Querying ATSQL Databases with Temporal Logic

Jan Chomicki
University at Buffalo
and
David Toman
University of Waterloo
and
Michael H. Böhlen
Aalborg University

We establish a correspondence between temporal logic and a subset of ATSQL, a temporal extension of SQL–92. In addition, we provide an effective translation from temporal logic to ATSQL that enables a user to write high-level queries which are then evaluated against a space-efficient representation of the database. A reverse translation, also provided in this paper, characterizes the expressive power of a syntactically defined subset of ATSQL queries.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Temporal Logic*; H.2.3 [**Database Management**]: Languages—*Query Languages*; H.2.4 [**Database Management**]: Systems—*Query Processing*

Additional Key Words and Phrases: Temporal databases, First-order temporal logic, ATSQL, Query translation

1. INTRODUCTION

This paper brings together two research directions in temporal databases. The first direction is concerned with temporal extensions to practical query languages such

Name: Jan Chomicki
Affiliation: Department of Computer Science and Engineering, University at Buffalo
Address: 226 Bell Hall, Box 602000, Buffalo, NY 14260, U.S.A., chomicki@cse.buffalo.edu
Name: David Toman
Affiliation: Department of Computer Science, University of Waterloo
Address: 200 University Ave. W, Waterloo, Ontario N2L 3G1, Canada, david@uwaterloo.ca
Name: Michael H. Böhlen
Affiliation: Department of Computer Science, Aalborg University
Address: Fredrik Bajers Vej 7E, DK-9220 Aalborg Øst, Denmark, boehlen@cs.auc.dk

This is a preliminary release of an article accepted by ACM Transactions on Database Systems. The definitive version is currently in production at ACM and, when released, will supersede this version.

as SQL [Gadia 1993; Navathe and Ahmed 1993; Sarda 1993]. The issues addressed include space-efficient storage, effective implementation techniques and handling of large amounts of data. This direction includes ATSQL [Böhlen and Jensen 1996; Böhlen et al. 2001], the integration of ideas from TSQL2 [Snodgrass 1995] and ChronoLog [Böhlen 1994]. The second direction of research focuses on high-level query languages for temporal databases based on temporal logic [Tuzhilin and Clifford 1990; Gabbay and McBrien 1991; Clifford et al. 1994]. The advantages of using a logic-based query language come from their well-understood mathematical properties [Gabbay et al. 1994]. The declarative character of these languages also allows the use of advanced optimization techniques. In addition, temporal logic has been proposed as the language of choice for formulating temporal integrity constraints and triggers [Chomicki 1995; Chomicki and Toman 1995; Gertz and Lipeck 1996; Lipeck and Saake 1987; Sistla and Wolfson 1995], as it admits space-efficient methods for enforcing these constraints.

While temporal logic may seem to be a natural choice for a temporal query language, its semantics is defined with respect to abstract temporal databases: time-point-indexed sequences of database states [Gabbay et al. 1994; Chomicki 1994]. This view has disqualified temporal logic as a practical temporal query language. For efficiency reasons we cannot construct and store all the individual states explicitly. This task becomes simply impossible if the sequence of states is not finite. On the other hand, most of the practical temporal data models adopt a different approach. With every fact (usually represented as a tuple), a temporal data model associates some *concise encoding* of the set of time points at which the fact holds. The encoding is commonly realized by *periods*[1] [Navathe and Ahmed 1993; Sarda 1990; Snodgrass 1987; Tansel 1986] or *temporal elements* (finite unions of periods [Clifford and Croker 1987; Gadia 1988; Snodgrass 1995]) that represent large, perhaps infinite sets of individual time points. Even in the finite case the encoding may be exponentially more succinct than the corresponding abstract temporal database, since ranges are used to represent explicitly enumerated sets.

Temporal logic, in addition to its well-understood and clean foundations, provides a high level of abstraction for querying temporal databases, uncommon in the world of practical temporal query languages: It shields the user from the particular details of the concise encoding of temporal relations (in our case periods). In particular, all queries have a well-defined meaning in the temporal data model (where truth of facts is related to *individual time points*) and the interaction between the queries and the encoding is completely encapsulated in a temporal logic-to-ATSQL translation. In this way temporal logic serves as a natural common language for temporal databases that use potentially different concise encodings of sets of time points but are based on a common point-based data model. Such an approach provides convenient means for temporal database interoperability until (if at all) a standard temporal data model emerges. It has been shown that, unlike in temporal logic, the meaning of ATSQL queries *cannot* be always defined in a point-based temporal data model

[1]In this paper we use the term 'period' rather than the term 'interval' commonly used in temporal logic because the latter term conflicts with the SQL `INTERVAL`s, which are unanchored durations, such as "3 months".

as some ATSQL queries relate facts to time points (e.g., when *coalescing* is used) while others to periods (by allowing explicit access to the period-based encoding) [Toman 1996; Chomicki and Toman 1998]. This fact can be traced to ATSQL's choice of a *distinguished period-valued* temporal attribute to denote the *valid time* for a given tuple (cf. Section 2.2). Note however, that in this paper we use ATSQL as the *target* of our translation and we guarantee that only such ATSQL queries are generated for which point-based semantics is enforced by the use of coalescing (cf. Section 3.3).

The contributions of the paper can be summarized as follows:

(1) The paper provides a complete characterization of a subset of ATSQL queries equivalent to temporal logic in expressive power. This is achieved by establishing translations from temporal logic to ATSQL and from a subset of ATSQL back to temporal logic. This subset is characterized by a *locality* property: ATSQL is strictly more expressive than (safe) temporal logic even if only first-order features (e.g., no aggregation) and only a single temporal dimension (e.g., only valid time) are allowed.

(2) The paper gives a complete characterization of safety for queries formulated in temporal logic. Surprisingly, over ATSQL-like databases, additional syntactic restrictions (beyond those needed for standard relational calculus) are necessary to handle the temporal dimension. E.g., we can ask queries of the form "return all time points when a given closed query is false". The paper shows that safety for temporal logic queries depends only on the active *data* domain (all the constants that have ever appeared in any snapshot of the database), but is always independent of the temporal domain if a period-based representation is used. The paper also provides a syntactic characterization of a class of queries equivalent to the class of all safe temporal logic queries in expressive power (while safety is undecidable, as one would expect).

(3) Moreover, the translation from temporal logic to ATSQL produces "efficient" queries: the data complexity of the resulting ATSQL queries is in PTIME with respect to the *size* of the ATSQL database (unlike, e.g., IXRM [Lorentzos 1993] where the complexity depends on the size of the databases in the *abstract* temporal model that may be exponentially larger than their ATSQL counterparts, if they are finite at all [Chomicki and Toman 1998]). The existence of such a translation was not known previously and therefore temporal logic was often considered not to be practical. Our translation allows temporal logic to be used as a clean and declarative front-end to an ATSQL-based temporal DBMS. It can also be adapted to other temporal query languages, similar to ATSQL.

The paper is organized as follows: We start with a discussion of the basic framework in Section 2, including the syntax and semantics of temporal logic and ATSQL (in the case of ATSQL we introduce only constructs relevant to the development in this paper; for a full description see [Böhlen and Jensen 1996; Böhlen et al. 2001]). In Section 3 we give the translation from temporal logic to ATSQL. We conclude the section with an example and the discussion of some implementation issues. In Section 4 we discuss the reverse translation and relate the expressive power of (a subset of) ATSQL and temporal logic. Section 5 discusses the relations to other temporal query languages including the impact of the presented results.

2. BASIC FRAMEWORK

Before we start comparing temporal logic and ATSQL we need to set up a common formal framework suitable to both languages. In this paper we fix the structure of time to be *integer-like*: linear (totally ordered), discrete, unbounded in both the past and the future. However, our approach can be easily adopted to other structures of time, e.g., bounded in the past (natural-numbers-like time), or dense (rational-like time). The proposed translations change in only minor ways to accommodate such extensions. We also assume a single, fixed time granularity (one year).

All the references to time in this paper represent the *valid-time* that captures the relationship between individual time points and validity of facts *in reality* [Jensen et al. 1994]. *Transaction time*, which relates facts to the time when they are stored in the database, is not considered. (This is because the standard temporal logic deals only with a single temporal dimension.)

Finally, we restrict the discussion to the point-based view of temporal databases—the view where the truth (the presence of tuples in the appropriate relation instances) is associated with individual time points. This approach is adopted by temporal logic. On the other hand ATSQL uses period-valued temporal attributes to compactly represent contiguous periods of validity; for a detailed analysis of these choices and their consequences see [Chomicki and Toman 1998]. We use coalescing to enforce strictly point-based semantics. Coalescing is a unary operation on ATSQL temporal relations that merges value-equivalent tuples with adjacent or overlapping periods into a single tuple [Böhlen et al. 1996]. Throughout, we make sure that base relations as well as intermediate relations are coalesced.

2.1 Temporal Logic

Temporal logic is an *abstract query language*, a language defined with respect to the class of abstract temporal databases [Chomicki 1994; Chomicki and Toman 1998]. An *abstract temporal database*, in turn, is a database which captures the formal semantics of a temporal database by associating standard relational databases with individual time points (a fact is true at time t if an appropriate tuple exists in the database associated with the time point t; cf. Definition 2.2) without considering any particular representation issues.

It is possible to view an abstract temporal database in several different but equivalent ways [Chomicki 1994; Chomicki and Toman 1998]. We choose the *snapshot* view in which every time point is associated with a (finite) set of facts that hold at it. For integer-like time, abstract temporal databases can be viewed as infinite sequences of finite database states of the form $(\ldots, D_{-2}, D_{-1}, D_0, D_1, D_2, \ldots)$.

Example 2.1. *Figure 1 presents an example of an abstract temporal database, viewed as a sequence of states. The database represents information about Eastern European history, modeling the independence of various countries [Chomicki 1994]. Each fact specifies an independent nation and its capital. This database is used as a running example throughout the paper.*

Syntax. First-order temporal logic (FOTL) extends first order logic with binary temporal connectives **since** and **until**, and unary connectives $\bullet$ ("previous" or

Year	Timeslice
...	...
1024	{}
1025	$\{Indep(\text{'Poland'}, \text{'Gniezno'})\}$
...	...
1039	$\{Indep(\text{'Poland'}, \text{'Gniezno'})\}$
1040	$\{Indep(\text{'Poland'}, \text{'Cracow'})\}$
...	...
1197	$\{Indep(\text{'Poland'}, \text{'Cracow'})\}$
1198	$\{Indep(\text{'Czech Kingdom'}, \text{'Prague'}), Indep(\text{'Poland'}, \text{'Cracow'})\}$
...	...
1595	$\{Indep(\text{'Czech Kingdom'}, \text{'Prague'}), Indep(\text{'Poland'}, \text{'Cracow'})\}$
1596	$\{Indep(\text{'Czech Kingdom'}, \text{'Prague'}), Indep(\text{'Poland'}, \text{'Warsaw'})\}$
...	...
1620	$\{Indep(\text{'Czech Kingdom'}, \text{'Prague'}), Indep(\text{'Poland'}, \text{'Warsaw'})\}$
1621	$\{Indep(\text{'Poland'}, \text{'Warsaw'})\}$
...	...
1794	$\{Indep(\text{'Poland'}, \text{'Warsaw'})\}$
1795	{}
...	...
1917	{}
1918	$\{Indep(\text{'Czechoslovakia'}, \text{'Prague'}), Indep(\text{'Poland'}, \text{'Warsaw'})\}$
...	...
1938	$\{Indep(\text{'Czechoslovakia'}, \text{'Prague'}), Indep(\text{'Poland'}, \text{'Warsaw'})\}$
1939	$\{Indep(\text{'Poland'}, \text{'Warsaw'})\}$
1940	$\{Indep(\text{'Slovakia'}, \text{'Bratislava'})\}$
...	...
1944	$\{Indep(\text{'Slovakia'}, \text{'Bratislava'})\}$
1945	$\{Indep(\text{'Czechoslovakia'}, \text{'Prague'}), Indep(\text{'Poland'}, \text{'Warsaw'})\}$
...	...
1992	$\{Indep(\text{'Czechoslovakia'}, \text{'Prague'}), Indep(\text{'Poland'}, \text{'Warsaw'})\}$
1993	$\{Indep(\text{'Czech Republic'}, \text{'Prague'}), Indep(\text{'Poland'}, \text{'Warsaw'}),$ $Indep(\text{'Slovakia'}, \text{'Bratislava'})\}$
...	...

