well as questions in which temporality is implied by the TDB context (e.g. Did Mary work in marketing? Which employees worked in marketing?). We handle both clausal and phrasal temporal PPs (e.g. after John worked in R&D, during every year). An important strength of this semantic theory is that it allows for arbitrary iteration of PPs (e.g. one month during every year). In addition, our grammar also handles coordination, negation and quantification over individuals (e.g. some employee).

Input questions are parsed using a lexicalized type-logical grammar. Lexical items are associated with a syntactic category and a higher-order lambda-term representing its semantics. Taking advantage of TLG’s elegantly tight coupling of syntax and semantics, parsing and construction of a semantic representation in the form of an I.xx formula proceed simultaneously, in a bottom-up fashion. We have found using TLG to be advantageous over a feature-structure based formalism (such as HPSG as in (And96)), since formula construction is an integral part of the parsing and does not require complex ad-hoc manipulations of feature structures.

Using a particular grammar helps reduce some of the ambiguity inherent in unrestricted NL. For instance, whereas in general a preposition such as at is ambiguous between a temporal and a locative interpretation, the choice of

the complement NP relative to a given schema induced grammar deterministically fixes the interpretation. As another example, whereas iterating several temporal PPs (e.g. during some month every year) opens up exponential scoping possibilities, some choices are eliminated by world knowledge, which is encoded in the grammar (e.g. every year must have higher scope than some month since months are included in years and not vice-versa). In cases of remaining ambiguity, the user is presented with all the distinct possibilities. Future work will allow the user to make informed choices between different possible readings, e.g. by presenting him with NL paraphrases of the alternatives.

We translate NL questions into Lxx. The main reason for not translating directly to SQL/Temporal is that the latter is not closed for sub-formulae, i.e. a sub-formula of a well-formed query is not necessarily well-formed. Since Lxx is closed for sub-formulae, compositionally constructing formulae while parsing in a bottom-up fashion becomes much easier.

Lxx is defined as follows (Tom96). Let \( R \) be the database schema \((R_1, \ldots, R_k)\). Define:

\[
L := R_i(x, I)|L \land L|\neg L|\exists x.L|\exists I.L|\{x, I\}|\sigma.J
\]

where \( x, y \) are variables over \( D \), \( x \) is a vector of such variables, \( I, J \) are variables or constants over \( T_1 \), and \( \sigma \) is one of the operators: precedes, meets, overlaps, equals, contains. Lxx is defined as the set of formulae \( \varphi \in L \) that contain at most one free variable over \( T_1 \). The answer to a formula \( \varphi \) relative to a TDB \( D \) is \( \{x, I|D |\models \varphi(x, I)\} \).

To illustrate, consider the NL question:

**During which years did Mary work in marketing?**

The Lxx representation for it is constructed in a bottom-up manner. The meaning representation of the main clause Mary worked in marketing is constructed as:

\[
\lambda I, \exists J (work(mary, marketing, J) \land J \subseteq past \land J \subseteq I)
\]

In this formula, \( J \) denotes a Reichenbachian-like reference time, \( J \) denotes a time interval during which Mary worked in marketing, which is located in the past (the contribution of the tense) and is included within \( I \).

The meaning of the full question is constructed by applying the meaning of the interrogative temporal PP during which year to the
meaning of the clause. Without going into details, the result is:
\[ year(I) \land \exists J \ldots \]

The effect of applying the PP is that the variable \( I \) is now both free and restricted to be the time of a year. The answer to the formula is the set of intervals \( I \) that are years, and during which there is an interval \( J \) contained in the past, during which Mary worked in marketing. We allow iterated PPs to apply in a similar manner.

Not every \( L_{\text{Allen}} \) formula corresponds to an evaluable SQL/Temporal query. In particular, formulae might have an infinite answer. Formulae that are safe from this and related problems are termed domain-independent (GT91; AHV95). Domain independence is an undecidable semantic property. However, we impose certain syntactic restrictions on generated formulae that ensure it. These restrictions also simplify the translation task from \( L_{\text{Allen}} \) to SQL/Temporal. This translation is based on a modification of the translator from first-order TL over time-points to ATSQL2 of (BCST96).

The syntactically restricted version of \( L_{\text{Allen}} \) we use has the unique advantage of being very close both to the language used in (NF99; PF00) on the one hand and to SQL/Temporal on the other. The semantics of NL temporal expressions is often expressed using explicit reference to intervals. Likewise, SQL/Temporal has Allen-style operators over intervals. (And96) uses a customized TL as an intermediate language, in which temporal relations are encoded using temporal operators rather than explicit reference to intervals. By contrast, we have found using a syntactically restricted version of \( L_{\text{Allen}} \) to be advantageous, in that it actually simplifies the translation.

Continuing our previous example, the resulting \( L_{\text{Allen}} \) formula is subsequently translated into the following SQL/Temporal query:

```
SELECT DISTINCT a0.c1 AS c1  
FROM work1 AS a1, year1 AS a0  
WHERE VALIDTIME(a0) contains VALIDTIME(a1)  
  AND a1.c1 = 'mary'  
  AND a1.c2 = 'marketing'  
  AND PERIOD(TIMESTAMP'now') contains VALIDTIME(a1)
```

The query asks for the first argument of the relation instance \( \text{year1} \) such that the relation instance \( \text{work1} \) includes a tuple consisting of \( 'Mary', 'marketing' \) and a valid time, which is temporally included in the valid time of the year, as well as in the interval starting at the 'beginning' of time and ending 'now' - viz. the past. The TDB responds by returning a table containing exactly the requested year names.

### 5 Conclusion

The addition of the temporal dimension to database systems increases their power but also their complexity. To increase the usability of TDBs, we present a prototype controlled NL interface to a TDB. Our semantic focus is on the use of temporal generalized quantifiers, based on (PF00), rather than tense and aspect. As argued by (CJ90), handling quantification is one of the areas in which NL interfaces have a potential advantage over both formal languages and graphical user interfaces.

In comparison with previous work, we are able to considerably simplify the translation method. First, using TLG, rather than a feature-structure formalism provides a much simpler method for constructing semantic representations. Second, using \( L_{\text{Allen}} \) as an intermediate meaning representation language yields a much more straightforward translation than using a restricted TL.

One must bear in mind, that our implementation is at the prototype stage. Turning it into a practical tool would require considerable work, as is true of most comparable systems. Future work includes increased NL coverage, adding a disambiguation module, handling nominal and temporal anaphora, allowing multiple-sentence queries, and generation of NL answers from the results presented by the TDB.

### References


I. Androutsopoulos. *A principled Framework for Constructing Natural Language Interfaces*