Fig. 1. Eastern European history: the abstract temporal database

"yesterday") and $\bigcirc$ ("next" or "tomorrow"). Informally, A **since** B is true in a state if A is true for states between when B was true and now (this state). A **until** B is true in a state if A will be true into the future until B will be true. The rest of the usual temporal connectives can be defined in terms of these, e.g.,

$\blacklozenge A \equiv$ `true` **since** `A`	A was true sometime in the past
$\Diamond A \equiv$ `true` **until** `A`	A will be true sometime in the future
$\blacksquare A \equiv \neg\blacklozenge\neg A$	A was true always in the past
$\Box A \equiv \neg\Diamond\neg A$	A will be true always in the future

In the rest of this paper we also consider the universal quantifier $(\forall x)A$ to be a shorthand for $\neg(\exists x)\neg A$, the implication $A \rightarrow B$ a shorthand for $\neg A \vee B$, etc. Formally, the semantics of the temporal logic queries is defined as follows:

DEFINITION 2.2. *Let $\sigma = (R_1, \ldots, R_k)$ be a relational schema. An abstract temporal database over σ is an integer-indexed sequence of database states $D = (\ldots, D_{-2}, D_{-1}, D_0, D_1, D_2, \ldots)$. Every database state D_i contains a relation (relation instance) r for each relation schema $R \in \sigma$. We define the semantics of temporal logic formulas in terms of a satisfaction relation $\models$ (a relation that relates true temporal logic formulas to a given abstract temporal database with respect to a valuation and a time point) and a valuation ν (a valuation is a mapping from variables to constants):*

—*$D, \nu, i \models R(x_1, \ldots, x_k)$ iff R is atomic and the tuple $(\nu(x_1), \ldots, \nu(x_k))$ is an element of the instance r of R in the database state D_i,*
—*$D, \nu, i \models \neg A$ iff $D, \nu, i \not\models A$,*
—*$D, \nu, i \models A \wedge B$ iff $D, \nu, i \models A$ and $D, \nu, i \models B$, similarly for $\vee$ and $\Rightarrow$,*
—*$D, \nu, i \models (\exists x)A$ iff there exists a c such that $D, \nu[x \mapsto c], i \models A$ where $\nu[X \mapsto c]$ is a valuation identical to ν except that it maps the variable x to the value c,*
—*$D, \nu, i \models A$ **since** B iff $\exists j(j < i \wedge D, \nu, j \models B \wedge \forall k(j < k \leq i \rightarrow D, \nu, k \models A))$,*
—*$D, \nu, i \models A$ **until** B iff $\exists j(j > i \wedge D, \nu, j \models B \wedge \forall k(i \leq k < j \rightarrow D, \nu, k \models A))$,*
—*$D, \nu, i \models \bullet A$ iff $D, \nu, i - 1 \models A$,*
—*$D, \nu, i \models \circ A$ iff $D, \nu, i + 1 \models A$.*

An answer *to a temporal logic query φ in D is the set $\varphi(D) = \{(i, \nu) : D, \nu, i \models \varphi\}$. Thus, temporal logic may be viewed as a natural extension of relational calculus.*

EXAMPLE 2.3. *Our first example is a query which does not relate different database states. The query*

$$(\exists city)(Indep(\text{'Poland'}, city) \wedge \neg(\exists city2)Indep(\text{'Slovakia'}, city2))$$

determines all years when Poland but not Slovakia was an independent country, i.e., the times when the query evaluates to true. For the particular database from Figure 1 these time points (years) are $\{1025, \ldots, 1794, 1918, \ldots, 1939, 1945, \ldots, 1992\}$.

EXAMPLE 2.4. *The second example relates different database states. The query*

$$(Indep(\text{'Poland'}, city) \wedge city \neq \text{'Cracow'}) \textbf{ since } Indep(\text{'Poland'}, \text{'Cracow'})$$

returns the name of the city that superseded Cracow as Poland's capital and the years when this city was the capital. In our sample database the answer is 'Warsaw' in years $\{1596, \ldots, 1794\}$.

EXAMPLE 2.5. *Consider the query [Chomicki 1994, p.515] "list all countries that lost and regained independence" over the abstract temporal database shown in Figure 1. This is formulated in temporal logic as:*

$$(\exists s1, s2)(\blacklozenge Indep(x, s1) \wedge \lozenge Indep(x, s2) \wedge (\forall s)\neg Indep(x, s)).$$

For a country and a year to result, the country will have been independent in the past, will be independent in the future, but is not independent for the year *it is associated with in the answer of the query (the "evaluation point"). The answers, again for our sample database, are 'Poland' (years* $\{1795, \ldots, 1917\}$ *and* $\{1940, \ldots, 1944\}$*), 'Czechoslovakia' (years* $\{1939, \ldots, 1944\}$*), and 'Slovakia' (years* $\{1945, \ldots, 1992\}$*).*

Example 2.6. *Another query may ask for all cities that were "first capitals of countries". A first capital is a city that became the capital of a country when the country gained independence (if a country gained independence several times, it may have several "first capitals"). In temporal logic this query is formulated as follows:*

$$(\exists x)(Indep(x, s) \textbf{ since } (\forall c')\neg Indep(x, c'))$$

In the example database the "first capitals" are

'Gniezno' *(years* $\{1025, \ldots, 1039\}$*),*
'Warsaw' *(years* $\{1918, \ldots, 1939\}$ *and* $\{1945, \ldots\}$*),*
'Prague' *(years* $\{1198, \ldots, 1620\}$*,* $\{1918, \ldots, 1938\}$*, and* $\{1945, \ldots\}$*), and*
'Bratislava' *(years* $\{1940, \ldots, 1944\}$ *and* $\{1993, \ldots\}$*).*

As indicated by the example queries, temporal logic provides a convenient means for expressing rather involved English queries in a natural way. However, the point-based semantics of temporal logic does not suggest an efficient implementation of such queries. Any implementation taking advantage of a compact period-based representation of temporal databases promises much better performance.

2.2 ATSQL

ATSQL [Böhlen and Jensen 1996; Böhlen et al. 2001] is a temporal extension of SQL–92. An early version of ATSQL has been proposed to the ISO international committee for standardization for incorporation into the SQL/Temporal standard, and has been implemented. Therefore, we use it as the target query language of our translation.

ATSQL Databases. A *period* is a pair $[a, b]$ where a is the *left endpoint* and b the *right endpoint*. The period $[a, b]$ is used to encode the set of points $\{t \,|a \le t \le b\}$. A *valid-time relation* is a finite relation where tuples are implicitly timestamped with periods. An *ATSQL database* is a finite collection of valid-time relations. Figure 2 shows an ATSQL database that *encodes* the abstract temporal database shown in Figure 1. Remember that throughout the paper we enforce that all the ATSQL temporal relations are coalesced. The timestamps are represented by maximal non-overlapping periods. This is *fundamental* for our translation of temporal logic queries to ATSQL to work correctly.

ATSQL Queries. ATSQL extends the query language of SQL–92 [Melton and Simon 1993]. The crucial concepts in ATSQL are *statement modifiers* (or *flags*) that can be prepended to queries to modify their *temporal behavior*. As a consequence ATSQL queries come in three flavors (in the rest of the paper we underline the ATSQL-specific additions to SQL):

(1) SQL–92 queries (without any additional flags) are executed on the temporal database with respect to the current time point (now).

Indep

Country	Capital	VTIME
Czech Kingdom	Prague	$[1198, 1620]$
Czechoslovakia	Prague	$[1918, 1938]$
Czechoslovakia	Prague	$[1945, 1992]$
Czech Republic	Prague	$[1993, \infty]$
Slovakia	Bratislava	$[1940, 1944]$
Slovakia	Bratislava	$[1993, \infty]$
Poland	Gniezno	$[1025, 1039]$
Poland	Cracow	$[1040, 1595]$
Poland	Warsaw	$[1596, 1794]$
Poland	Warsaw	$[1918, 1939]$
Poland	Warsaw	$[1945, \infty]$

Fig. 2. Eastern European history: the concrete ATSQL relation

(2) SQL–92 queries preceded by the `SEQ VT` flag are evaluated relative to *every snapshot* of the temporal database; the results are then collected in a temporal relation with timestamps corresponding to the evaluation point (cf. snapshot reducibility [Böhlen and Jensen 1996; Böhlen et al. 2001]).

(3) SQL–92 queries preceded by the `NSEQ VT` flag: in this case the processing of the timestamps is completely controlled by the query, rather than by some implicit mechanism of the underlying DBMS. In other words, the enclosed statement is executed with standard SQL–92 semantics. Timestamps are treated like all other attributes and no built-in temporal processing is performed. The manipulation of timestamps is made explicit using the following constructs:

—Given a period p, BEGIN(p) denotes the start point of p and END(p) the endpoint of p. We often use the shorthands p^- and p^+, respectively, to denote the endpoints.

—Given two time points b and e, $b \leq e$, PERIOD(b,e) denotes the period constructed out of two time points (points). Again, we often use the shorthand $[b, e]$.

—We use integer constants to denote time points (e.g., `1998` stands for the year 1998). We also include constants `TIMESTAMP 'beginning'` or $-\infty$, denoting the start of time, and `TIMESTAMP 'forever'` or ∞, denoting the end of time.

—To dislocate a given time point, t by one year[2] we write t `+ 1`. In similar fashion we can dislocate periods: to dislocate period $p = [b, e]$ by one year we write $p + 1$; the resulting period is $[b + 1, e + 1]$.

—Finally, we use FIRST(t,s) and LAST(t,s) to find the earlier and later, respectively, point of t and s.

The above syntax is used to manipulate timestamps in ATSQL select-blocks. First, we need to gain access to the implicit valid-time attributes of ATSQL relations: VTIME(R) denotes the timestamp associated with the range variable (tuple in the relation) R and substitutes for the lack of explicit temporal at-

[2]In this paper we assume that the valid-time has the granularity of a year. Thus `+1` is shorthand for `+INTERVAL '1' YEAR` and `-1` for `-INTERVAL '1' YEAR`.

tributes. The `WHERE` clause uses *temporal built-in predicates* to specify temporal relationships between periods. While all the relationships between two periods can be expressed using order relationships on their endpoints, ATSQL also supports Allen algebra-like comparisons of pairs of periods (which we won't use further in this paper). To be consistent with SQL–92, these relationships have a somewhat different meaning than the identically-named relationships in [Allen 1983]:

$$\begin{array}{lll}
\underline{I_1 = I_2} & \text{iff} & I_1^- = I_2^- \wedge I_1^+ = I_2^+ \\
\underline{I_1 \texttt{ CONTAINS } I_2} & \text{iff} & I_1^- \leq I_2^- \wedge I_1^+ \geq I_2^+ \\
\underline{I_1 \texttt{ MEETS } I_2} & \text{iff} & succ(I_1^+) = I_2^- \\
\underline{I_1 \texttt{ OVERLAPS } I_2} & \text{iff} & I_1^- \leq I_2^+ \wedge I_2^- \leq I_1^+ \\
\underline{I_1 \texttt{ PRECEDES } I_2} & \text{iff} & I_1^+ < I_2^- .
\end{array}$$

It is easy to see that the above relationships (and their Boolean combinations) can express all period relationships of [Allen 1983] and thus all possible topological relationships between two periods. Metric relationships—relationships that capture *distance* along the time line—can be captured using the above timestamp constructs. Finally, the SET VT p clause, which is part of the NSEQ VT statement modifier and precedes the actual query, defines p to be the resulting timestamp period for all tuples in the answer to the query (p is usually a function of the VTIME(R) attributes).

In addition, every query or table reference can be followed by a (VT) flag to enforce *coalescing* of the corresponding temporal table, that is, tuples with identical explicit attribute values whose valid-times overlap or are adjacent are merged into a single tuple, with a period equal to the union of the periods of the original tuples. As a side-effect, duplicates are eliminated.

EXAMPLE 2.7. *In order to determine the name of the city that superseded Cracow as Poland's capital (cf. Example 2.4), the query has to relate different database states. In ATSQL this means that we have to specify the* NSEQ VT *clause and the required temporal relationship. This results in the following ATSQL query:*

```
NSEQ VT
SET VT VTIME(i1)
  SELECT i1.Capital
  FROM Indep(VT) AS i1, Indep(VT) AS i2
  WHERE i1.Country = 'Poland'
  AND i2.Country = 'Poland'
  AND i2.Capital = 'Cracow'
  AND VTIME(i2) MEETS VTIME(i1)
```

EXAMPLE 2.8. *The formulation of a query becomes even simpler if it can be answered by looking at single snapshots. In this case the user simply specifies sequenced semantics when formulating a query, as illustrated in the following query, which determines all periods when Poland was independent but not Slovakia (cf. Example 2.3):*

```
( SEQ VT
    SELECT i1.Country
    FROM Indep(VT) AS i1
    WHERE i1.Country = 'Poland'
    AND NOT EXISTS ( SELECT *
                     FROM Indep(VT) AS i2
                     WHERE i2.Country = 'Slovakia' )
)(VT)
```

The proposed translation uses both the SEQ VT variant of the ATSQL queries (to translate the first-order fragments of the temporal logic queries) and the NSEQ VT queries (to translate the temporal connectives).

3. MAPPING TEMPORAL LOGIC TO ATSQL

In this section we introduce the main result of this paper: the translation of queries formulated in temporal logic to ATSQL. Similarly to mapping relational calculus queries to SQL, our translation has to identify a syntactic subset of domain-independent temporal queries that can be safely translated to ATSQL. The syntactic criterion is based on an extension of the criterion presented in [Abiteboul et al. 1995]. However, our approach can be analogously used in more complicated translations, e.g., [Van Gelder and Topor 1991]. We discuss several possible refinements in Section 3.5.

3.1 Correspondence of Temporal Databases

Before we can describe the actual translation of temporal formulas to ATSQL we need to establish a relationship between temporal databases (over which the semantics of temporal logic queries is defined) and ATSQL databases, the target of our translation.

DEFINITION 3.1. *Let $D = (\ldots, D_0, D_1, D_2, \ldots)$ be an abstract temporal database. The* support *of a temporal logic formula φ under a valuation ν is the set of time points $\{i : D, \nu, i \models \varphi\}$.*

The support of a formula for a fixed valuation (tuple) is the set of time points for which the particular tuple appears in the answer of the query (i.e., the formula is true in the given abstract temporal database after applying the substitution; cf. Definition 2.2). It follows immediately that the support for ground formulas (facts in particular) does not depend on the valuation. In this way the definition of the support yields the definition of the class of abstract temporal databases we are interested in.

DEFINITION 3.2. *An abstract temporal database is* finitary *if it contains a finite number of facts and the support of every fact can be represented as a finite union of periods.*

Not every abstract temporal database is finitary. For example, the database that contains a single fact $P('a')$ in every even-numbered state and whose every odd-numbered state is empty cannot be finitely represented by a union of periods. On the other hand, the class of finitary temporal databases captures exactly all ATSQL databases restricted to valid-time.

PROPOSITION 3.3. *Every ATSQL database represents a unique finitary abstract temporal database and every finitary abstract temporal database can be represented as an ATSQL database.*

In the rest of the paper we use $\|.\|$ to denote the mapping of ATSQL databases to the corresponding finitary abstract temporal databases.

3.2 Domain Independence and Range Restricted Queries

The actual translation of temporal logic queries to ATSQL is based on the semantic rules in Definition 2.2. However, there is a problem with a direct use of these rules: the interpretation of variables is relative to a potentially infinite universe of *data values*. Thus it is easy to formulate "unsafe" queries in temporal logic that produce non-finitary answers or use quantification over the infinite universe of data values (similarly to relational calculus [Abiteboul et al. 1995]). To avoid these problems we introduce the notion of domain-independent temporal logic queries.

DEFINITION 3.4. *Let φ be a FOTL query and D an abstract temporal database. We define the* active domain*, $adom(D, \varphi)$, to be the set of all data constants that appear in D or φ.*

The interpretation $\models_U$ is the $\models$ relation from Definition 2.2 relativized to the universe of data *values U. We assume that U always contains all the data constants appearing in the query and the temporal database.*

A temporal logic query φ is domain-independent *if for all sets U_1, U_2 such that $adom(D, \varphi) \subseteq U_1 \cap U_2$ we have $D, i, \nu \models_{U_1} \varphi \iff D, i, \nu \models_{U_2} \varphi$, for all $i \in Z$ and ν a valuation of free variables of φ over $U_1 \cup U_2$.*

Note that the above definition relativizes the interpretation of the queries only with respect to the *data domain*. The universe of time points is fixed to an integer-like linear order $(Z, \leq)$. It is easy to see that to obtain an answer to a domain-independent query it is sufficient to evaluate the query using the active domain interpretation, i.e., for $U = adom(D, \varphi)$. Moreover, the formula characterizing the active domain for a fixed query can be expressed uniformly for all D using another query in temporal logic.

LEMMA 3.5. *Let D be an abstract temporal database and φ a temporal query. Then there is a formula $adom_{D,\varphi}(x)$ such that $D, i, [x \mapsto c] \models adom_{D,\varphi}(x) \iff c \in adom(D, \varphi)$ for all $i \in Z$.*

PROOF. Let C be the set of all constants in φ and F the set of all formulas of the form $(\exists y_1, \ldots, y_k)(R(y_1, \ldots, y_k) \wedge x = y_i)$ for R a relation symbol in the schema of D, $k = \text{arity}(R)$, and $0 < i \leq k$. We define

$$adom_{D,\varphi}(x) ::= \bigvee_{\psi \in F} \Diamond\blacklozenge\psi \vee \bigvee_{c \in C} x = c.$$

□

The formula $adom_{D,\varphi}(x)$ is used to restrict variables in domain-independent temporal logic queries without changing their meaning.

We present a syntactic criterion that guarantees domain-independence of temporal logic queries. While domain independence itself is not decidable (the class of temporal logic queries contains all relational calculus queries), we show that the *safe-range queries*—those queries that pass our syntactic criterion—can express all the domain-independent queries in temporal logic.

The criterion is based on a modification of the criterion for the relational calculus queries [Abiteboul et al. 1995]; we treat the binary temporal connectives **since** and **until** as $\wedge$, and ignore the unary ones, $\blacklozenge, \blacksquare, \lozenge, \square, \bullet$, and $\circ$.

DEFINITION 3.6 (RANGE RESTRICTION (rr)). *Let φ be an arbitrary temporal query and $FV(\varphi)$ the set of free variables in φ. We define*

$$rr(\varphi) = \begin{cases} \{x_1, \ldots, x_k\} & \text{for } \varphi = R(x_1, \ldots, x_k) \\ rr(\phi) \cup rr(\psi) & \text{for } \varphi = \phi \wedge \psi, \phi \textbf{ until } \psi, \textit{ or } \phi \textbf{ since } \psi \\ rr(\phi) \cap rr(\psi) & \text{for } \varphi = \phi \vee \psi \\ \emptyset & \text{for } \varphi = \neg\psi \\ rr(\phi) - \{x\} & \text{for } \varphi = (\exists x)\phi : x \in rr(\phi); \text{ fails otherwise} \\ rr(\phi) & \text{for } \varphi = \blacklozenge\phi, \blacksquare\phi, \lozenge\phi, \square\phi, \bullet\phi, \textit{ or } \circ\phi \end{cases}$$

We say that a formula φ is safe-range *if $rr(\varphi) = FV(\varphi)$ and for all subformulas of φ of the form $(\exists x)\psi$ we have $x \in FV(\psi) \supset x \in rr(\psi)$.*

Note that this extension of the original criterion for relational calculus queries is the *strongest* possible: we treat **since** and **until** as $\wedge$ and ignore the unary temporal connectives. To achieve better results we would have to start with a stronger criterion for the first order case, e.g., [Van Gelder and Topor 1991].

THEOREM 3.7. *Let φ be a domain-independent query. Then there is an equivalent safe-range query.*

PROOF. Every domain-independent query φ can be correctly evaluated using the active-domain semantics. The active domain $adom(D, \varphi)$ can be defined uniformly for all D by a temporal logic query $adom_{D,\varphi}(x)$ (cf. Lemma 3.5). We add conjuncts that restrict the domain of every variable in all subformulas of φ. The resulting formula is equivalent to φ and is safe-range (follows from an easy induction on the structure of the formula). $\square$

Therefore, every domain-independent query can be equivalently asked using a safe-range query. Moreover,

LEMMA 3.8. *Let D be a finitary abstract temporal database and φ a safe-range query. Then $\varphi(D)$ is also finitary.*

PROOF. By induction on the structure of φ: it is sufficient to observe that (i) temporal connectives preserve the finitary properties and (ii) all variables in φ are range-restricted. $\square$

However, while domain independence is preserved under equivalence of queries, the rr criterion is not. Therefore the translation itself is defined in three steps:

(1) The first step corresponds to transforming the formula to a SRNF-like normal form [Abiteboul et al. 1995]; essentially we clean up the formula and remove superfluous connectives, especially double negations.
(2) In the second step we test all the variables in the cleaned-up formula for the safe-range property.

(3) For the formulas that pass the check—the safe-range formulas—we propagate the range restrictions so that all (significant) subformulas also became safe-range.

In the first step we define a normal form of temporal logic queries to *improve* our chances of discovering an equivalent safe-range reformulation of a given query:

DEFINITION 3.9 (SRNF_{TL}). *Let φ be an arbitrary temporal logic query. We define:*

Variable substitution*: We rename all quantified variables using unique names to avoid variable name clashes in the subsequent transformations.*

Removal of $\forall$*: We replace subformulas of the form $(\forall x)A$ by $\neg(\exists x)\neg A$.*

Removal of $\rightarrow$ and $\leftrightarrow$*: We replace implications $A \rightarrow B$ by $\neg A \vee B$, and similarly for the equivalences.*

Pushing down negations*: We use the following rules to push negations towards leaves of the formulas and to remove double negations:*

(1) $\neg\neg A \mapsto A$

(2) $(\exists x)A \mapsto A$ *if* $x \notin FV(A)$.

(3) $\neg(A \vee B) \mapsto (\neg A \wedge \neg B)$, *and* $\neg(A \wedge B) \mapsto (\neg A \vee \neg B)$

(4) $\neg\blacklozenge A \mapsto \blacksquare\neg A$, $\neg\blacksquare A \mapsto \blacklozenge\neg A$, $\neg\Diamond A \mapsto \Box\neg A$, *and* $\neg\Box A \mapsto \Diamond\neg A$.

(5) $\neg\bigcirc A \mapsto \bigcirc\neg A$, *and* $\neg\newmoon A \mapsto \newmoon\neg A$.

(6) $\neg(A\ \mathbf{since}\ B) \mapsto \blacksquare\neg A \vee (\neg A \wedge \neg B\ \mathbf{since}\ \neg A)$, *and*
$\neg(A\ \mathbf{until}\ B) \mapsto \Box\neg A \vee (\neg A \wedge \neg B\ \mathbf{until}\ \neg A)$.

A SRNF_{TL} *formula resulting from applying these rules to a temporal formula φ as long as possible is denoted* $\mathrm{SRNF}_{TL}(\varphi)$.

Note that the last rule for **since** and **until** is valid only for discrete time; for dense time we have to omit this rule.[3] For time bounded in the past we would also have to remove the part handling $\newmoon$ from rule (5), as the equivalence does not hold for time bounded in the past (natural numbers-like). Clearly, all the above transformations are equivalence-preserving.

LEMMA 3.10. $DB, \nu, i \models \varphi \iff DB, \nu, i \models \mathrm{SRNF}_{TL}(\varphi)$.

At the end of this step we are left with an equivalent and cleaned-up temporal formula. In addition it is easy to see that:

LEMMA 3.11. *Let φ be safe-range. Then* $\mathrm{SRNF}_{TL}(\varphi)$ *is safe-range.*

This lemma guarantees that applying the SRNF_{TL} transformation on the given query can only improve the chances that the query passes the *rr* criterion. Thus the *rr* criterion is always applied on the result of the SRNF_{TL} transformation.

The safe-range criterion *rr* assumes that the **since** and **until** connectives behave like $\wedge$. Unfortunately, the temporal connectives do not have the same commutative and distributive properties $\wedge$ has, e.g., $(\phi \vee \psi)\ \mathbf{until}\ \varphi \not\equiv (\phi\ \mathbf{until}\ \varphi) \vee (\psi\ \mathbf{until}\ \varphi)$. However, it is easy to see that the variable x in the formula $\neg P(x)\ \mathbf{until}\ Q(x)$ is

[3]The rule may also significantly increase the size of the resulting formula and we may not want to use it even in the case of discrete time.

safe-range by the atomic formula $Q(x)$: clearly if there is a valuation ν such that $DB, \nu, i \models \neg P(x)$ **until** $Q(x)$, then there must be another time point i' such that $DB, \nu, i' \models Q(x)$. Thus the formula $Q(x)$ gives us a range restriction for x. We exploit this fact to propagate the range restricting subformulas towards the leaves of the original formula using the following equivalences:

LEMMA 3.12. *ϕ **until** $\psi \equiv \phi$ **until** $(\blacklozenge\phi \wedge \psi)$, ϕ **until** $\psi \equiv (\phi \wedge \lozenge\psi)$ **until** ψ.*

PROOF. We show only the first equivalence; proof of the second one is analogous.
$\Rightarrow$: Let $DB, \nu, i \models \phi$ **until** ψ. Then by the definition of **until** we know that there is a $j > i$ such that $DB, \nu, j \models \psi$ and for all k such that $i \leq k < j$ and $DB, \nu, k \models \phi$. Thus $DB, \nu, j \models \blacklozenge\phi$ and using the definition of **until** we get $DB, \nu, i \models \phi$ **until** $(\blacklozenge\phi \wedge \psi)$.
$\Leftarrow$: Let $DB, \nu, i \models \phi$ **until** $(\blacklozenge\phi \wedge \psi)$. Then similarly to the previous case there is a $j > i$ such that $DB, \nu, j \models \blacklozenge\phi \wedge \psi$ and for all k such that $i \leq k < j$ and $DB, \nu, k \models \phi$. Clearly $DB, \nu, j \models \psi$ and thus $DB, \nu, i \models \phi$ **until** ψ. $\square$

A similar lemma holds for the **since** connective. Using these equivalences we can *move* the range-restricting subformulas between the left- and right-hand sides of the **since** and **until** connectives. In addition, we may need to move a range-restricting formula *into* the scope of a temporal connective:

LEMMA 3.13. *$\varphi \wedge (\phi$ **until** $\psi) \equiv \varphi \wedge (\phi$ **until** $(\blacklozenge\varphi \wedge \psi))$, $\varphi \wedge (\phi$ **until** $\psi) \equiv \varphi \wedge ((\phi \wedge \bigcirc\blacklozenge\varphi)$ **until** $\psi)$.*

PROOF. Again, we prove only the first statement.
$\Rightarrow$: Let $DB, \nu, i \models \varphi \wedge (\phi$ **until** $\psi)$. Then $DB, \nu, i \models \varphi$ and by the definition of **until** we know that there is a $j > i$ such that $DB, \nu, j \models \psi$ and for all k such that $i \leq k < j$ and $DB, \nu, k \models \phi$. Thus $DB, \nu, j \models \blacklozenge\varphi$ and using the definition of **until** we get $DB, \nu, i \models \varphi \wedge (\phi$ **until** $(\blacklozenge\varphi \wedge \psi))$.
$\Leftarrow$: Let $DB, \nu, i \models \varphi \wedge (\phi$ **until** $(\blacklozenge\varphi \wedge \psi))$. Similarly to the previous case $DB, \nu, i \models \varphi$ and there is a $j > i$ such that $DB, \nu, j \models \blacklozenge\phi \wedge \psi$ and for all k such that $i \leq k < j$ and $DB, \nu, k \models \phi$. Clearly $DB, \nu, j \models \blacklozenge\varphi$ is implied by $DB, \nu, i \models \varphi$ as $i < j$ and thus $DB, \nu, i \models \varphi \wedge (\phi$ **until** $\psi)$. $\square$

Similar laws hold for the remaining connectives, including the unary ones (cf. the rewriting rules in Definition 3.14). We use these laws in the final step of the conversion to propagate the range restricting subformulas towards the leaves of the query. In this way the final ATSQL query can always be evaluated bottom-up. This goal is achieved by a modified RANF transformation [Abiteboul et al. 1995].

DEFINITION 3.14 (RANF_{TL}). *Let φ be a safe-range temporal formula. A $\text{RANF}_{TL}(\varphi)$ is the result of applying the rules in Figure 3 together with commutativity and associativity of conjunction to φ, starting from the top-level connective.*

Clearly, all the rewriting rules preserve the meaning of the formula:

LEMMA 3.15. *$DB, \nu, i \models \varphi \iff DB, \nu, i \models \text{RANF}_{TL}(\text{SRNF}_{TL}(\varphi))$ for safe-range queries φ.*

PROOF. Follows from Lemmas 3.12 and 3.13, and standard equivalences for first order logic. $\square$

$$
\begin{array}{rcll}
A \wedge (B \vee C) & \mapsto & (A \wedge B) \vee (A \wedge C) & \text{(push into } \vee\text{)} \\
A \wedge (\exists x)B & \mapsto & A \wedge (\exists x)(A \wedge B) & \text{(push into } \exists\text{)} \\
A \wedge \neg B & \mapsto & A \wedge \neg(A \wedge B) & \text{(push into } \neg\text{)} \\
(A \wedge C) \textbf{ since } B & \mapsto & (A \wedge C) \textbf{ since } (\Diamond A \wedge B) & \text{(distribute in \textbf{since} left-to-right)} \\
(A \wedge C) \textbf{ until } B & \mapsto & (A \wedge C) \textbf{ until } (\blacklozenge A \wedge B) & \text{(distribute in \textbf{until} left-to-right)} \\
B \textbf{ since } (A \wedge C) & \mapsto & (\blacklozenge A \wedge B) \textbf{ since } (A \wedge C) & \text{(distribute in \textbf{since} right-to-left)} \\
B \textbf{ until } (A \wedge C) & \mapsto & (\Diamond A \wedge B) \textbf{ until } (A \wedge C) & \text{(distribute in \textbf{until} right-to-left)} \\
A \wedge (B \textbf{ since } C) & \mapsto & A \wedge ((\bullet \Diamond A \wedge B) \textbf{ since } C) & \text{(push into \textbf{since}, left side)} \\
A \wedge (B \textbf{ until } C) & \mapsto & A \wedge ((\circ \blacklozenge A \wedge B) \textbf{ until } C) & \text{(push into \textbf{until}, left side)} \\
A \wedge (C \textbf{ since } B) & \mapsto & A \wedge (C \textbf{ since } (\Diamond A \wedge B)) & \text{(push into \textbf{since}, right side)} \\
A \wedge (C \textbf{ until } B) & \mapsto & A \wedge (C \textbf{ until } (\blacklozenge A \wedge B)) & \text{(push into \textbf{until}, right side)} \\
A \wedge \blacklozenge B & \mapsto & A \wedge \blacklozenge(\Diamond A \wedge B) & \text{(push into } \blacklozenge\text{)} \\
A \wedge \Diamond B & \mapsto & A \wedge \Diamond(\blacklozenge A \wedge B) & \text{(push into } \Diamond\text{)} \\
A \wedge \blacksquare B & \mapsto & A \wedge \blacksquare(\Diamond A \wedge B) & \text{(push into } \blacksquare\text{)} \\
A \wedge \square B & \mapsto & A \wedge \square(\blacklozenge A \wedge B) & \text{(push into } \square\text{)} \\
A \wedge \bullet B & \mapsto & A \wedge \bullet(\circ A \wedge B) & \text{(push into } \bullet\text{)} \\
A \wedge \circ B & \mapsto & A \wedge \circ(\bullet A \wedge B) & \text{(push into } \circ\text{)}
\end{array}
$$

The rules are used when x is a variable range restricted in the subformula denoted by A, $x \in rr(A)$, and free but not range restricted in the subformula denoted by B, $x \in FV(B) - rr(B)$.

Fig. 3. RANF_{TL} transformation rules.

It is also easy to see that every rule in Figure 3 propagates x's restriction in A towards B.

LEMMA 3.16. *Let φ be a safe-range temporal formula. Then every subformula of $\text{RANF}_{TL}(\varphi)$ not rooted by $\wedge$ or $\neg$ is safe-range.*

PROOF. Assume that $\text{RANF}_{TL}(\varphi)$ contains a subformula not rooted by $\wedge$ or $\neg$ that is not safe-range. By case analysis we can show that φ is not safe-range (as none of the rules in Definition 3.14 is applicable by the assumption); a contradiction. □

Similarly to [Abiteboul et al. 1995] the RANF_{TL} rewriting terminates, as there are only finitely many subformulas in the original query. Moreover, every safe-range temporal query is domain independent:

LEMMA 3.17. *Let φ be a safe-range temporal query. Then φ is domain independent.*

PROOF. By Lemma 3.15 φ is equivalent to $\text{RANF}_{TL}(\varphi)$. Because query equivalence preserves domain independence, it is sufficient to show that $\text{RANF}_{TL}(\varphi)$ is domain-independent. This follows from an easy induction on the structure of $\text{RANF}_{TL}(\varphi)$ and Lemma 3.16 applied on every subformula of $\text{RANF}_{TL}(\varphi)$. □

This result, together with Theorem 3.7, shows that the classes of domain-independent queries and safe-range queries coincide.

EXAMPLE 3.18. *The query from Example 2.6 is indeed safe-range and can be transformed to* RANF_{TL} *as follows (using the "distribute in* **since** *left-to-right" rule):*

$$(\exists x)\,(Indep(x, c) \textbf{ since } (\Diamond Indep(x, c) \wedge \neg(\exists c')Indep(x, c'))$$

In the result every subquery is safe-range and thus amenable to translation to AT-SQL.

3.3 Translation to ATSQL

The next step in traditional translations is the translation to relational algebra. However, we have chosen ATSQL as our target language. The translation of temporal logic formulas in RANF$_{TL}$ to ATSQL is defined by induction on the structure of the formula. The input to this transformation is a safe-range temporal logic formula in RANF$_{TL}$. It is translated to ATSQL by repeating the following two steps:

(1) First the maximal non-temporal subformulas are translated to *sequenced* ATSQL queries; this is done using a simple RANF$_{TL}$ to SQL translation (patterned, e.g., after the RANF to Relational Algebra translation in [Abiteboul et al. 1995]).

(2) The translations of the subformulas are combined using the translations of the temporal connectives defined in the next section.

This process is repeated until the whole formula is translated.

3.3.1 *Temporal Logic Connectives.* We define translations of the individual temporal connectives to ATSQL as ATSQL query templates. The subformulas rooted by the temporal connectives are then translated to subqueries embedded into these templates.

The connectives since and until. Figure 4 graphically illustrates the semantics of **since** and **until** over periods. We have listed all possible temporal relationships [Allen 1983] between the truth periods of two formulas A and B. For each relationship we have determined the truth period of A **since** B and A **until** B respectively. More formally the truth periods of A **since** B and A **until** B are defined as follows.

$$A \;\textbf{since}\; B \mapsto [max(A^-, succ(B^-)), A^+] \text{ for } max(A^-, succ(B^-)) \leq A^+ \text{ and } B^+ \geq A^-$$

$$A \;\textbf{until}\; B \mapsto [A^-, min(A^+, pred(B^+))] \text{ for } A^- \leq min(A^+, pred(B^+)) \text{ and } A^+ \geq B^-.$$

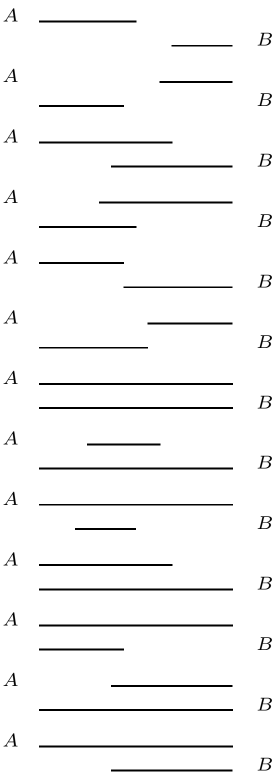
The reader may verify that these general expressions evaluated on any particular relationship given in Figure 4 result in the correct truth period.

These expressions are translated to ATSQL using the NSEQ VT modifier and specifying the final timestamp using the SET VT p clause. The additional conditions are translated into appropriate `WHERE` clause conditions. More precisely, A **since** B is translated to

```
NSEQ VT
SET VT PERIOD(LAST(BEGIN(VTIME(a0)), BEGIN(VTIME(a1))+1),END(VTIME(a0)))
  SELECT ...
  FROM A'(VT) AS a0, B'(VT) AS a1
  WHERE LAST(BEGIN(VTIME(a0)),BEGIN(VTIME(a1))+1) <= END(VTIME(a0))
    AND BEGIN(VTIME(a0)) <= END(VTIME(a1))
    AND ...
```

and A **until** B is translated to

Temporal relationship between formulas A and B



Temporal logic formula F	Truth period of formula F
A **since** B	$[\,]$
A **until** B	$[\,]$
A **since** B	$[\,]$
A **until** B	$[\,]$
A **since** B	$[succ(B^-), A^+]$
A **until** B	$[A^-, A^+]$
A **since** B	$[A^-, A^+]$
A **until** B	$[A^-, pred(B^+)]$
A **since** B	$[\,]$
A **until** B	$[A^-, A^+]$
A **since** B	$[A^-, A^+]$
A **until** B	$[\,]$
A **since** B	$[succ(B^-), A^+]$
A **until** B	$[A^-, pred(B^+)]$
A **since** B	$[A^-, A^+]$
A **until** B	$[A^-, A^+]$
A **since** B	$[succ(B^-), A^+]$
A **until** B	$[A^-, pred(B^+)]$
A **since** B	$[succ(B^-), A^+]$
A **until** B	$[A^-, A^+]$
A **since** B	$[succ(B^-), A^+]$
A **until** B	$[A^-, pred(B^+)]$
A **since** B	$[A^-, A^+]$
A **until** B	$[A^-, pred(B^+)]$
A **since** B	$[succ(B^-), A^+]$
A **until** B	$[A^-, pred(B^+)]$

Fig. 4. Period semantics of **since** and **until**

```
NSEQ VT
SET VT PERIOD(BEGIN(VTIME(a0)), FIRST(END(VTIME(a0)),END(VTIME(a1)))-1)
  SELECT ...
  FROM A'(VT) AS a0, B'(VT) AS a1
  WHERE BEGIN(VTIME(a0)) <= FIRST(END(VTIME(a0)),END(VTIME(a1))-1)
    AND BEGIN(VTIME(a1)) <= END(VTIME(a0))
    AND ...
```

where A' and B' are the results of applying the translation recursively on A and B, respectively. The `SELECT` lists of the ATSQL statements are derived from the sets of free variables occurring in A and B. Variables used in both A and B give rise to additional `WHERE` clause conditions that equate the corresponding attributes in A' and B'.

It is important to remember that the translations of non-atomic formulas A and B are *required* to produce coalesced temporal relations.

Example 3.19. *Consider a temporal database D containing two temporal relations $A(x,t) = \{(a,[0,5]),(b,[4,9])\}$ and $B(x,t) = \{(c,[8,10])\}$. Clearly, $D,t \models ((\exists x)A)$* **until** *$((\exists x)B)$ for $0 \leq t \leq 9$. It is easy to see that if coalescing has not been* *enforced at every step of the translation, e.g., if the relation A was not re-coalesced after projecting out the x attribute, the translation would not be correct any more. Indeed, applying the translation of* **until** *on the non-coalesced results of* $(\exists x)A$ *and* $(\exists x)B$ *would give us the result* $t \in [4, 9]$ *instead of the correct result* $t \in [0, 9]$.

3.3.2 *Specialized Mappings.* The translation of the remaining temporal connectives can be derived from the translations for **since** and **until**. While theoretically feasible, such an approach is cumbersome in practice and leads to unnecessarily complicated ATSQL statements. Moreover, introducing these specialized translations allows us to translate a wider class of temporal formulas to ATSQL (cf. Section 3.2).

The connectives $\blacklozenge$ *and* $\Diamond$. We illustrate how the definition of **since** is used to derive an efficient special purpose translation for $\blacklozenge$.

The formula $\blacklozenge B$ is equivalent to `true` **since** `B`. Therefore we take the definition of A **since** B (Section 3.3.1) and substitute A by `true`. We notice that the truth period of `true` is the whole time line which means that <u>`BEGIN(VTIME(a0))`</u> evaluates to $-\infty$ (beginning of time) and <u>`END(VTIME(a0))`</u> evaluates to $+\infty$ (end of time). After the obvious simplifications we obtain:

```
NSEQ VT
SET VT PERIOD(BEGIN(VTIME(a0))+1, TIMESTAMP 'forever')
  SELECT ...
  FROM B'(VT) AS a0
```

which is considerably less complex than the original statement. Similarly, we can use the definition of **until** to derive a translation for $\Diamond B$, namely

```
NSEQ VT
SET VT PERIOD(TIMESTAMP 'beginning',END(VTIME(a0))-1)
  SELECT ...
  FROM B'(VT) AS a0
```

The connectives $\blacksquare$ *and* $\Box$. For $\blacksquare A$, one can rewrite it as $\neg\blacklozenge\neg A$ and use the approach presented above. Unfortunately, this approach is not very practical as it may lead to formulas that cannot be translated (e.g., $\neg\blacklozenge\neg P(x)$ versus $\blacksquare P(x)$). Therefore we derive a ATSQL translation of $\blacksquare A$ from the definition

$$D, \nu, i \models \blacksquare A \text{ iff } \forall j (j < i \rightarrow D, \nu, j \models A).$$

This definition is easily expressed in ATSQL as follows (the `'beginning'` keyword stands for $-\infty$):

```
NSEQ VT
SET VT PERIOD(BEGIN(VTIME(a0)), END(VTIME(a0))+1)
  SELECT ...
  FROM A'(VT) AS a0
  WHERE BEGIN(VTIME(a0)) = TIMESTAMP 'beginning'
```

Again, coalescing of A' is crucial for the translation to work correctly. By analogy, a special purpose translation for $\Box A$ is derived as follows:

```
NSEQ VT
SET VT PERIOD(BEGIN(VTIME(a0))-1, END(VTIME(a0)))
  SELECT ...
  FROM A'(VT) AS a0
  WHERE END(VTIME(a0)) = TIMESTAMP 'forever'
```

The connectives ● and ○. We use discrete time to model the temporal domain in the ATSQL databases. Thus, in addition to the **since** and **until** connectives, we add temporal connectives that allow us to refer to the immediately previous (●) and the immediately following (○) time points.[4] The translation of these connectives is defined as follows: First we define the truth periods for $\bullet A$ and $\circ A$ with respect to the truth period of A:

$$\bullet A \mapsto [succ(A^-), succ(A^+)]$$
$$\circ A \mapsto [pred(A^-), pred(A^+)].$$

The result is translated to ATSQL using a definition of the corresponding valid-time clause that shifts the valid-time period by one in the appropriate direction. The translation of $\bullet A$ is

```
NSEQ VT
SET VT VTIME(a0)+1
  SELECT ...
  FROM A'(VT) AS a0
```

and the translation of $\circ A$ is

```
NSEQ VT
SET VT VTIME(a0)-1
  SELECT ...
  FROM A'(VT) AS a0
```

Similarly to the previous cases, the `SELECT` list is obtained from the set of free variables of A and A' is the ATSQL translation of A.

3.4 Putting it Together

Using the transformation defined in Section 3.2 we can convert every safe-range temporal query to an equivalent RANF$_{TL}$ formula. We have already shown that the RANF$_{TL}$ transformation preserves equivalence. We have also shown that the translations of the individual temporal connectives are correct. By composing these two steps we have:

THEOREM 3.20. *Let φ be a safe-range temporal logic formula and D an ATSQL database. Then $\varphi(\|D\|) = \|\varphi^{ATSQL}(D)\|$ where φ^{ATSQL} is the ATSQL translation of the temporal logic query φ.*

EXAMPLE 3.21. *Now we can finish the translation of the query from Example 2.6 to ATSQL. We already transformed the original query to* RANF$_{TL}$ *(cf. Example 3.18). The translation to ATSQL proceeds by applying the temporal logic*

[4]With respect to the chosen granularity of time which in this paper is a year.

to ATSQL translation rules on (maximal) nontemporal subqueries. The resulting query is:

```
01. NSEQ VT
02. SET VT PERIOD(LAST(BEGIN(VTIME(a2)),BEGIN(VTIME(a3))+1),
                                                      END(VTIME(a2)))
03.   SELECT a2.Capital
04.   FROM Indep(VT) AS a2,
05.        (
06.        SEQ VT
07.        SELECT a1.Country, a1.Capital
08.        FROM (
09.             NSEQ VT
10.             SET VT PERIOD(BEGIN(VTIME(a0))+1,TIMESTAMP 'forever')
11.               SELECT a0.Country, a0.Capital
12.               FROM Indep(VT) AS a0
13.             )(VT) AS a1
14.        WHERE a1.Country NOT IN (
15.             SELECT Country
16.             FROM Indep(VT) )
17.        )(VT) AS a3
18.   WHERE LAST(BEGIN(VTIME(a2)),BEGIN(VTIME(a3))+1) <= END(VTIME(a2))
19.     AND BEGIN(VTIME(a2)) <= END(VTIME(a3))
20.     AND a2.Country=a3.Country
21.     AND a2.Capital=a3.Capital
```

Lines 1–5 and 17–21 correspond to the translation of the **since** *connective, line 4 is the translation of the left-hand subquery (a simple table access), lines 5–16 are the translation of the right-hand side subformula. The right-hand side subformula is translated to a sequenced query that captures the translation of the negation (lines 6–8 and 14) applied to the translation of the inner temporal subformula rooted by the $\Diamond$ connective (lines 9–13) and a reference to a base table (lines 15–16).*

3.5 Refinement of the Translation

In Section 3.2 we described only the simplest version of the translation; we used a direct temporal extension of the translation presented in [Abiteboul et al. 1995]. However, such a direct extension has several drawbacks. We address some of them in this section.

Negation is pushed too deep during the SRNF_{TL} *phase.* This is necessary to find all double negations in the original formula and to eliminate them:

$$\neg(\neg Q(x)\ \mathbf{until}\ P(x)) \xrightarrow{\mathrm{SRNF}_{TL}} \Box Q(x) \vee (Q(x) \wedge \neg P(x)\ \mathbf{until}\ Q(x)).$$

However, if there are no hidden double negations (and this happens in many common cases) then the resulting formula in SRNF_{TL} does not improve range restrictedness of the variables and may be unnecessarily complicated.

$$P(x) \wedge \neg(R(x)\ \mathbf{until}\ Q(x)) \xrightarrow{\mathrm{SRNF}_{TL}} P(x) \wedge (\Box\neg R(x) \vee (\neg Q(x) \wedge \neg R(x)\ \mathbf{until}\ \neg R(x))).$$

In such cases we might be better off using the original query. The SRNF_{TL} translation can be easily modified to push the negations down in the query tree only if there is a chance they may cancel out in the subformulas.

The restricting formulas are unnecessarily duplicated. The second problem is intimately connected with the first one: by transforming the original query we may end up with a formula, where we need to propagate the bindings for variables across numerous connectives in order to obtain a formula in RANF_{TL}. However, this propagation often unnecessarily duplicates subformulas in the resulting query. For example:

$$\begin{aligned} P(x) \wedge \neg(\exists y)(Q(y) \wedge \neg R(x,y)) &\longrightarrow P(x) \wedge \neg(P(x) \wedge (\exists y)(Q(y) \wedge \neg R(x,y))) \\ &\longrightarrow P(x) \wedge \neg(\underline{P(x) \wedge} (\exists y)(P(x) \wedge Q(y) \wedge \neg R(x,y))) \end{aligned}$$

The underlined $P(x) \wedge$ part of the resulting formula is redundant. Note, that this is a general problem with the conversion proposed in [Abiteboul et al. 1995] (and most of the other proposals) rather than with its temporal extension—our example indeed uses only pure first order logic. This problem can be addressed in two ways:

(1) by restricting the depth to which $\neg$ gets pushed during the SRNF_{TL} translation (this is often the main source of this problem).

(2) by eliminating the superfluous restricting formulas; this can be achieved by an additional bottom-up pass through the generated query after the RANF_{TL} conversion.

Note that in the additional pass we are not trying to eliminate redundant parts of the original query; that is too difficult in general. We only eliminate unnecessary subformulas *introduced during the* RANF_{TL} *transformation.*

Nested temporal connectives (and conjunctions) create unnecessary ATSQL query blocks. The translation generates a separate ATSQL query block for every temporal connective (and conjunction). However, this approach may produce unnecessarily nested query blocks, that may be merged into a single block. For example, consider the query $\blacklozenge\blacksquare R(x)$. The translation produces the following code:

```
( NSEQ VT
  SET VT PERIOD(BEGIN(VTIME(R))+1, TIMESTAMP ´forever´)
    SELECT x
    FROM ( NSEQ VT
           SET VT PERIOD(BEGIN(VTIME(R)), END(VTIME(R))+1)
             SELECT x
             FROM   R(VT)
             WHERE  BEGIN(VTIME(R)) = TIMESTAMP ´beginning´
         )(VT) AS R
)(VT)
```

However, it is obvious we could merge the nested select blocks into a single equivalent block as the inner temporal operation ($\blacksquare$) preserves coalescing and thus the re-coalescing is not necessary.

```
( NSEQ VT
  SET VT PERIOD(BEGIN(VTIME(R))+1, TIMESTAMP ´forever´)
    SELECT x
    FROM   R(VT)
    WHERE  BEGIN(VTIME(R)) = TIMESTAMP ´beginning´
)(VT)
```

However, at this point we need to emphasize that the translations of the individual temporal connectives *do* require the input relations to be *coalesced* (cf. Example 3.19); note that the above example does not use projection as all data attributes of R are present in the result of the query. Therefore we can only merge those select blocks that preserve coalescing (as in the above example). The optimization corresponds to flattening conjunctions in the relational calculus queries. While in theory this step could be performed by a smart query optimizer, we are not aware of any implementation that would be able to perform such an optimization: most query optimizers are not able to perform arithmetic simplifications needed during the process (e.g., evaluating $\infty + 1 = \infty$ as used in our example). This observation is summarized by the following lemma:

LEMMA 3.22. *The translations of A* **since** *B, A* **until** *B, $\blacksquare A$, $\square A$, $\blacklozenge B$, $\lozenge B$, $\bullet B$, and $\circ B$ remain correct even if the ATSQL translation of B is not coalesced (however, A has to be coalesced in all cases). Moreover, if B is coalesced, the result of applying a temporal connective is coalesced too.*

This lemma, together with the observation that coalescing is preserved under temporal joins and differences (and not preserved by unions and projections) allows us to safely remove redundant coalescing operators in the translated formula.

4. MAPPING ATSQL TO TEMPORAL LOGIC

Establishing a mapping between a subset of ATSQL and temporal logic is less important from a practical point of view than establishing the mapping in the other direction, as described in the previous section. A possible application is the *decompilation* of ATSQL queries. However, the main purpose of establishing a mapping from ATSQL to temporal logic is to identify a subset of the former that has the same expressive power as the latter. This clarifies the issue of expressive power of some proposed restrictions of ATSQL, e.g., [Widom and Yang 1998].

Indeed, only a subset of ATSQL can be mapped back to temporal logic. There are several reasons for that. First, ATSQL inherits SQL–92's duplicate (bag-theoretic) semantics, while the semantics of temporal logic is set-theoretic. Second, ATSQL (like SQL–92) has aggregate operations that are not first-order expressible. Finally, ATSQL, similarly to any other two-sorted first-order logic based temporal query language (i.e., with explicit variables and quantification over a time sort [Chomicki and Toman 1998]), can express queries referring to multiple temporal contexts. Recently, it has been shown [Abiteboul et al. 1999; Toman and Niwinski 1996] that such queries are not expressible in temporal logic.

Therefore, to define the subset of ATSQL corresponding to temporal logic we introduce three syntactic restrictions. The first limits ATSQL to the SQL–92 constructs that can be mapped to relational algebra or calculus. The second guarantees set-theoretic semantics. The second and third guarantee that the subset of ATSQL has clear, point-based semantics.

DEFINITION 4.1. *An ATSQL query is* pure *if:*

(1) It does not use aggregate functions.

(2) Coalescing of periods is forced using `(VT)`*. As a side-effect, this ensures that no duplicates are generated.*

(3) the only set operations are `UNION` *and* `EXCEPT`*, the duplicate preserving variants* `UNION ALL` *and* `EXCEPT ALL` *are not allowed.*

(4) Sequenced queries do not use temporal built-in predicates at the top level.

Another restriction prohibits referring to multiple temporal contexts.

DEFINITION 4.2. *An ATSQL query Q is* local *if*

(1) every subquery in the `FROM` *clause is a coalesced valid-time ATSQL query (i.e., with a single valid-time attribute).*

(2) every subquery in the `WHERE` *clause refers only to attributes associated with its own range variables or to* data attributes *of the range variables of the enclosing select blocks. This implies that nested* `SELECT` *clauses cannot refer to the valid-times of range variables specified in the* `FROM` *clause of an enclosing* `SELECT`*.*

The condition on the `WHERE` subqueries may seem more strict than necessary—for the `FROM` subqueries we allow one shared temporal variable to be passed from the subquery to the enclosing query. However, the restriction is necessary as ATSQL does not syntactically guarantee coalescing within select blocks. Consider the following example:

EXAMPLE 4.3. *The following ATSQL query is not local.*

```
( NSEQ VT
  SET VT VTIME(a)
    SELECT * FROM R(VT) AS a
    WHERE EXISTS
      ( SELECT * FROM S(VT) AS b
        WHERE VTIME(b) CONTAINS VTIME(a)
      )
)(VT)
```

Our mapping translates a pure local ATSQL query Q to a temporal logic formula ϕ_Q. We define it step by step.

Temporal built-in predicates. We start by considering a simplified form of nonsequenced ATSQL queries:

```
( NSEQ VT
  SET VT PERIOD(q1,q2)
    SELECT *
    FROM R1(VT),...,Rm(VT)
    WHERE α
)(VT)
```

Additionally, we assume at first that:

(1) the query does not contain subqueries,

(2) the only references to valid time in the query are of the form <u>`VTIME(`v`)`</u> where v is one of the range variables $R_1, \ldots, R_m$, where R_i's range over base ATSQL relations.

We will subsequently relax these assumptions. For ATSQL queries obeying the above restrictions the mapping is defined in the following steps:

(1) We define a set of *special points* to contain all the time points explicitly referenced by the query (essentially the endpoints of valid periods for all relations in the `FROM` clause and their immediate predecessors/successors). These points are ordered linearly along the time line consistently with the `WHERE` clause α—we try all such linear orders one by one.

(2) Each of the above linear orderings divides the time line into isolated points and (open) periods, each of which corresponds to a set of time points that make a *local characteristic formula* written in FOTL true over a given database.

(3) Similarly, the timestamp for the result of the ATSQL query is represented by a disjunction of *global characteristic formulas* (again one for each of the partitions of the time line defined above).

In the rest of this section we develop this idea formally and show how it can be extended to a more general class of queries. For a query Q define the set S_Q of *special points* of Q to be

$$S_Q = \{R_1^-, \ldots, R_m^-, R_1^+, \ldots, R_m^+, R_1^- - 1, \ldots, R_m^- - 1, R_1^+ + 1, \ldots, R_m^+ + 1\} \cup \{-\infty, +\infty\}.$$

Also, for every range variable v of Q, define $l(v)$ as the literal $r(\bar{x})$ where r is the relation symbol of v and $\bar{x}$ is a vector of unique (logical) variables of length equal to the arity of r. In this way, a unique logical variable is also assigned to every attribute of v.

Now every temporal predicate in α (or its negation) can also be written as a disjunction of atomic order predicates relating some of the special points in S_Q. Therefore, there is one or more (but finitely many) strict linear orderings of S_Q that are consistent with α and the "natural" order of the special points, i.e., $R_i^- - 1 < R_i^- \leq R_i^+ < R_i^+ + 1$. (In such orderings some of the special points may coincide.) For every such ordering W we will construct a FOTL formula γ_W that encodes it. For every special point $p \notin \{-\infty, +\infty\}$ in W we determine the set of atomic formulas $T(p)$ that are true in W:

$$T(p) = \{l(v) : v^- \leq p \leq v^+\} \cup \{\neg l(v) : p = v^- - 1 \vee p = v^+ + 1\}.$$

For the $-\infty$ and ∞ special points the local characteristic formula is defined analogously as

$$T(-\infty) = \{l(v) : v^- = -\infty\} \;\text{ and }\; T(\infty) = \{l(v) : v^+ = \infty\}$$

as these two points are always the first and last ones, respectively, in any consistent linear order. Similarly, for every open period $i = (p_1, p_2)$ where p_1 and p_2 are special points and i does not contain any other special points of W we determine the set of atomic formulas $T(i)$ true in W,

$$T(i) = \{l(v) : i \subseteq (v^-, v^+)\}.$$

We now define the *local characteristic formula* Φ_p of a special point p in W as

$$\Phi_p = \bigwedge_{A \in T(p)} A$$

and the *local characteristic formula* Φ_i of an open period i in W as

$$\Phi_i = \bigwedge_{A \in T(i)} A.$$

An empty set $T(p)$ or $T(i)$ corresponds naturally to the local characteristic formula *true*. Now consider a pair of consecutive special points in W. They may correspond to consecutive time points, e.g., v^- and $v^- + 1$, or not, e.g., v^- and v^+. We will call the first the *point-point* case. In the second case we also have to consider the open period between those two points and have two cases: *point-period* and *period-point*.

The `SET VALID` clause of the query Q specifies a closed period $i_0 = [q_1, q_2]$ where both endpoints are special points in W. This period consists of a number of special points and open periods between those points. We construct *global characteristic formulas* Ψ_p for each special point in i_0 and Ψ_i for each (open) period in i_0. Such formulas encode the ordering W and the position of the point (resp. period) in this ordering. A formula Ψ_p is defined as the conjunction $\Psi_p^L \wedge \Psi_p^R$ where Ψ_p^L encodes the past of p in W and Ψ_p^R encodes the future of p in W (for periods this is defined similarly). The global characteristic formulas Ψ_p^L and Ψ_i^L are defined inductively with respect to the predecessor relation induced by the chosen linear order as follows:

$\Psi_{-\infty}^L = \Phi_{-\infty} \wedge \blacksquare \Phi_{-\infty}$
$\Psi_p^L = \Phi_p \wedge \CIRCLE \Psi_{p'}^L$ if p' is the predecessor of p (point-point), or
$\Psi_p^L = \Phi_p \wedge (\CIRCLE \Psi_{p'}^L \vee \CIRCLE \Psi_i^L)$ if $i = (p', p)$ is the predecessor of p (period-point), and
$\Psi_i^L = \Phi_i$ **since** Ψ_p^L if p is the predecessor of $i = (p, p')$ (point-period).

The predecessor relation is defined by the chosen linear order of special points W. In the cases where two special points are equal in the order, we handle them as a single point (and chose one of the names as a representative). The formulas Ψ_p^R (resp. Ψ_i^R) are symmetric to Ψ_p^L (resp. Ψ_i^L) with $\Circle$ used instead of $\CIRCLE$ and **until** instead of **since**. To get the formula γ_W we take the disjunction of all the global characteristic formulas corresponding to the points and periods in i_0.

Finally, the FOTL query ϕ_Q corresponding to the ATSQL query Q is obtained as the disjunction of all the formulas γ_W over all linear orders W of S_Q that are consistent with α.

4.1 Lifting the Restrictions

We will now show how the reverse translation can be extended from the restricted ATSQL queries to all pure local ones.

Nontemporal conditions. Every condition $a_1\theta a_2$ is translated as $x_1\theta x_2$ where x_1 (resp. x_2) is the variable corresponding to the attribute a_1 (resp. a_2). A condition $a_1\theta c$ is translated similarly.

Nested subqueries. The `FROM` subqueries Q_i are translated to FOTL formulas ϕ_{Q_i}. These formulas are then used to form the local characteristic formulas Φ in the same way base relations are used. This is possible as the results of these queries must be coalesced valid-time ATSQL relations (views). Finally, a `UNION` appearing as a subquery in the `FROM` clause is translated as a disjunction and `EXCEPT` as "and not".

Subqueries in the `WHERE` clause are transformed to subqueries in the `FROM` clause using the standard SQL equivalences: `EXISTS` and `IN` to a coalesced `FROM` subquery, `NOT EXISTS` and `NOT IN` to a set difference nested in the `FROM` clause. Note that as the original query is local, this transformation yields coalesced valid-time subqueries in the `FROM` clause and thus the result is an equivalent local query. Also, in the cases of `EXISTS` and `NOT EXISTS`, the generated subquery timestamps are trivial ($[-\infty, \infty]$) and lead to further possibilities of simplification.

Temporal constants and expressions. To handle a temporal constant c we need to introduce a constant 0-ary relation r_c and treat this relation as if it were part of every `FROM` clause. Clearly, $r_c^- = r_c^+$. (It is enough to have just one constant relation, e.g., `ZERO`, and define the remaining ones using $\bullet$ or $\circ$.) For every occurrence of a temporal expression $\underline{\texttt{VTIME}(v) + k}$ we need to add the points

$$\underline{\texttt{VTIME}(v) + 1}, \ldots, \underline{\texttt{VTIME}(v) + k}, \underline{\texttt{VTIME}(v) + k + 1}$$

to the set of special points of the query. Similarly for $\underline{\texttt{VTIME}(v) - k}$. It is easy to see that every pure local ATSQL query can be rewritten to a query in which all the temporal expressions are either constants or of the form $\underline{\texttt{VTIME}(v) \pm k}$. In particular, the occurrences of $\underline{\texttt{FIRST}}$/$\underline{\texttt{LAST}}$ can be eliminated by splitting the query.

Assume now that the `WHERE` clause of a pure local ATSQL query Q is a conjunction of temporal predicates, nontemporal conditions and conditions with subqueries. The formula ϕ_Q is a conjunction of the formulas obtained by translating each of those separately and conjoining the result with $l(v)$ for every range variable v in the query (using a consistent naming of variables that correspond to relation attributes). A `SELECT` list with attributes $A_1, \ldots, A_n$ is translated into an existential quantifier prefix consisting of all the variables corresponding to the attributes that are not among $A_1, \ldots, A_n$ but come from the range variables of Q.

Sequenced queries. Pure sequenced queries do not contain temporal predicates, except in subqueries. Therefore the main query is translated as in the standard translation of SQL into relational calculus. Temporal subqueries are translated as nested subqueries of nonsequenced queries (see above).

THEOREM 4.4. *For every pure local ATSQL query Q, there is a temporal logic formula β_Q such that for every ATSQL database D, a tuple $\bar{a}$ timestamped by an period i belongs to the answer of Q over D iff $||D||, \nu, t \models \beta_Q$ for every time point t in i (where $||D||$ is the abstract temporal database corresponding to D and ν is the valuation that maps the free variables of β_Q to $\bar{a}$).*

EXAMPLE 4.5. *Assume that a relation* A *has two attributes,* x *and* y, *and a relation* B *one attribute,* z. *Consider the following (pure local) ATSQL query.*

```
( NSEQ VT
  SET VT VTIME(b)
    SELECT *
    FROM A(VT) AS a, B(VT) AS b
    WHERE VTIME(a) CONTAINS VTIME(b)
      AND a.x=b.z
)(VT)
```

We extend the notation for denoting the endpoints of periods to tuple variables as follows: $\mathtt{x}^-$ *denotes* BEGIN(VTIME(x)) *and* $\mathtt{x}^+$ *denotes* END(VTIME(x)). *The* WHERE *clause of the above query generates the following partial order of endpoints:*

$$\mathtt{a}^- \le \mathtt{b}^- \wedge \mathtt{b}^+ \le \mathtt{a}^+.$$

The following points are special:

$$\mathtt{a}^- - 1, \mathtt{a}^-, \mathtt{a}^+, \mathtt{a}^+ + 1, \mathtt{b}^- - 1, \mathtt{b}^-, \mathtt{b}^+, \mathtt{b}^+ + 1.$$

Consider now all linear orders of special points that are consistent with the above partial order, for example, the linear order O_1*:*

$$-\infty < \mathtt{a}^- - 1 < \mathtt{a}^- < \mathtt{b}^- - 1 < \mathtt{b}^- < \mathtt{a}^+ = \mathtt{b}^+ < \mathtt{a}^+ + 1 = \mathtt{b}^+ + 1 < +\infty.$$

The local characteristic formulas Φ_p *and* Φ_i *corresponding to this order are as follows:*

$$\begin{aligned}
\Phi_{-\infty} \equiv \Phi_{\infty} &\equiv \mathit{true} \\
\Phi_{\mathtt{a}^- -1} &\equiv \neg A(x,y) \\
\Phi_{\mathtt{a}^-} &\equiv A(x,y) \\
\Phi_{\mathtt{b}^- -1} &\equiv A(x,y) \wedge \neg B(z) \\
\Phi_{\mathtt{b}^-} \equiv \Phi_{\mathtt{b}^+} \equiv \Phi_{\mathtt{a}^+} &\equiv A(x,y) \wedge B(z) \\
\Phi_{\mathtt{b}^+ +1} \equiv \Phi_{\mathtt{a}^+ +1} &\equiv \neg A(x,y) \wedge \neg B(z)
\end{aligned}
\qquad
\begin{aligned}
\Phi_{(-\infty,\mathtt{a}^- -1)} &\equiv \mathit{true} \\
\Phi_{(\mathtt{a}^-,\mathtt{b}^- -1)} &\equiv A(x,y) \\
\Phi_{(\mathtt{b}^-,\mathtt{b}^+)} &\equiv A(x,y) \wedge B(z) \\
\Phi_{(\mathtt{b}^+ +1,+\infty)} &\equiv \mathit{true}
\end{aligned}$$

The global characteristic formulas have to describe the final timestamp, in our case the period $[\mathtt{b}^-, \mathtt{b}^+]$. *This period consists of, in the chosen order* O_1, *three items: the two isolated special points* $\mathtt{b}^-$ *and* $\mathtt{b}^+$ *and the open period* $(\mathtt{b}^-, \mathtt{b}^+)$. *The characteristic formulas corresponding to this order are as follows (going from left to right in the order and simplifying trivial cases):*

$$\begin{aligned}
\Psi^L_{-\infty} \equiv \Psi^L_{(-\infty,\mathtt{a}^- -1)} &\equiv \mathit{true} \\
\Psi^L_{\mathtt{a}^- -1} &\equiv \neg A(x,y) \\
\Psi^L_{\mathtt{a}^-} &\equiv A(x,y) \wedge \bullet\neg A(x,y) \\
\Psi^L_{(\mathtt{a}^-,\mathtt{b}^- -1)} &\equiv A(x,y) \textbf{ since } \Psi^L_{\mathtt{a}^-} \\
\Psi^L_{\mathtt{b}^- -1} &\equiv A(x,y) \wedge \neg B(z) \wedge (\bullet\Psi^L_{\mathtt{a}^-} \vee \bullet\Psi^L_{(\mathtt{a}^-,\mathtt{b}^- -1)}) \\
\Psi^L_{\mathtt{b}^-} &\equiv A(x,y) \wedge B(z) \wedge \bullet\Psi^L_{\mathtt{b}^- -1} \\
\Psi^L_{(\mathtt{b}^-,\mathtt{b}^+)} &\equiv (A(x,y) \wedge B(z)) \textbf{ since } \Psi^L_{\mathtt{b}^-} \\
\Psi^L_{\mathtt{b}^+} &\equiv A(x,y) \wedge B(z) \wedge (\bullet\Psi^L_{\mathtt{b}^-} \vee \bullet\Psi^L_{(\mathtt{b}^-,\mathtt{b}^+)})
\end{aligned}$$

In a similar way, going from right to left, $\Psi^R_{\mathtt{b}^-}$, $\Psi^R_{(\mathtt{b}^-,\mathtt{b}^+)}$, *and* $\Psi^R_{\mathtt{b}^+}$ *are obtained. All together* $(\Psi^L_{\mathtt{b}^-} \wedge \Psi^R_{\mathtt{b}^-} \vee \Psi^L_{(\mathtt{b}^-,\mathtt{b}^+)} \wedge \Psi^R_{(\mathtt{b}^-,\mathtt{b}^+)} \vee \Psi^L_{\mathtt{b}^+} \wedge \Psi^R_{\mathtt{b}^+})$ *conjoined with the nontemporal* *condition $x = z$ characterizes the result of the original query in the ordering O_1. To obtain the final formula we form a disjunction of formulas constructed analogously for all linear orders of special points that are consistent with the partial order of endpoints generated by the query.*

The translation from temporal logic to ATSQL presented in the previous section produces pure local ATSQL queries. Thus:

COROLLARY 4.6. *Temporal logic and pure local ATSQL have the same expressive power as query languages.*

The following is a natural next question to ask: Is there a logical query language equivalent to full ATSQL? The lack of aggregate functions in temporal logic can be remedied by a syntactic extension of the language, along the lines of one proposed for relational calculus [Klug 1982]. The other requirements from Definition 4.1 are more fundamental. Relaxing them calls for a temporal logic that is not point- but period-based.

The restriction to local queries is also critical. Pure ATSQL has the same expressive power as two-sorted first-order logic in which there is a separate sort for time. It has been recently shown [Abiteboul et al. 1999; Toman and Niwinski 1996] that temporal logic is strictly less expressive than the above two-sorted logic. Thus, there cannot be a translation from ATSQL to temporal logic that works for all pure queries.

5. RELATED WORK

Despite extensive studies of theoretical properties of temporal logic and other logic-based temporal query languages [Gabbay and McBrien 1991; Clifford et al. 1994; Abiteboul et al. 1999; Toman and Niwinski 1996], there has been surprisingly little work on implementations of these languages. Often the main reason for disregarding temporal logic (and the derived query languages) was the assumption that queries in these languages can only be be evaluated over an explicitly constructed snapshot temporal databases. Indeed, some of the early works have pursues such an approach [Tuzhilin and Clifford 1990]. which results in exponenential space blow-up (in the number of bits) for finite databases. Moreover, this approach completely fails for infinite (but finitary) temporal databases. Our approach avoids the above problems while preserving the declarative nature of temporal logic. We note that similar properties are enjoyed by some other translation-based approaches, e.g., [Toman 1997; Toman 1998]).

Our translation converts safe-range temporal logic formulas to ATSQL. While conveniently using ATSQL's *sequenced queries* to translate the nontemporal parts of the queries, the translation could have used any other well-behaved temporal query language as its target. The requirements the target temporal query language must satisfy are very modest: It has to operate over temporal databases based on a single distinguished temporal attribute ranging over periods and it has to enforce coalescing. The majority of temporal data models and languages satisfy these requirements. The sequenced queries needed for the translation of temporal logic can be simulated in a first-order complete temporal query language with temporal attributes ranging over periods [Toman 1996; Toman 1997]. Therefore temporal logic can serve as a convenient tool for interoperability of temporal databases represented using one of these models (until a single standard emerges).

A more general translation from two-sorted first-order logic to ATSQL, which is of clear practical interest, is considerably more complicated than the translation from temporal logic to ATSQL given in the present paper. In [Toman 1996] a translation from a point-based two-sorted first-order logic to a period-based temporal query language was proposed. This approach was subsequently extended to a SQL-based temporal query language SQL/TP [Toman 1997; Toman 1998]. This translation could serve as a translation from two-sorted first-order logic to ATSQL. There are two subtle points about this translation:

—It generates non-local ATSQL queries. Indeed, the results [Abiteboul et al. 1999; Toman and Niwinski 1996] show that there cannot be a translation of the two-sorted first-order logic to local ATSQL queries (and views).

—In general the generated query may be exponential in the size of the input query.[5] [Toman 1997] defined a syntactic criterion that guarantees only polynomial (linear) increase in size for a subclass of the two-sorted first-order logic queries. Moreover, this subclass contains the first-order temporal logic.

The reverse translation, while of little practical use, provides the crucially needed insight into how the various practical query languages for temporal databases compare in expressive power. It allows us to classify temporal extensions of SQL (and related languages) that are essentially equivalent to temporal logic (with **since** and **until** connectives).[6] The notion of *locality* (the restriction to a single temporal context in any subquery) plays a major role in this classification:

Languages equivalent to temporal logic: TQuel [Snodgrass 1987] and its corresponding temporal algebra [McKenzie and Snodgrass 1991], HSQL [Sarda 1993], temporal algebra [Widom and Yang 1998] used for temporal data warehousing, TL [Clifford et al. 1994], TA [Clifford et al. 1994].

Languages whose fragments are equivalent to two-sorted first-order logic (and thus strictly stronger than temporal logic): ATSQL [Böhlen and Jensen 1996; Böhlen et al. 2001] and SQL/Temporal [Snodgrass et al. 1996] (using explicit coercion of temporal attributes to data attributes and vice versa), IXRM [Lorentzos 1993] (description incomplete), SQL/TP [Toman 1997; Toman 1998], TC [Clifford et al. 1994].

The so-called "grouped" languages [Clifford et al. 1994] fall, strictly speaking, outside this classification, since they use set-valued variables and therefore should be treated as *second-order* (like the query languages for complex object databases [Abiteboul et al. 1995]). The discussion of those languages is beyond the scope of the present paper.

An important consequence of the above classification is that only the first group of languages can be implemented by a temporal relational algebra over the universe of temporal relations (with a single distinguished valid-time attribute) [Toman and

[5]However, this may happen even in the translation from relational calculus to algebra [Abiteboul et al. 1995].

[6]More precisely, their *first-order* fragments are equivalent to FOTL.

Niwinski 1996]. All the languages in the second group require relations with multiple temporal attributes to store intermediate results during bottom-up query evaluation. Moreover, there is no upper bound on the number of temporal attributes needed even if the top-level query is boolean. For a more comprehensive discussion of the above issues see [Chomicki and Toman 1998; Toman 1998].

6. SUMMARY

We have established an exact correspondence between temporal logic and a syntactically defined subset of ATSQL. The translation from temporal logic to ATSQL allows the efficient implementation of temporal logic queries within a temporal database management system supporting ATSQL.

Future work includes both theoretical and practical issues:

—On the theoretical side we explore extensions to temporal logic that capture various features of ATSQL missing in standard linear-time temporal logic. This includes handling duplicate semantics and aggregation. Another extension of the proposed approach is connected with *dense* models of time. This would require first extending ATSQL to such a domain, including support for half-open and open periods, and then extending the mapping introduced here. In addition, as pointed out in Section 4, the translations assume a point-based temporal model—this requirement translates to requiring coalesced ATSQL relations. Relaxing this requirement on the ATSQL side requires an period-based temporal data model and in turn would require an period-based temporal logic. Another extension is needed due to ATSQL's capability to handle both valid and transaction time. Handling two independent temporal dimensions in temporal logic is possible. However, it is not clear how such a logic can be translated to ATSQL to seamlessly operate over the bitemporal data model of ATSQL.

—An entirely different set of problems appears if the translated query language is not temporal logic but rather one of the deductive or fixpoint temporal query languages [Baudinet et al. 1993; Abiteboul et al. 1999]. Then ATSQL is no longer sufficient as the target query language but has to be extended with fixpoints or recursion. No such extension exists at this time.

—On the practical side the translation from temporal logic to ATSQL can serve as a declarative front end to a temporal DBMS based on ATSQL (or, as pointed out in Section 5, any other temporal extension of SQL that meets very mild requirements). In addition the translations of the **since** and **until** connectives may lead to extending the usual set operations in sequenced ATSQL (such as `UNION` or `EXCEPT` that operate on individual snapshots of the temporal database) with their temporal counterparts `SINCE` and `UNTIL` that would allow the users to relate different snapshots of the database without the need to use the (vastly more complex) nonsequenced ATSQL semantics.

An additional, more general topic of interest is the issue of eliminating unnecessary coalescing operators from temporal queries. Lemma 3.22 describes cases where coalescing can safely be removed in the presence of temporal connectives. However, the more general question of sufficient and/or necessary conditions for removal of coalescing operators in ATSQL queries remains open.

ACKNOWLEDGMENTS

We are grateful to Rick Snodgrass for participating in the preparation of the first version of this paper and his continued guidance. Jan Chomicki's work was partially supported by NSF grant IRI-9632870. David Toman's work was supported by NATO/NSERC PDF grant at the University of Toronto and NSERC grant RPGIN 217353-99 at the University of Waterloo. Part of this work was completed while visiting BRICS, Centre for Basic Research in Computer Science of the Danish National Research Foundation at the University of Aarhus. Michael Böhlen's work was supported in part by a grant from the Nykredit Corporation, by grants from the Danish Technical Research Council, and by the CHOROCHRONOS project, funded by the European Commission DG XII Science, Research and Development, as a Networks Activity of the Training and Mobility of Researchers Programme, contract no. FMRX-CT96-0056.

REFERENCES

Abiteboul, S., Herr, L., and Van den Bussche, J. 1999. Temporal Connectives Versus Explicit Timestamps to Query Temporal Databases. *J. Comput. Syst. Sci. 58*, 1, 54–68.

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Allen, J. F. 1983. Maintaining Knowledge about Temporal Intervals. *Commun. ACM 26*, 11, 832–843.

Baudinet, M., Chomicki, J., and Wolper, P. 1993. Temporal Deductive Databases. In A. U. Tansel, J. Clifford, S. K. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass Eds., *Temporal Databases: Theory, Design, and Implementation*, pp. 294–320. Benjamin/Cummings.

Böhlen, M., Snodgrass, R. T., and Soo, M. D. 1996. Coalescing in Temporal Databases. In *International Conference on Very Large Data Bases* (1996), pp. 180–191.

Böhlen, M. H. 1994. *Managing Temporal Knowledge in Deductive Databases*. Ph. D. thesis, ETH Zürich, Zürich, Switzerland.

Böhlen, M. H. and Jensen, C. S. 1996. Seamless Integration of Time into SQL. Technical Report R-96-2094, Institute for Electronic Systems, Aalborg University.

Böhlen, M. H., Jensen, C. S., and Snodgrass, R. T. 2001. Temporal Statement Modifiers. *ACM Transactions on Database Systems*. (to appear).

Chomicki, J. 1994. Temporal Query Languages: A Survey. In *Temporal Logic, First International Conference* (1994), pp. 506–534. Springer-Verlag, LNAI 827.

Chomicki, J. 1995. Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *ACM Transactions on Database Systems 20*, 2 (June), 149–186.

Chomicki, J. and Toman, D. 1995. Implementing Temporal Integrity Constraints Using an Active DBMS. *IEEE Transactions on Knowledge and Data Engineering 7*, 4, 566–582.

Chomicki, J. and Toman, D. 1998. Temporal Logic in Information Systems. In J. Chomicki and G. Saake Eds., *Logics for Databases and Information Systems*, Chapter 3, pp. 31–70. Kluwer.

Clifford, J. and Croker, A. 1987. The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In *IEEE International Conference on Data Engineering* (1987), pp. 528–537.

Clifford, J., Croker, A., and Tuzhilin, A. 1994. On Completeness of Historical Relational Query Languages. *ACM Transactions on Database Systems 19*, 1, 64–116.

Gabbay, D. M., Hodkinson, I. M., and Reynolds, M. 1994. *Temporal Logic: Mathematical Foundations and Computational Aspects*. Oxford University Press.

Gabbay, D. M. and McBrien, P. 1991. Temporal Logic and Historical Databases. In *International Conference on Very Large Data Bases* (1991), pp. 423–430.

Gadia, S. K. 1988. A Homogenous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems 13*, 4, 418–448.

Gadia, S. K. 1993. Temporal Databases: A Prelude to Parametric Data. In A. U. Tansel, J. Clifford, S. K. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass Eds., *Temporal Databases: Theory, Design, and Implementation*, pp. 28–66. Benjamin/Cummings.

Gertz, M. and Lipeck, U. W. 1996. Deriving optimized integrity monitoring triggers from dynamic integrity constraints. *Data and Knowledge Engineering 20*, 2, 163–193.

Jensen, C. S., Clifford, J., Elmasri, R., Gadia, S. K., Hayes, P., and Jajodia, S. 1994. A Glossary of Temporal Database Concepts. *SIGMOD Record 23*, 1, 52–64.

Klug, A. C. 1982. Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *Journal of the ACM 29*, 3, 699–717.

Lipeck, U. W. and Saake, G. 1987. Monitoring Dynamic Integrity Constraints Based on Temporal Logic. *Information Systems 12*, 3, 255–269.

Lorentzos, N. A. 1993. The Interval-extended Relational Model and Its Application to Valid-time Databases. In A. U. Tansel, J. Clifford, S. K. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass Eds., *Temporal Databases: Theory, Design, and Implementation*, pp. 67–91. Benjamin/Cummings.

McKenzie, L. J. and Snodgrass, R. 1991. Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys 23*, 4 (December), 501–543.

Melton, J. and Simon, A. R. 1993. *Understanding the new SQL: A Complete Guide*. Morgan Kaufmann Publishers.

Navathe, S. B. and Ahmed, R. 1993. Temporal Extensions to the Relational Model and SQL. In A. U. Tansel, J. Clifford, S. K. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass Eds., *Temporal Databases: Theory, Design, and Implementation*, pp. 92–109. Benjamin/Cummings.

Sarda, N. L. 1990. Extensions to SQL for Historical Databases. *IEEE Transactions on Knowledge and Data Engineering 2*, 2, 220–230.

Sarda, N. L. 1993. HSQL: A Historical Query Language. In A. U. Tansel, J. Clifford, S. K. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass Eds., *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings.

Sistla, A. P. and Wolfson, O. 1995. Temporal Triggers in Active Databases. *IEEE Transactions on Knowledge and Data Engineering 7*, 3, 471–486.

Snodgrass, R. T. 1987. The Temporal Query Language TQuel. *ACM Transactions on Database Systems 12*, 2, 247–298.

Snodgrass, R. T. Ed. 1995. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers.

Snodgrass, R. T., Böhlen, M. H., Jensen, C. S., and Steiner, A. 1996. Adding Valid Time to SQL/Temporal. ISO/IEC JTC1/SC21/WG3 DBL MAD-146r2 21/11/96, (change proposal), International Organization for Standardization.

Tansel, A. U. 1986. Adding Time Dimension to Relational Model and Extending Relational Algebra. *IEEE Transactions on Knowledge and Data Engineering 11*, 4, 343–355.

Tansel, A. U., Clifford, J., Gadia, S. K., Jajodia, S., Segev, A., and Snodgrass, R. T. Eds. 1993. *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings.

Toman, D. 1996. Point vs. Interval-based Query Languages for Temporal Databases. In *ACM Symposium on Principles of Database Systems* (1996), pp. 58–67.

Toman, D. 1997. Point-based Temporal Extensions of SQL. In *International Conference on Deductive and Object-Oriented Databases* (1997), pp. 103–121.

Toman, D. 1998. Point-Based Temporal Extensions of SQL and Their Efficient Implementation. In O. Etzion, S. Jajodia, and S. Sripada Eds., *Temporal Databases: Research and Practice* (1998), pp. 211–237. Springer LNCS State-of-the-Art Survey.

Toman, D. and Niwinski, D. 1996. First-order queries over temporal databases inexpressible in temporal logic. In *International Conference on Extending Database Technology* (1996), pp. 307–324.

TUZHILIN, A. AND CLIFFORD, J. 1990. A Temporal Relational Algebra as a Basis for Temporal Relational Completeness. In *International Conference on Very Large Data Bases* (1990), pp. 13–23.

VAN GELDER, A. AND TOPOR, R. W. 1991. Safety and Translation of Relational Calculus Queries. *ACM Transactions on Database Systems 16*, 2 (June), 235–278.

WIDOM, J. AND YANG, J. 1998. Maintaining temporal views over non-temporal information sources for data warehousing. In *International Conference on Extending Database Technology* (1998), pp. 389–403.